Behavioral Fault Modeling in a VHDL Synthesis Environment

A Dissertation

Presented to the Faculty of the School of Engineering and Applied Science University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy (Electrical Engineering)

by

Ronald J. Hayne

DISTRIBUTION STATEMENT A Approved for Public Release

May 1999

Distribution Unlimited

19990701 057

DTIC QUALITY INSPECTED 4

Approved for public release; distribution is unlimited.

© Copyright by Ronald J. Hayne All rights reserved May 1999

•

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Electrical Engineering)

Ronald J. Hayne

This dissertation has been read and approved by the Examining Committee:

Barry W. Johnson, Dissertation Advisor

Joanne Bechta Dugan, Committee Chair

110 CC

John C. Knight, Minor Representative

Juleson

Stephen G. Wilson

Han

Mircea R. Stan

Accepted for the School of Engineering and Applied Science:

R.W. M.C.2

Dean, School of Engineering and Applied Science

May 1999

Abstract

Integrated circuit designs continue to increase in both size and complexity, making fault simulation and testing more difficult and costly. Computer aided design tools and hardware description languages are now commonly used to represent designs at higher levels of abstraction. However, fault simulation and testing of digital circuits have been historically done using fault models at the gate level or below. A design methodology is needed for performing fault simulation throughout the design process, incorporating fault models at higher levels of abstraction. Use of these higher level fault models has the promise of reducing complexity, providing earlier identification of potential problems, and improving integration of fault simulation into the overall design process.

Previous behavioral fault models lack a well defined link to the hardware which they attempt to describe. Though some relationships to possible hardware faults are proposed, there is no detailed analysis to justify these assertions. Approaches based on perturbing language constructs, such as *ADD* to *SUB*, do not accurately reflect underlying hardware faults. In order to compensate for this "big micro-operation problem," alternate methods such as heuristics are used to supplement test vector sets to increase the equivalent gate level fault coverage.

This dissertation proposes a new set of fault models for VHDL behavioral descriptions of combinational logic circuits. These fault models exploit hardware relationships that exist in a design environment which involves synthesis of behavioral descriptions into gate level circuits. A functional analysis technique is used to evaluate the effects of industry standard *single-stuck-line (SSL)* faults on gate level implementations. The generalized functional faults are then abstracted into the behavioral domain by examining their relationship with the higher level language construct.

Test vectors derived from the new behavioral fault models are applied to synthesized gate level realizations of a range of circuits that include typical arithmetic and logic functions. Resulting gate level fault coverage is determined via fault simulation and used as a measure of effectiveness for the new fault models. Because the behavioral faults are derived from a functional analysis of low level faults, they provide improved fault coverage over previous fault models, over a broad range of implementations.

Acknowledgments

I would like to thank the United States Army for sponsoring my continuing education. My advisor, Dr. Barry W. Johnson, deserves a special thanks for his encouragement and support. His technical leadership helped shape many of the ideas embodied in this dissertation. Along with Dr. Johnson, I would also like to thank the other members of my advisory committee, Dr. Joanne Bechta Dugan, Dr. John C. Knight, Dr. Stephen G. Wilson, and Dr. Mircea R. Stan. Their guidance and feedback was invaluable throughout my research. My gratitude is extended to Dr. D. Todd Smith and Todd A. DeLong for serving as a sounding board and sanity check for many of my ideas. Thanks also to the United States Air Force Rome Laboratory, specifically Dr. Warren H. Debany Jr. and James P. Hanna, for additional comments and perspectives. The interactions of Dr. Robert H. Klenke, Dr. Lori M. Kaufman, Dr. James H. Aylor, and Dr. Ronald D. Williams are also greatly appreciated.

I owe a debt of gratitude to the many individuals who have helped me throughout my program. A special thanks is extended to the following individuals: Maximo Salinas, Carl Elks, Dave Garrett, Michael Reynolds, Ryan Baucom, Wes Dungan, and Erik Laurila. Thanks to the EE computer system administrators: Melissa Thrush and Diane Calleson. Thanks also to the secretaries: Lisa Sites, Peggy McCauley, Susan Malone, and Robbie Burton.

My family has provided endless love and support. The biggest thanks to my wife, Marva, for always being there for me. Her encouragement and endless optimism helped me deal with the ups and downs of the past few years. Thanks also to my sons, James and Kyle, for trying to understand all the hours spent in the conquest of this goal. My brother, Tomas, has also been a big source of encouragement. Our weekly outings to the gym helped relieve the stress inherent in such an endeavor. Finally, thanks to my parents, Jim and Audrey, for everything. They brought me up to believe that I could achieve anything I put my mind to.

Biography

Ronald J. Hayne is a Lieutenant Colonel in the U.S. Army with over 18 years of service in a variety of command and staff positions. He received his B.S. in 1980 from the United Stated Military Academy at West Point and was stationed with the 25th Infantry Division in Hawaii. He received his M.S. in Electrical Engineering in 1987 from the University of Arizona, followed by a teaching assignment with the Department of Electrical Engineering and Computer Science at West Point. LTC Hayne is now a member of the Army Acquisition Corps, responsible for the research and development of new military systems, and has served with the Space and Strategic Defense Command and the Army Research Laboratory. His world travels in the military have taken him to the Republic of Korea, the Republic of the Philippines, Vienna Austria, and Kwajalein Atoll in the Republic of the Marshall Islands.

Publications

- Hayne, R.J. and B.W. Johnson, "Behavioral Fault Modeling in a VHDL Synthesis Environment," *Proceedings VLSI Test Symposium*, April 1999.
- Kaufman, L.M., R. Gretlein, and R.J. Hayne, "A Quantitative Assessment of the Application of Software Reliability to Reusable Code," *Proceedings Reliability and Maintainability Symposium*, January 1999.
- Hayne, R.J., P A. Brown, S.G. Hair and J.E. Oristian, "An Innovative Educational Application of the VHSIC Hardware Description Language," *Proceedings Frontiers in Education Conference*, July 1990.

Table of Contents

.

Table of Contents i					
List of F	List of Figures vii				
List of T	ables .		xii		
List of A	Cronyn	ns and Abbreviations	cvi		
Chapter	·1: Inti	roduction	. 1		
1.1	Previo	us Research	. 2		
1.2	Behavi	ioral Modeling	. 4		
	1.2.1	VHDL Subset	. 4		
	1.2.2	Hardware Implementation of VHDL Constructs	. 6		
1.3	Functio	onal Analysis	. 7		
1.4	Fault I	njection Using WAVES	. 8		
1.5	Compr	ehensive Examples	. 8		
1.6	Contril	butions and Future Work	. 9		
~			10		
Chapter	· 2: Pre	vious Research	10		
2.1	Fault N	Aodels	10		
	2.1.1	Functional Faults	11		
	2.1.2	Physically-Induced Faults	13		
	2.1.3	Behavior Faults	15		
	2.1.4	Model Perturbation	16		
	2.1.5	Other Research	18		
	2.1.6	Summary	22		
2.2	Fault In	njection Techniques	22		
	2.2.1	Instruction Set Architecture	22		
	2.2.2	MEFISTO	23		
	2.2.3	Hybrid Fault Emulation	23		
	2.2.4	Summary	24		
2.3	Test G	eneration Techniques	24		
	2.3.1	S-Algorithm	25		
	2.3.2	B-Algorithm	25		
	2.3.3	Other Research	26		
2.4	Conclu	isions	28		
Chapter	3: A N	lew Control Fault Model	3U		
3.1	IF-TH	EN-ELSE	30		
	3.1.1	Synthesis of a Simple Example	30		
	3.1.2	Functional Analysis	31		
	3.1.3	Alternate Implementation	33		
	3.1.4	Generalized Functional Fault Model	35		

			1
	3.1.5 Deve	elopment of a Behavioral Fault Model	35
	3.1.6 Eval	uation of the New Behavioral Fault Model	37
	3.1.7 Com	parison with Previous Behavioral Fault Models	38
	3.1.8 Expa	ansion of the Fault Model	40
	3.1.8	3.1 Functional Analysis	41
	3.1.8	3.2 Generalized Functional Fault Model	44
	3.1.8	B.3 Behavioral Fault Model	45
	3.1.8	8.4 Evaluation of the Behavioral Fault Model	46
	3.1.9 Sum	mary	47
3.2	CASE		48
<i>v.</i> _	3.2.1 Appl	lication of the Control Fault Model	49
	3.2.2 Eval	uation of the Fault Model	51
	3.2.3 Com	parison with Previous Behavioral Fault Models	53
3.3	Conclusions	1	55
Chapter	4: Relationa	I Operators	56
4.1	Greater Than	ı (GT)	56
	4.1.1 Gene	eralized Functional Faults	57
	4.1.2 Class	sification of Functional Faults	58
	4.1.3 Beha	avioral Fault Model	60
	4.1.4 Adap	oting the Model for GE, LT, and LE	62
	4.1.5 Sum	mary	63
4.2	Threshold De	etection	63
	4.2.1 Grea	ter Than Signed Threshold	63
	4.2.2 A Qu	uick Example	64
4.3	Equal (EQ)		66
	4.3.1 Func	tional Faults	66
	4.3.2 Beha	avioral Fault Model	67
4.4	Comparison	with Previous Fault Models	68
4.5	Application of	of the New Fault Models	69
	4.5.1 Fault	ts on Relational Operators	69
	4.5.2 Cont	rol Faults	
	4.5.3 Final	Behavioral Test Vector Set	73
4.6	Evaluation of	f Behavioral Test Vectors	
	4.6.1 Gate	Level Realizations	
	4.6.2 Expa	ansion of the Data Path	
	4.6.3 Expa	ansion of the Control Signals	
4.7	Conclusions		80
Chapter	5: Arithmet	ic Operators	82
5.1	Addition		
	5.1.1 Ripp	le Carry Adder	
	5.1.1	.1 Functional Testing	
	5.1.1	.2 Scalability	
	5.1.1	.3 Behavioral Fault Model	86

,

•

ii

		5.1.1.4	Evaluation of the Behavioral Test Vectors	
		5.1.1.5	Carry-in and Carry-out	
	5.1.2	Carry Lo	ook-Ahead Adder	
		5.1.2.1	Functional Testing	
		5.1.2.2	Application of the Behavioral Test Vectors	
		5.1.2.3	Scalability	
		5.1.2.4	Optimization of CLA Behavioral Faults	
	5.1.3	Summar	V	100
5.2	Subtra	action	٠	101
	5.2.1	Direct S	ubtraction	101
		5.2.1.1	Functional Testing	101
		5.2.1.2	Application of the Behavioral Test Vectors	102
	5.2.2	Subtract	ion Using Addition Circuitry	103
	5.2.3	Summar	y	104
5.3	Consta	ants as Ope	erands	104
	5.3.1	Function	al Testing	105
	5.3.2	Generali	zed Behavioral Fault Model	105
5.4	Comp	arison with	Previous Fault Models	106
5.5	Conclu	usions		107
Chapter 6.1	r 6: Otl Logica	her Opera al Operator	tors s	109
	6.1.1	AND/OF	۲	110
		6.1.1.1	Functional Faults	110
		6.1.1.2	Complex Expressions	111
		6.1.1.3	Scalability	113
		6.1.1.4	Behavioral Fault Model	113
		6.1.1.5	Application of the New Fault Models	114
	6.1.2	XOR		118
		6.1.2.1	Functional Faults	118
		6.1.2.2	Optimized Test Generation	119
		6.1.2.3	Evaluation of the Generalized Test Vectors	122
6.0	6.1.3	Compari	son with Previous Fault Models	
6.2	Unary	Operators	*7 1	
	6.2.1	Absolute	Value	
	6.2.2	Negation		
	0.2.3	Generaliz	zed Functional Faults	
	0.2.4	Benavior	al Fault Model	
()	0.2.3 Camala	Evaluatio	on of Benavioral lest vectors	
0.3	Conclu	isions		
Chapter	7: Oth	ner Progra	mming Constructs	129
7.1 [.]	Loops.			
	/.1.1	A Simple	Example	
	1.1.2	Comparis	son with Previous Fault Models	

•

iii

7.2	Functions and Procedures	134
	7.2.1 Example ADD4fn	135
	7.2.2 Example ADD4pr	136
7.3	Conclusions	137
Chapter	· 8: Comprehensive Examples	138
8.1	Arithmetic Logic Unit	138
0.1	811 Example ALU4wc	138
	8 1 1 1 Faults on Logical Operators	
	8 1 1 2 Faults on Arithmetic Operators	
	8 1 1 3 Control Faults	
	8 1 1 4 Final Behavioral Test Vector Set	
	8.1.2 Evaluation of the Behavioral Test Vectors	
	813 Expansion of the Data Path	
	814 Summary	
82	Error Correcting Circuit	
0.2	8.2.1 Example HAMMING4	
	8 2 1 1 Faults on XOR-only Expressions	
	8 2 1 2 Faults on Other Logical Expressions	159
	8 2 1 3 Final Behavioral Test Vector Set	162
	8.2.2 Evaluation of the Behavioral Test Vectors	163
	8.2.3 Expansion of the Data Path	163
	8.2.4 Summary	166
83	Conclusions	166
010		
Chapter	9: Conclusions and Future Work	167
9.1	Research Contributions	
	9.1.1 Generalized Functional Faults	
	9.1.2 New Behavioral Fault Models	
	9.1.3 Gate Level Fault Coverage of Behavioral Test Vectors	
	9.1.4 Behavioral Test Generation	
	9.1.5 Behavioral Fault Simulation	
9.2	Future Work	170
	9.2.1 Expansion of Behavioral Fault Models	
	9.2.2 Tool Development	
	9.2.3 Higher Levels of Abstraction	
9.3	Concluding Remarks	172
eference	ces	173
nnendi	ix A: Additional Examples	
A.1	Array Indexing	
A.2	Generalization of the Control Fault Model	
A.3	Signed Comparison	185
A.4	Unsigned Threshold	188

•

A.5	Adder/Subtractor	
	A.5.1 Faults on Arithmetic Operators	
	A.5.2 Control Faults	190
	A.5.3 Evaluation of the Behavioral Test Vectors	191
	A.5.4 CLA Implementation	191
A.6	Arithmetic with Constants	
	A.6.1 Example PLUS25	194
	A.6.2 Example MINUS25	195
A.7	XOR4	
	A.7.1 Parse Tree Test Vectors	
	A.7.2 Evaluation of Behavioral Test Vectors	
	A.7.3 Optimized Test Vectors	
	A.7.4 Evaluation of Optimized Test Vectors	
Append	lix B: Fault Experiment Results	200
Append	ix C: VHDL Behavioral Descriptions	205
C.1	Variables and Signals	205
C.2	Expressions	206
·C.3	If Statement	206
C.4	Case Statement	207
C.5	Loop Statements	208
C.6	Process Statement	
C.7	Procedures and Functions	210
Append	ix D: VHDL Synthesis	
D.1	Level-0	
D.2	Mentor Graphics	213
D.3	Synopsys	214
D.4	IEEE Draft Standard	216
Annond	in F. Handware Implementation of VHDL Constructs	210
ыррени Б 1	Structured Logic Design	·····410 710
E.1 臣う	Mentor Granhics	·····210
E. 2	Mentor Graphics	224
Append	ix F: VHDL Source Code	228
F.1	CASE1	229
F.2	ARRAY4	230
F.3	SHIFT4u	231
F.4	LESS2	232
F.5	EQUAL3	
F.6	GREATER3	
F.7	LE5	
F.8	GE23u	237
F.9	LT12u	
F.10	GT3n	

. **v**

E 11	COMPADE	240
Г.II Е 12		240 242
Г.12 Е 12		
F.13		
F.14		
F.15	ADD4	
F.16	ADD4wc	
F.17	ADD8	
F.18	SUB4	
F.19	ADDSUB4	254
F.20	INC4	256
F.21	INC8	257
F.22	DEC4	258
F.23	ADDINC4	259
. F.24	PLUS3	260
F.25	MINUS5	261
F.26	PLUS25	262
F.27	MINUS25	263
F.28	SOP1	264
F.29	SOP4	265
F.30	POS1	266
F.3 1	GT	
F.32	XOR4	
F.33	XOR5	270
F.34	ABS4	272
F.35	ABS8	273
F.36	NEG4	
F.37	NEG8	275
F.38	ALU4wc	
F.39	ALU8wc	278
F.40	HAMMING4	
F.41	HAMMING8	
	· · · · · · · · · · · · · · · · · · ·	

. . vi vi

.

List of Figures

Chapter 1		
Figure 1-1	Design guidelines	5
Chapter 2		
Figure 2-1	Three-input majority circuit <i>M</i>	14
Chapter 3		
Figure 3-1	Behavioral description for example IF1.	30
Figure 3-2	Synthesized hardware for example IF1	31
Figure 3-3	SOP gate level circuit for MUX21	31
Figure 3-4	Karnaugh map for MUX	33
Figure 3-5	POS gate level MUX.	34
Figure 3-6	Control fault model for if-then-else.	37
Figure 3-7	Behavioral description for example IF2.	40
Figure 3-8	Synthesized hardware for example IF2	41
Figure 3-9	Gate level circuit for example IF2.	41
Figure 3-10	Control fault model for expanded if-then-else	45
Figure 3-11	Equivalent if-then-else and case statements.	49
Figure 3-12	Logical adjacencies among clauses.	50
Figure 3-13	WHEN-CORRUPT (AND) faults for example CASE1	50
Figure 3-14	Synthesized Structure1 for example CASE1	52
Figure 3-15	WAVES test vectors for example CASE1	52
Figure 3-16	Synthesized Structure2 for example CASE1	53
Figure 3-17	Fault coverage for Structure2 of example CASE1.	53
Chapter 4		
Figure 4-1	Karnaugh map for 2-bit GT function	57
Figure 4-2	Fault classes for 2-bit GT function.	59
Figure 4-3	Fault classes for 3-bit GT function.	60
Figure 4-4	Fault classes for 2-bit GE function.	62
Figure 4-5	Functional test vectors for signed GT threshold.	64
Figure 4-6	Behavioral description for example LE5.	65
Figure 4-7	Behavioral test vectors for example LE5.	65
Figure 4-8	Synthesized circuit for example LE5	65
Figure 4-9	Fault classes for 3-bit EQ function.	66
Figure 4-10	Behavioral description for example COMPARE	69
Figure 4-11	Synthesized Structure1 for example COMPARE.	75
Figure 4-12	Alternate set of behavioral test vectors for example COMPARE	76
Figure 4-13	Synthesized Structure2 for example COMPARE.	76

Chapter 5

Figure 5-1	Ripple carry adder	83
Figure 5-2	Behavioral fault model for ripple carry adder	
Figure 5-3	Behavioral description for example ADD4.	88
Figure 5-4	Synthesized circuit for example ADD4.	88
Figure 5-5	NOR-only realization of example ADD4.	89
Figure 5-6	Behavioral description for example ADD4wc	
Figure 5-7	Carry look-ahead adder	
Figure 5-8	CLA implementation of example ADD4	
Figure 5-9	Ripple carry implementation of example ADD8	
Figure 5-10	Fault coverage for ripple carry ADD8.	
Figure 5-11	Block diagram of modular CLA adder.	
Figure 5-12	Fault coverage for modular CLA adder	
Figure 5-13	Behavioral description for example SUB4.	102
Figure 5-14	Synthesized circuit for example SUB4	103
Figure 5-15	Subtractor implemented with full adders	103

Chapter 6

Figure 6-1	Behavioral description for example SOP1	111
Figure 6-2	Binary tree representing example SOP1	111
Figure 6-3	WAVES test vectors for example SOP4	113
Figure 6-4	Behavioral fault model for example SOP1	114
Figure 6-5	Behavioral description for example GT	114
Figure 6-6	Parse tree for example GT.	115
Figure 6-7	Synthesized Structure1 for example GT	
Figure 6-8	Synthesized Structure2 for example GT.	117
Figure 6-9	Fault coverage for Structure2 of example GT.	118
Figure 6-10	Labeling scheme for Bossen test	119
Figure 6-11	Structure Cascade1 for example XOR5	120
Figure 6-12	Structure Cascade2 for example XOR5	120
Figure 6-13	Modified Bossen test for Cascade2	121
Figure 6-14	Generalized Bossen test vectors for example XOR5	122
Figure 6-15	Structure4 for example XOR5.	122
Figure 6-16	Fault coverage for Structure4 of example XOR5.	123
Figure 6-17	Behavioral fault model for ABS.	126
Figure 6-18	WAVES test vectors for example ABS8.	126
Figure 6-19	Synthesized Structure1 of example ABS8	127
Figure 6-20	Fault coverage for Structure1 of example ABS8	127
	-	

viii

Figure 6-21	Synthesized circuit for example NEG8	128

Chapter 7

Figure 7-1	Behavioral description for example SHIFT4u	129
Figure 7-2	Expanded case statement for example SHIFT4u.	130
Figure 7-3	WAVES test vectors for example SHIFT4u	, 133
Figure 7-4	Synthesized Structure1 for example SHIFT4u.	133
Figure 7-5	Fault coverage for Structure1 of example SHIFT4u	134
Figure 7-6	Behavioral description for example ADD4fn	135
Figure 7-7	Functions for example ADD4fn	135
Figure 7-8	Expanded behavioral description for example ADD4fn	136
Figure 7-9	Behavioral description for example ADD4pr	136
Figure 7-10	Procedure FA for example ADD4pr.	137

Chapter 8

Figure 8-1	Entity description for example ALU4wc	138
Figure 8-2	Architecture description for example ALU4wc	139
Figure 8-3	Logical adjacencies among clauses.	147
Figure 8-4	WAVES test vectors for example ALU4wc	
Figure 8-5	Synthesized Structure1 for example ALU4wc.	153
Figure 8-6	Fault coverage for Structure1 of example ALU4wc	153
Figure 8-7	WAVES test vectors for example ALU8wc.	154
Figure 8-8	Entity description for example HAMMING4	156
Figure 8-9	Architecture description for example HAMMING4	157
Figure 8-10	Structure Cascade1 for 4-input XOR-only expression	157
Figure 8-11	Structure Cascade2 for 4-input XOR-only expression	157
Figure 8-12	Parse tree for expression D(1)	159
Figure 8-13	WAVES test vectors for example HAMMING4	
Figure 8-14	Synthesized Structure1 for example HAMMING4.	163
Figure 8-15	Entity description for example HAMMING8	164
Figure 8-16	Architecture description for example HAMMING8	164
Figure 8-17	WAVES test vectors for example HAMMING8	165

Chapter 9

Appendix A

Figure A-1	Behavioral description for example ARRAY4	
Figure A-2	Equivalent case statement for example ARRAY4	
Figure A-3	Logical adjacencies among clauses.	
Figure A-4	WAVES test vectors for example ARRAY4	
Figure A-5	Synthesized circuit for example ARRAY4.	
Figure A-6	Behavioral description for example CASE2.	
Figure A-7	Behavioral description for example GREATER3.	
-		

ix

.

Figure A-8	Fault classes for 3-bit signed GT function.	186
Figure A-9	WAVES test vectors for example GREATER3.	186
Figure A-10	Synthesized Structure1 for example GREATER3.	187
Figure A-11	Synthesized Structure3 for example GREATER3.	188
Figure A-12	Behavioral description for example GE23u.	188
Figure A-13	Behavioral test vectors for example GE23u.	189
Figure A-14	WAVES test vectors for example GE23u.	189
Figure A-15	Synthesized circuit for example GE23u.	189
Figure A-16	Behavioral description for example ADDSUB4.	189
Figure A-17	WAVES test vectors for example ADDSUB4.	190
Figure A-18	Synthesized circuit for example ADDSUB4	191
Figure A-19	CLA test vectors for example ADDSUB4.	193
Figure A-20	CLA implementation of example ADDSUB4	193
Figure A-21	Behavioral test vectors for example PLUS25	194
Figure A-22	Synthesized Structure1 for example PLUS25	194
Figure A-23	Parse tree for example XOR4.	196
Figure A-24	WAVES test vectors for example XOR4.	197
Figure A-25	Structure1 for example XOR4	197
Figure A-26	Structure2 for example XOR4	198
Figure A-27	Structure3 for example XOR4	198
Figure A-28	Structure4 for example XOR4	198
Figure A-29	Structure Cascade1 for example XOR4.	199
Figure A-30	Structure Cascade2 for example XOR4.	199

Appendix B

Appendix C

Appendix D

.

Figure D-1	Process types for Level-0.	
Figure D-2	Design restrictions for Synopsys	

Appendix E

Figure E-1	VHDL constructs that map to multiplexer elements.	219
Figure E-2	Hardware implementation for case statement.	220
Figure E-3	Hardware implementation for <i>if</i> statement.	221
Figure E-4	Hardware implementation for vector indexing	221
Figure E-5	VHDL description for a ripple carry adder.	
Figure E-6	Hardware implementation of ripple carry adder	222
Figure E-7	Using functions to represent combinational logic.	223
Figure E-8	Code example for simple <i>if</i> statement.	224
Figure E-9	Code example for <i>if-else</i> statement.	225
Figure E-10	Synthesized hardware for <i>if-else</i> statement.	225
Figure E-11	Code example for <i>case</i> statement.	226

Figure E-12	Synthesized hardware for case statement	
Figure E-13	Code example for variable index assignment.	

.

Appendix F

List of Tables

Chapter 1

.

Table 1-1	Levels of detail commonly used in design	. 2
Table 1-2	Predefined VHDL operators.	. 5

Chapter 2

Table 2-1	Minimal SIF test set	14
Table 2-2	Micro-operation Faults	18

Chapter 3

Table 3-1	SSL fault table for SOP MUX.	32
Table 3-2	Fault reductions	32
Table 3-3	Reduced fault table for SOP MUX.	33
Table 3-4	Reduced fault table for POS MUX.	34
Table 3-5	Generalized functional fault model	35
Table 3-6	SSL faults detected by behavioral test vectors.	38
Table 3-7	Stuck-data and stuck-control behavioral faults	39
Table 3-8	Faults detected by test vector Set 1	39
Table 3-9	Fault reductions for example IF2.	42
Table 3-10	Covering faults for external control line faults.	42
Table 3-11	Reduced functional faults for example IF2	43
Table 3-12	Generalized functional faults for example IF2.	44
Table 3-13	Behavioral test vectors for example IF2.	47
Table 3-14	Behavioral test vectors for example CASE1.	51

Chapter 4

Table 4-1	Reduced functional faults for SOP GT.	57
Table 4-2	Reduced functional faults for POS GT.	58
Table 4-3	Functional faults for n-bit GT.	60
Table 4-4	Behavioral faults for 2-bit GT	61
Table 4-5	Functional faults for n-bit GE.	62
Table 4-6	Behavioral faults for 2-bit GE	63
Table 4-7	Functional faults for n-bit EQ.	67
Table 4-8	Behavioral faults for 2-bit EQ	67
Table 4-9	Micro-operation Faults	68
Table 4-10	Test vectors for behavioral faults for A > B.	70
Table 4-11	Test vectors for behavioral faults for A < B.	71
Table 4-12	Test vectors for THEN-CORRUPT faults.	72
Table 4-13	Test vectors for ELSIF-CORRUPT and ELSE-CORRUPT faults	73
Table 4-14	Final behavioral test vector set for example COMPARE	74

Chapter 5

•

Table 5-1	Truth table for full adder	83
Table 5-2	Phase I functional tests	84
Table 5-3	Phase II functional tests.	85
Table 5-4	Phase III functional tests	85
Table 5-5	Functional tests for 4-bit ripple carry adder	86
Table 5-6	Functional tests for example ADD4wc	
Table 5-7	Carries for 4-bit CLA adder	
Table 5-8	Missing carry faults	
Table 5-9	Extra carry faults	
Table 5-10	Additional behavioral test vectors for CLA adder	
Table 5-11	Behavioral test vectors for extra carry faults	
Table 5-12	Behavioral test vectors for missing carry faults	
Table 5-13	Optimized test vectors for missing carry faults	100
Table 5-14	Optimized test vectors for extra carry faults	100
Table 5-15	Truth table for full subtractor	101
Table 5-16	Functional tests for 4-bit direct subtraction.	102
Table 5-17	Functional tests for adder by subtraction test vectors	104
Table 5-18	Functional tests for 4-bit increment function	105
Table 5-19	Functional tests for 6-bit function Z <= Y - 5	106
Table 5-20	Micro-operation Faults	107

Chapter 6

.

Functional faults for AND operation.	110
Functional faults for OR operation.	110
Functional test vectors for example SOP1	112
Reduced test vectors for example SOP1	113
Behavioral test vectors for example GT.	116
Reduced test vectors for example GT	117
Generalized functional faults for XOR operation	119
Bossen test vectors for Cascade1	120
Modified Bossen test vectors for Cascade2	122
Reduced functional faults for 4-bit ABS.	. 124
Reduced functional faults for 4-bit negation.	125
Generalized functional faults for absolute value and negation	125
	Functional faults for AND operation. Functional faults for OR operation. Functional test vectors for example SOP1. Reduced test vectors for example GT. Reduced test vectors for example GT. Generalized functional faults for XOR operation. Bossen test vectors for Cascade1. Modified Bossen test vectors for Cascade2. Reduced functional faults for 4-bit ABS. Reduced functional faults for 4-bit negation. Generalized functional faults for absolute value and negation.

Chapter 7

Table 7-1	Behavioral faults for example SHIFT4u	131
-----------	---------------------------------------	-----

Chapter 8

Table 8-1	Behavioral test vectors for OR operator.	141
Table 8-2	Behavioral test vectors for remaining Logical Functions	142

Table 8-3	Behavioral tests for 4-bit ADD with carry	143
Table 8-4	Possible test vectors for Arithmetic Functions.	143
Table 8-5	Control faults for Logical Functions	145
Table 8-6	Control faults for Arithmetic Functions	146
Table 8-7	THEN-CORRUPT control faults	148
Table 8-8	ELSE-CORRUPT control faults.	149
Table 8-9	Final behavioral test vectors for Logical Functions	150
Table 8-10	Final behavioral test vectors for Arithmetic Functions	151
Table 8-11	ALU fault experiment results	155
Table 8-12	Optimized test vectors for 4-input XOR-only expression	158
Table 8-13	Optimized test vectors for expression S(1)	158
Table 8-14	Optimized test vectors for expression S(2)	158
Table 8-15	Optimized test vectors for expression S(3)	159
Table 8-16	Reduced test vector set for XOR-only expressions.	159
Table 8-17	Behavioral test vectors for expression D(1)	160
Table 8-18	Reduced test vectors for expression D(1).	160
Table 8-19	Reduced test vectors for expression D(2).	161
Table 8-20	Reduced test vectors for expression D(3).	161
Table 8-21	Reduced test vectors for expression D(4).	161
Table 8-22	Test vectors for logical expressions D(1) through D(4)	162
Table 8-23	HAMMING fault experiments	166

Chapter 9

.

Appendix A

Table A-1	Behavioral faults for example ARRAY4.	181
Table A-2	Behavioral faults and corresponding test vectors for example CASE2.	.184
Table A-3	Coverage for control faults.	191
Table A-4	Missing carry faults for subtraction.	192
Table A-5	Extra carry faults for subtraction.	192
Table A-6	Functional tests for example MINUS25.	195
Table A-7	Behavioral test vectors for example XOR4.	196
Table A-8	Optimized test vectors for example XOR4	199

Appendix B

Table B-1	Control fault experiments.	200
Table B-2	Relational operator fault experiments.	
Table B-3	Arithmetic operator fault experiments.	201
Table B-4	Other operator fault experiments.	
Table B-5	Comprehensive fault experiments.	204

Appendix C

Table C-1	Predefined VHDL operator	
-----------	--------------------------	--

xiv

Appendix D

Appendix E

Table E-1	Truth table for process MUX2.	
-----------	-------------------------------	--

Appendix F

.

List of Acronyms and Abbreviations

ABS	Absolute Value
ALU	Arithmetic Logic Unit
ATPG	Automated Test Pattern Generation
BTG	Behavioral Test Generator
CLA	Carry Look-Ahead
CSIS	Center for Semicustom Integrated Systems (University of Virginia)
DEC	Decrement
EQ	Equal
FA	Full Adder
FF	Flip-Flop
FPGA	Field Programmable Gate Array
FS	Full Subtractor
GE	Greater than or Equal to
GT	Greater Than
HA	Half Adder
HS	Half Subtractor
IEEE	Institute of Electrical and Electronics Engineers
INC	Increment
ISA	Instruction Set Architecture
LE	Less than or Equal to
LFSR	Linear Feedback Shift Register
LT	Less Than
MEFISTO	Multi-level Error/Fault Injection Simulation Tool
MIL-STD	Military-Standard
MOD	Modulus
MUX	Multiplexer
NE	Not Equal
POS	Product-of-Sums
RAM	Random Access Memory

REM	Remainder
ROM	Read Only Memory
RTL	Register Transfer Level
SIF	SSL-Induced Fault
SOP	Sum-of-Products
SSL	Single-Stuck-Line
SUB	Subtraction
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration
WAVES	Waveform and Vector Exchange
XOR	Exclusive Or

.

.

•

.

Chapter 1 Introduction

Integrated circuit designs continue to increase in both size and complexity. Fault simulation and testing of these designs is, thus, becoming more difficult and costly. Designers now commonly make use of computer aided design tools and hardware description languages, such as VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language), to represent their designs at higher levels of abstraction. However, fault simulation and testing of digital circuits for manufacturing faults have been historically done using fault models at the gate level or below. Use of these lower level fault models adds complexity and delays these efforts to later in the design cycle.

There is a need to develop a design methodology for performing fault simulation throughout the design process, at many levels of abstraction. It is, therefore, desirable to develop fault models at higher levels of abstraction, based on functional or behavioral descriptions. Working with behavioral fault models will also allow fault simulation to be performed earlier in the design scheme, without details of the gate level implementation. In fact, depending on the source of the component, a gate level description may never be available. Thus, fault simulation and testing based on these higher level fault models have the promise of being less complex, providing earlier identification of potential problems, and improving integration into the overall design process.

This dissertation develops a new set of fault models for VHDL behavioral descriptions of combinational logic circuits. The fault models exploit hardware relationships that exist in a design environment which involves synthesis of behavioral descriptions into gate level circuits. The focus is on relating high level language constructs to lower level hardware faults, as opposed to perturbing the language elements as if they were simply software. A functional analysis technique is used to evaluate the effects of industry standard *single-stuck-line (SSL)* faults on gate level implementations. The generalized functional faults are then abstracted into the behavioral domain by examining their relationship with the higher level language construct. The resulting behavioral fault models are, thus, more strongly linked to underlying hardware faults than those developed by previous research.

1

As part of this research, test vectors derived from the new behavioral fault models are applied to synthesized gate level realizations of a range of circuits that include typical arithmetic and logic functions. Resulting gate level *SSL* fault coverage is determined via fault simulation and used as a measure of effectiveness for the new fault models. Because the behavioral faults are derived from a functional analysis of low level faults, they provide improved fault coverage over previous fault models, over a broad range of implementations.

1.1 Previous Research

Recent research efforts have attempted to develop fault models at higher levels of abstraction, which accurately represent faults which occur at lower levels. The models of interest in this research move up the design hierarchy, shown in Table 1-1 [10], beyond the gate level to the register or chip level. Functional fault models are based on the input/output relationship of higher level primitives which may incorporate a large number of gates. Behavioral fault models are based on procedural descriptions of the circuits desired function. Many models reference the register transfer level which describes procedural data flow among functional primitives.

Level of Detail	tail Behavioral Domain Representation Primitives		
System	Performance specification	Computer, disk, unit, radar	
Chip	Algorithm Microprocessor, RAM, ROM, U		
Register Data flow Re		Register, ALU, COUNTER, MUX	
Gate	Boolean equations	AND, OR, XOR, FF	
Circuit	Differential equations Transistor, R, L, C		
Layout/silicon	None	Geometrical Shapes	

Table 1-1 Levels of detail commonly used in design.

The majority of functional and behavioral modeling efforts can be traced to four prominent researchers: Jacob Abraham, James Armstrong, Sumit Ghosh, and John Hayes. Chapter 2 of this dissertation examines the models developed by groups including each of these researchers. Modeling at many levels of abstraction is discussed by Abraham et al. [1][2][17][65][66], but the main focus of this research concerns the functional fault models. Armstrong and his collaborators [6][7][8][9][22][28][69] have developed an ever evolving series of functional and behavioral fault models and have implemented test generation algorithms using these models. Ghosh and Chakraborty [18][26][27] have proposed a set of fault models which are based on the failure modes of the language constructs of a generic hardware description language. Finally, Hayes et al. [29][31][32] [33][34][63] has worked extensively on fault models for digital circuits, including descriptions of functional fault models leading to a new generic class called *induced* faults.

The survey of the state of the art in high level fault modeling clearly indicates that there is no widely accepted solution to the problem. Modeling techniques range from the functional analysis employed by Abraham and Hayes, to the procedural data flow of behavioral descriptions used by Ghosh and Armstrong. Previous behavioral fault models lack a well defined link to the hardware which they attempt to describe. Though some relationships to possible hardware faults are proposed, there is no detailed analysis to justify these assertions. Approaches based on perturbing language constructs, such as *ADD* to *SUB*, do not accurately reflect underlying hardware faults. In order to compensate for this "big micro-operation problem," alternate methods such as heuristics are used to supplement test vector sets to increase the equivalent gate level fault coverage.

Though previous research provides no clear cut solution to modeling faults at higher levels of abstraction, valuable insights are gained by the examination of each of these techniques. Certain key concepts from past efforts have immediate applicability here, notably *functional equivalence* and *fault dominance*. Further, the behavioral fault models developed in this dissertation only affect the *activation* step of the test generation process. Hence, the high level algorithms developed to handle the computationally intensive tasks of fault *propagation* and *justification* still remain valid. Integration of new fault models with an existing behavioral test generation algorithms already address problems such as reconvergent fanout, while use of more complex fault models can eliminate the need to supplement test vector sets via heuristics.

3.

1.2 Behavioral Modeling

Hardware description languages can be used to model system behavior at higher levels of abstraction than the traditional gate or circuit level. Languages like VHDL make use of sequential statements, much like conventional programming languages, to describe the desired behavior of a circuit. Several constructs such as *if-then-else* and *case* are normally provided to control the order of execution of these sequential statements. Designers can, therefore, use VHDL to develop behavioral models that can be simulated to verify their correct functioning, prior to generating hardware. VHDL behavioral descriptions and language constructs are detailed in Appendix C.

Modern synthesis tools interpret the behavioral VHDL source code as a description of an electronic circuit. Not all language constructs map directly to hardware in a synthesis environment, therefore, a language subset is defined for use with a specific tool. General modeling guidelines are normally provided to ensure that resulting hardware will be consistent with the designer's intent. Combining these guidelines with the VHDL subset, higher level models can be synthesized to create optimized gate level descriptions.

1.2.1 VHDL Subset

The VHDL behavioral models used in this research describe combinational logic circuits based on the *IEEE Draft Standard for VHDL Register Transfer Level Synthesis* [36]. The draft builds on the foundation laid by the European VHDL Synthesis Working Group's *Level-0 VHDL Synthesis Syntax and Semantics* [25] and incorporates constructs common to synthesis tools by Mentor Graphics [68] and Synopsys [64]. Details can be found in Appendix D.

The standard represents a subset of VHDL with corresponding design guidelines meant to ensure consistent synthesis of gate level netlists from behavioral descriptions. The key VHDL language constructs supported for behavioral modeling are listed below:

- 1) if statement, case statement, loop statement (for only).
- 2) procedure, function.
- 3) constant, variable, signal.
- 4) all predefined VHDL operators shown in Table 1-2.

Relational operators like greater than (>) and not equal (/=) compare like types and return a Boolean result. The concatenation operator (&) combines one-dimensional arrays to form a new array with the contents of the right operand following the contents of the left operand. Finally, the modulus (MOD), remainder (REM), exponentiation (**), and absolute value (ABS) operators are only defined for integer types. Definitions for all the VHDL operators can be found in Appendix C.

Туре			Oper	ators		
Logical	AND	OR	NAND	NOR	XOR	
Relational	=	/=	<	<=	>	>=
Adding	+	-	&			
Unary (sign)	+	-				
Multiplying	*	/	MOD	REM		
Miscellaneous	**	ABS	NOT			

 Table 1-2 Predefined VHDL operators.

Design guidelines for modeling combinational logic circuits, summarized in Figure 1-1, are extracted from those defined by Level-0 and Synopsys. Though the guidelines for

- Processes which model pure combinational logic must contain a sensitivity list including all the *signals* which are read into the process. All *signals* and *variables* must be assigned in all the conditional branches.
- 2) Integer types are automatically converted to bit vectors whose width is as small as possible to accommodate all possible values of the type's range, either in unsigned binary for nonnegative ranges or in 2's complement form for ranges that include negative numbers.
- 3) The arithmetic operators "+" and "-" are predefined for all integer operands.
- 4) *Multiplying operators* ("*", "/", *mod*, and *rem*) are predefined for all integer types with the following restrictions:

a) The right *operand* shall be a computable power of 2.

b) Neither operand shall be negative.

multiplying operators may seem overly restrictive, they still provide adequate flexibility for designing at the register transfer level and may yet be expanded in subsequent revisions to the standard.

Use of this subset is meant to enhance the portability of VHDL designs across synthesis tools conforming to the standard. Hence, it is used here as the basis for defining higher level fault models which have a closer relationship to resulting synthesized hardware. Behavioral fault models for each of these VHDL constructs are developed in Chapter 3 through Chapter 7 of this dissertation.

1.2.2 Hardware Implementation of VHDL Constructs

Several VHDL language constructs lend themselves directly to hardware implementation with common functional modules such as multiplexers. By examining these language to hardware relationships, this research intends to build the foundation on which higher level fault models can be defined, that are more closely related to their underlying gate level counterparts. Some insights are drawn from two resources which directly discuss the relationship between certain VHDL constructs and the ultimate hardware.

One discussion of hardware implementation of VHDL constructs comes from *Structured Logic Design with VHDL* by Armstrong and Gray [10]. In a section titled "Automated Synthesis of VHDL Constructs," they show the relationship between multiplexers and language constructs that involve selection, like *if* and *case*. Another insight into the relationship between VHDL language constructs and hardware comes from the *VHDL Style Guide for AutoLogic II* by Mentor Graphics [68]. Again, the link is established between the control constructs *if* and *case* and the multiplexer functional building block. These examples reinforce the intuition that a language construct that involves selection leads naturally to a hardware construct that implements selection, the multiplexer. Details can be found in Appendix E.

Armstrong and Gray also discuss program *loops*, *functions*, and *procedures* in relation to hardware [10]. Multiple implementations of a 4-bit adder are used for illustration. Their discussions and examples serve as the basis for the analysis of these programming constructs in Chapter 7 of this dissertation.

1.3 Functional Analysis

Previous research has proposed fault models for behavioral constructs based solely on perturbing the language without a well defined link to the underlying hardware. This dissertation presents new behavioral fault models based on a functional analysis of gate level implementations. By combining VHDL synthesis information with industry standard *SSL* faults, new fault models can be obtained which are more closely linked to the underlying hardware.

A technique has been developed for abstracting SSL faults from the gate level into the behavioral domain. First, synthesis information about hardware implementation of VHDL constructs is exploited to obtain a gate level basis for a functional analysis. Next, a reduced set of functional faults, covering all SSL gate level faults, is determined with the aid of fault reductions via *functional equivalence* and *fault dominance* [45]. Faults are generalized from various possible implementations to form a set of functional faults not tied to any specific realization. Most importantly, a detailed analysis of the relationship between the generalized set of functional faults and the original VHDL description yields a behavioral fault model for the language construct.

The new fault models developed by this research provide the well defined link to underlying hardware faults that was lacking in previous behavioral fault models. The functional analysis of *SSL* faults takes advantage of VHDL/hardware relationships that exist in a synthesis environment. This analysis of gate level faults adapts the functional techniques employed by Abraham [2] and Hayes [29] with two important extensions. First, functional faults are not tied to a specific implementation, rather they are generalized to be applicable to multiple realizations. Second, the functional faults are further abstracted into the behavioral domain via their relationship with the original VHDL construct being modeled.

Details of the functional analysis technique are presented during the development of the control fault model in Chapter 3. Graphical techniques for examining the relationships between functional faults are first demonstrated with *relational operators* in Chapter 4. Complete functional testing of regular structures of functional building blocks is introduced during the analysis of *arithmetic operators* in Chapter 5. Finally, interactions among behavioral faults and VHDL constructs are explored throughout the dissertation, but especially in Chapter 7 and Chapter 8.

1.4 Fault Injection Using WAVES

Gate level fault injection experiments were performed throughout this research using a tool developed by DeLong et al. [24]. Test vectors are applied to structural VHDL models using the *IEEE Standard for Waveform and Vector Exchange (WAVES)* [37]. *SSL* fault simulations are determined using gate level equivalent fault classes according to MIL-STD 883D [52]. Test vectors are applied, one at a time, from the input WAVES file. Fault coverage is then evaluated as the ratio of detected faults to total faults and can be plotted versus the test vectors as they are applied.

Fault experiments using behavioral test vectors were conducted during the development of each group of behavioral fault models. The normal requirement for industrial designs is that the set of test vectors provided by the designer achieve at least a 95% *SSL* gate level fault coverage [27][40][60]. In order to more fully examine the effectiveness of the new fault models, additional examples, not presented in the individual chapters, are provided in Appendix A. Examples have been chosen to represent a broad range of design possibilities. Resulting gate level fault coverage was evaluated to illustrate the effectiveness of the behavioral fault models and is summarized in Appendix B.

1.5 Comprehensive Examples

Two comprehensive example circuits are presented in Chapter 8 to demonstrate the gate level fault coverage of the new behavioral fault models. The first is an arithmetic logic unit (ALU) which performs selected functions on data inputs. The second example is a single error correcting circuit used in fault tolerant applications. Other obvious examples such as a multiplexer or a magnitude comparator do not need to be investigated here due to their detailed analysis as part of the development of the fault models for the *if* statement and *relational operators*.

Application of the behavioral fault models to the comprehensive examples results in a set of test vectors necessary to detect the behavioral faults. These test vector sets are then applied to synthesized gate level implementations. Multiple synthesis tools and target architectures are employed to create a variety of realizations of the behavioral descrip-

tions. AutoLogic II [68] from Mentor Graphics is used with both the autologic/default and gen_lib/default target technologies. The Leonardo [47] synthesis tool from Exemplar Logic is also used to map the behavioral descriptions to a Xilinx field programmable gate array (FPGA) architecture. The resulting gate level fault coverage provides experimental validation of the effectiveness of the behavioral fault models.

1.6 Contributions and Future Work

The main contributions of this dissertation include improved behavioral fault models as well as the techniques for generalizing the effects of low level faults and abstracting them into the behavioral domain. The new fault models are more closely linked to underlying hardware faults than those developed by previous research. Test vectors based on these new behavioral fault models achieve complete *SSL* gate level fault coverage over a broad range of implementations.

The models and techniques presented in this dissertation represent another important step in the development of a design methodology for performing fault simulation throughout the design process. Chapter 9 presents a brief description of directions for future research. These include expansion of the behavioral fault models, tool development for behavioral test generation and behavioral fault simulation, and migration of fault models to even higher levels of abstraction.

Chapter 2 Previous Research

This chapter surveys previous research in three major areas involved in testing circuits based on behavioral descriptions. First, a survey of the current state of the art in fault modeling at higher levels of abstraction provides the basis from which tests for digital circuits can be developed. Recent research efforts have attempted to develop higher level fault models, which accurately represent faults which occur at lower levels. Second, several methods of injecting faults into higher level models are explored. Injecting faults into a behavioral model can be accomplished by manipulating data or signal values within the model or by actually changing or perturbing the model itself. Finally, various techniques for generating behavioral or functional tests provide a means for evaluating the effective-ness of high level modeling efforts. Their results can be compared to more conventional gate level methods to provide a quantitative measure of fault coverage.

2.1 Fault Models

Fault models provide the underlying basis for the development of tests for digital circuits. The models of interest in this dissertation move up the design hierarchy beyond the gate level to the register or chip level. Functional fault models are based on the input/output relationship of higher level primitives which may incorporate a large number of gates. Behavioral fault models are based on procedural descriptions of the circuits desired functioning. Many models reference the register transfer level which describes procedural data flow among functional primitives.

The majority of functional and behavioral modeling efforts can be traced to four prominent researchers: Jacob Abraham, John Hayes, Sumit Ghosh, and James Armstrong. As a foundation for the development and application of more accurate behavioral fault models, this dissertation examines the models developed by groups including each of these researchers. Modeling at many levels of abstraction is discussed by Abraham et al. [1][2] [17][65][66], but the main focus of this dissertation concerns the functional fault models. Hayes et al. [29][31][32][33][34][63] has worked extensively on fault models for digital circuits, including descriptions of functional fault models leading to a new generic class called *induced* faults. Ghosh and Chakraborty [18][26][27] have proposed a set of fault models which are based on the failure modes of the language constructs of a generic hardware description language. Armstrong and his collaborators [6][7][8][9][22][28][69] have developed an ever evolving series of functional and behavioral fault models and, as will be discussed later, have implemented test generation algorithms using these models.

2.1.1 Functional Faults

In "Fault and Error Models for VLSI," [2] Abraham and Fuchs provide an extensive review of research efforts aimed at deriving realistic models at higher levels which can accurately represent the faults and errors at lower levels. Of primary interest here are their descriptions of several functional fault models: general fault models for functional blocks, models for small functional modules, and fault models for microprocessors.

Given a combinational function with N inputs, a general fault model assumes that this function can be transformed into any other combinational function of N inputs and, therefore, testing it requires application of all 2^N input combinations. Such exhaustive testing is impractical if N is large, however, testing may be manageable if the function is implemented as an interconnection of subfunctions. The exhaustive general fault model could then be used effectively to test these subfunctions.

Models for several small functional modules provide the building blocks for handling larger functions. A key functional module found in many digital circuits is the decoder. It can be described functionally as having N inputs and 2^N outputs and, under normal operations, exactly one output line is activated corresponding to the input address. A detailed study of all single transistor-level faults by Banerjee [12] resulted in the following functional fault model for a decoder:

- 1) Instead of the correct line, an incorrect line is activated.
- 2) In addition to the correct line, an incorrect line is activated.
- 3) No line is activated.

Though such a description is very simple, it was shown to incorporate all of the physical shorts and opens possible in the transistor-level description.

Further study [1] involved another important building block, the multiplexer. This functional module has N inputs, log_2N control signals, and one output. The output is

11

selected to be one of the inputs as determined by the address on the control lines. Under a fault, it can be shown that the behavior of the multiplexer module can be described in the following functional manner:

- 1) A 0 and a 1 cannot be selected on every input line.
- 2) When selecting some input, another input will be selected instead of, or in addition to, the correct input.

Similar fault models exist for other building blocks of more complex functional units.

Even though microprocessors are quite complex, fairly effective functional fault models have been derived at the register transfer level. Thatte [66] visualizes a microprocessor as a set of functions including register decoding, data transfer, data manipulation, and instruction sequencing. A functional fault model is developed for each of these functions. Improvements to this model made by Brahme [17] are based on the conceptual treatment of instructions as consisting of micro-instructions, which, in turn, are composed of a set of micro-orders. The combined fault model for the microprocessor contains the following:

- 1) Fault Model for the Register Decoding Function:
 - Fault-free

 $f_D(R_i) = R_i$ Register *i* selected.

- Faulty

 $f_D(R_i) = R_j$ Register j selected instead of Register i.

 $f_D(R_i) = \phi$ No register selected.

 $f_D(R_i) = \{R_i, R_j\}$ Register *j* selected in addition to Register *i*.

- 2) Fault Model for the Data Transfer Function:
 - any number of lines can be stuck at 0 or 1.
 - any pair of lines *i*, *j* can be coupled.
- 3) Fault Model for the Data Manipulation Function:
 - No specific fault model is presented. (It is assumed that the complete test set for any given ALU can be easily determined.)
- 4) Fault Model for the Instruction Sequencing Function:
 - Under a fault we can have one or more of the following events:
 - One or more microorders can be inactive.
 - Microorders which are normally inactive become active.
 - A set of microinstructions is active in addition to, or instead of, the normal microinstructions.

This approach allows derivation of tests for a microprocessor even if the details of implementation are not known.

With this set of fault models, Abraham and his collaborators have attempted to describe accurately the effects of faults within higher functional modules and thus make complex systems tractable by reducing the number of primitive elements. Though largely based on actual circuit descriptions, care has been taken to make these models as implementation independent as possible by concentrating on the functionality provided by each module.

2.1.2 Physically-Induced Faults

In "Fault Modeling" [34], Hayes also discusses the general functional (GF) fault model as one that allows arbitrary changes to a circuit's truth table (combinational case) or state table (sequential case). The maximum number of states, which can be taken to be one in the combinational case, is assumed to remain constant when faults are present. Detection of GF faults requires essentially exhaustive testing procedures and is thus feasible for a moderate number of input lines. Tests for GF faults in a sequential circuit are termed checking sequences and tend to be long and difficult to compute. Such tests have, however been applied successfully to the representation of certain types of pattern-sensitive faults in RAMs [33] and to testing simple bit-sliced microprocessors [63].

In more recently published research, Hansen and Hayes [29] present a new high-level fault model called the *physically-induced* fault model. If gate-level *single-stuck-line* (SSL) faults are considered with this model, the authors claim that complete functional fault detection can guarantee complete SSL fault detection. The induction concept implies changing the abstraction level at which faults and their effects are considered from a lower to a higher abstraction level. The physical faults of interest include SSL faults, *bridging* faults, and *switch-level* faults. The target abstraction level is the functional level, which is loosely defined to correspond to the register-transfer level. Just as the SSL fault model is a natural functional-level fault model.

The 3-input majority circuit M, shown in Figure 2-1, is presented as a preliminary example where G, P, and C represent carry-generate, carry-propagate, and carry-in respec-

13
tively. There are 11 lines producing 22 *SSL* faults. Analyzing the faulty responses produces 14 different *SIF* functions which can be reduced by *functional equivalence* and *fault dominance* to a minimal set of 6. Table 2-1 lists 6 SIF tests that detect all *SSL* faults.



Figure 2-1 Three-input majority circuit M.

SIF	SIF test	Z	САВ
A cannot propagate	Propagate with A	1	1 1 0
B cannot propagate	Propagate with B	1	1 0 1
M always propagates	Stop propagate	0	100
M cannot generate	Generate	1	0 1 1
B always generates	Stop generate with B	0	0 1 0
A always generates	Stop generate with A	0	0 0 1

Table 2-1
 Minimal SIF test set.

Physically-induced fault techniques have also been applied to several medium-scale examples from the 74X-series. The tests derived for these circuits cover all *SSL* faults in the standard 74X-series gate-level designs [67], a property that has been verified by complete gate-level fault simulation. Furthermore, the test sets are provably minimal and are generally smaller than those found by an efficient gate-level test generator.

By examining different realizations of the same majority function, other independent functional faults may be added. Additional tests could, therefore, be required to detect the *SIF* "generate invalidates propagate." However, given a full set of physical faults, a set of functional faults can be derived, usually without too much difficulty. For example, the work of Shen et al. [62] on inductive fault analysis can be used to supply a comprehensive physical-fault list.

2.1.3 Behavior Faults

In "Behavior-Level Fault Simulation," [26] Ghosh uses behavioral fault models to represent complex failures in VLSI designs. Faulty values of variables that represent state/ timing parameters or a faulty description that is substituted for part of the good description are deliberately introduced into a design that contains no faults. A severe limitation of this method is determining how to select fault models that represent actual failures from the large number that do not. The recommended approach urges designers to use a library of fault models of complex devices that are based on actual failures.

In subsequent research, Ghosh and Chakraborty [18][27] propose fault models based on the failure modes of the language constructs of a generic hardware description language. The programming language C is used to describe hardware with assurances that its language constructs may be extended to other hardware description languages. The fault models presented are relatively complex and attempt to rationalize a link to actual possible hardware faults. Some of the key components of these behavioral fault models include:

- 1) The states of a sequential component may be expressed through variables of integer, Boolean or real types and may fail in one of two modes the state is permanently held at either V_1 or V_2 where V_1 and V_2 specify the lower and upper extremes of the logical value system.
- 2) A "function call" may exhibit two failure modes where it permanently returns V_1 or V_2 , the lower and upper extremes of the range of the function.
- (3) In the "for (CC) $\{E_I\}$ " clause, the body $\{E_I\}$ may either never be executed or always executed regardless of the condition CC.
- 4) In a "switch (Id)" clause, the switch may select either the cases corresponding to the lower or upper extremes of the switch identifier's value system, all or none of the specified cases.
- 5) The "if (X) then $\{E_1\}$ else $\{E_2\}$ " construct may fail such that E_1 is always executed and E_2 is never executed, E_1 is never executed and E_2 is always executed, or E_1 and E_2 are executed when X evaluates to false and true respectively.
- 6) The assignment statement "X = Y" may fail such that X remains unchanged or assumes the lower (V₁) or upper (V₂) extremes of the value system, or X assumes V₁ or V₂ depending on a probability function.

Though relationships to possible hardware faults are proposed, there is no detailed analysis to justify these assertions. A further shortcoming of these models is the restriction to the lower (V_1) or upper (V_2) extremes of the value system. Multiple bit signals must all be stuck at 0 or stuck at 1 rather than allowing for only a single stuck line.

In order to evaluate the performance of the proposed behavior fault models, example designs were fault simulated and compared with gate-level fault simulation in the presence of *stuck-at* faults. The example designs included: 16-to-4 multiplexer; 4-, 8-, and 16-bit ALUs with carry lookahead; shift register; synchronous and asynchronous counters; AMD AM2903 bit slice processor; and the control unit of a complex protocol formatter chip. Between 40 and 60 randomly generated test-vector sets were used for gate-level and behavioral fault simulation and the gate-level and behavioral fault coverages were then compared. The researchers found a strong and consistent correlation of the gate-level and behavioral fault coverage with no occurrence of a behavioral fault coverage exceeding 90% while the gate-level fault coverage was less than 85%.

2.1.4 Model Perturbation

Armstrong et al. has developed a series of behavioral fault models based on the concept of model perturbation [28] of designs using hardware description languages, most recently VHDL. In "Behavioral Fault Simulation in VHDL," [69] Ward and Armstrong define eight behavioral fault classes:

- 1) *Stuck-Then*: represents a failure of the *if-then-else* construct to ever execute the *else* statements.
- 2) *Stuck-Else*: represents a failure of the *if-then-else* construct to ever execute the *then* statements.
- 3) Assignment Control: represents a failure of the VHDL assignment operator to assign a new value to a signal.
- 4) Dead Process: failure of the statements within a process construct to execute.
- 5) *Dead Clause*: failure of the VHDL *CASE* construct to execute one of the alternative sequences of statements (clauses).
- 6) *Micro-operation*: failure of an operator to perform its intended function. The operator may fail to any other operator in its class.

- Local Stuck-data: failure of a signal or variable to have the correct value. The local stuck-data fault is restricted to the expression into which it is mapped.
- 8) *Global Stuck-data*: failure of a signal or variable to change value within the device model.

These fault classes have been continually refined by subsequent research, but still they serve as the basis for many current efforts in behavioral test pattern generation.

As a further refinement to their earlier model, Armstrong, Lam, and Ward, [9] subdivide behavioral faults into two broad categories, *control faults* and *micro-operation faults*. *Control faults* perturb the control points that switch between micro-operation sequences. This fault group includes:

- IF: stuck THEN, stuck ELSE branching will always occur in one direction, independent of control signal values.
- 2) CASE: *dead clause* the selected clause does not execute.
- 3) Assignment fault models the effect of a single assignment not taking place.
- 4) Dead Process fault.

Micro-operation faults perturb individual micro-operations to another and include:

- 1) AND <-> OR.
- 2) INC <-> DEC.
- 3) ADD <-> SUB.

Some significant problems with this modeling technique include choosing to which microoperation to perturb [19] and whether any of these faults can actually occur in hardware.

As part of the development of the "B-algorithm: A Behavioral Test Generation Algorithm," [21][22] Cho and Armstrong developed a new behavioral fault model by applying the concept of *equivalent faults* to the previous model. *Stuck-THEN/stuck-ELSE* faults can be removed from the behavioral fault list if *stuck-at* faults are defined for unnamed signals corresponding to the conditional expressions of the *IF* statement. Likewise, a *micro-operation* fault for a logic operator is detected by a test for a *stuck-at* fault on one of its arguments. Finally, a *dead-clause* fault is equivalent to an *assignment control* fault under the assumption of a single behavioral fault model.

The new behavioral fault model renames *stuck-at* faults to *behavioral stuck-at* faults. Assignment control faults are renamed *behavioral stuck-open* faults and *micro-operation*

faults for arithmetic or relational operators are renamed *micro-operation* faults. The reduced model now includes three types of faults:

- Behavioral Stuck-at (BSA) Fault a bit of a signal, virtual signal, a fanout stem, or a fanout branch is permanently *stuck-at* logic 1 or 0.
- 2) *Behavioral Stuck-open (BSO)* Fault the value of the source expression (righthand side) of an assignment statement is not correctly transferred to its target.
- 3) Micro-operation (MOP) Fault an arithmetic or a relational operator is faulted to another operator. For example, ADD(A,B) is perturbed to SUB(A,B) and SUB(B,A). A summary of fault-free operators and their corresponding faulty operators is provided in Table 2-2.

Fault-free Operator	Faulty Operator
ADD	SUB, XOR
SUB	ADD, XOR
BVEQ	BVNEQ
BVNEQ	BVEQ
BVLT	BVGE
BVLE	BVGT

Table 2-2 Micro-operation Faults

Again, perturbing of micro-operations raises doubts concerning the relationship to actual hardware faults. Use of this fault model with the B-algorithm is discussed later along with other test generation techniques.

2.1.5 Other Research

Two other recent research efforts deserve mention here for completeness. The first, by Riesgo and Uceda [60], attempts to define an RTL fault model which they claim is totally oriented to model hardware faults. At the other end of the spectrum, Al Hayek and Robach [4][5] consider behavioral faults as software faults and apply an adaptation of mutation-based testing, originally proposed to test software programs.

In "A Fault Model for VHDL Descriptions at the Register Transfer Level," [60] Riesgo and Uceda present a fault model based on the VHDL level-0 synthesis subset [25]. Example behavioral descriptions can then be directly synthesized and results compared to the corresponding gate-level designs. The fault model is divided into three classes, depending on the type of object affected by the fault:

- Faults on data: the fault model is based on "stuck-at" faults. The affected expression will take a constant value and the insertion will be made in a statement where the object is referenced. Examples include:
 - bit: stuck-at-'0' and stuck-at-'1'.
 - enumerated: stuck-at-"all possible values".
 - integer: each bit of the resulting bus can be *stuck-at'0'* or *stuck-at-'1'*.

The assumed codification is binary for positive numbers and 2's complement for negative numbers.

2) Faults on expressions: the fault model is based on "*stuck-at*" faults. The affected expression will take a constant value. Examples include:

- if_then_else: the condition of the if statement may be *stuck-at-true* or *stuck-at-false*.

- case_is_when: the expression which controls the case statement may be *stuck-at-"all possible values"*

- for_in_loop: the index controlling the loop may change its range from the *minimum* to the *maximum*+1 and from the *minimum*-1 to the *maximum*.

3) Faults on statements: the fault model is based on "*dead*" faults. The effect of the fault is that the affected statements are not executed. Examples include:

- if_then_else: dead-then, dead-else.

- case_is_when: *dead-alternative*.
- for_in_loop: *dead-loop*.
- procedure call: *dead-call*.
- signal or variable assignment: dead-assignment.

Fault insertion is made on the VHDL code by adding code perturbations to the input description. Code perturbations consist of adding, switching, or eliminating code sentences, to model the circuit behavior under a fault condition.

Experiments were conducted on several VHDL descriptions which were then synthesized in order to obtain a comparison with the *stuck-at* fault model at the logic gate level. The example circuits included 8- and 16-bit ALUs, sequential multiplier, shift register, 16bit counter, and a reception-transmission unit. Test vectors were randomly generated, then VHDL/RT and logic fault coverages were compared. The fault model presented did not give a precise value of the fault coverage achieved at the lower levels, rather an estimation. For those circuits with a large combinational part, the estimation of fault coverage was "optimistic" and the synthesis options had an influence on the accuracy of the estimation. Highly sequential circuits produced "pessimistic" results due to a large number of faults at the RT level that were very difficult to detect or even undetectable. The achieved results are claimed better than Armstrong and Ghosh due to their correlation coefficients being larger than 90%.

In contrast to other hardware oriented approaches, Al Hayek and Robach [4] propose a mutation-based testing strategy in which VHDL behavioral faults are considered as software faults. The generated test set is used to validate the VHDL description, seen as a software program, against (software) design faults as well as its hardware implementation against hardware faults. A VHDL subset is also used, which allows high level functional description of any combinational or sequential circuit that can be easily synthesized by current tools.

The mutation-based fault model [5] defines a set of mutation operators for use with VHDL behavioral descriptions. Mutation operators include:

1) AOR: Arithmetic Operator Replacement.

- replace "+" with "-".

2) ABS: ABSolute value insertion.

3) CR: Constant Replacement

- integer: increment and decrement by 1.

- bit-vector: 1's complement.

- Boolean: complement.

4) CVR: Constant for Variable Replacement.

- every compatible constant.

5) LOR: Logical Operator Replacement.

- AND, OR, NAND, NOR, XOR

- replaced by each of the other operators

6) ROR: Relational Operator Replacement.

- <, >, <=, >=, =, /=

- replaced by each of the other operators

- 7) NOR: No Operation Replacement.
- 8) VCR: Variable for Constant Replacement.
 - every compatible variable.
- 9) UOI: Unary Operator Insertion.
 - each arithmetic expression negated, incremented, and decremented by 1.
 - each logical expression complemented.

An automated test environment was built by translating VHDL to FORTRAN and using an existing software testing system.

On a behavioral VHDL description, the test environment delivers a test set and the associated mutation score value that represents the percentage of non-equivalent detected mutants by the test set. In the context of design and test of hardware systems, the mutation score is viewed as a behavioral fault coverage that measures how well the design has been tested. On a set of high-level synthesis benchmarks (Decoder, ALU, Differential equation, Elliptical wave filter,...), experimental results show that on combinational circuits the obtained gate-level fault coverage is about 94% in the average against 99% for the traditional gate-level ATPGs. However, on sequential circuits the mutation-based test is claimed better as it yields 94% in the average of gate-level fault coverage against 85% for the traditional ATPGs, when they are used without any user assistance.

In order to improve the performance of the mutation-based technique, the authors chose to enhance the test set for certain *complex* operators such as AOR. Mutation analysis does not take into account the size of the hardware implementation, because it considers addition/subtraction as a software operation and consequently generates only one test vector. A heuristic is proposed to supplement the original test set with extra vectors to sufficiently test the complex operators. This heuristic consists of generating N new test vectors for each complex operator, where N is the maximum dimension of the input parameters. The necessity to supplement the original test set simply attempts to cover up an underlying deficiency in the mutation-based fault model.

2.1.6 Summary

All of the modeling efforts presented here attempt to develop fault models at higher levels of abstraction, which accurately represent faults which occur at lower levels. This discussion has been meant to provide a survey of the current state of the art in behavioral fault modeling. Later in this chapter, some of these fault models will serve as the basis for several behavioral test generation algorithms. But first, some techniques for injecting faults into behavioral models will be briefly discussed.

2.2 Fault Injection Techniques

Once a fault model has been defined, some method of injecting these faults into a model of the digital circuit must be developed. Injecting faults into a behavioral model can be accomplished by manipulating data or signal values within the model or by actually changing or perturbing the model itself. An example of the signal manipulation technique is provided by the work of DeLong, Johnson, and Profeta in "A Fault Injection Technique for VHDL Behavioral-Level Models." [23] Modification of behavioral models is demonstrated by Jenn et al. in "Fault Injection into VHDL Models: The MEFISTO Tool." [39] Finally, Yount and Siewiorek present an approach called hybrid fault emulation in "A Methodology for the Rapid Injection of Transient Hardware Errors." [71]

2.2.1 Instruction Set Architecture

In order to perform fault injection experiments early in the design cycle, DeLong, Johnson, and Profeta [23] developed a technique to inject faults into a VHDL behavioral model of a system. This technique is demonstrated on an instruction set architecture (ISA) model of an embedded control system. Single or multiple bit faults are injected into internal processor registers, any location in memory, and any range of locations in the memory map. Signal values are corrupted by using a user-defined VHDL data type to communicate with a bus resolution function. When two different sources are trying to update a signal at the same time, the bus resolution function resolves the conflict and assigns the desired value to the signal. This technique allows the designer to inject faults on desired signals in a behavioral description with minimal changes to the existing code. Because the functionality of the design is not changed, the same model can be used to simulate both fault-free and faulty behavior.

2.2.2 MEFISTO

In contrast to the previous technique, the MEFISTO Tool [39] uses two other techniques to inject faults into VHDL models. The first category requires modification of the VHDL model and the second one uses the built-in commands of the simulator. Modification of the VHDL model is accomplished through the addition of components called *saboteurs* and *mutants*. A *saboteur* is a VHDL component that alters the value or timing characteristics of one or several signals when activated. A *mutant* is a component description that replaces another component description. When activated, it imitates the component's behavior in the presence of faults. Both signal and variable manipulations can be used for controlling, i.e., activating and deactivating, *saboteurs* and *mutants*. In this way, the injection of faults can be controlled by built-in commands of the simulator.

The main reason for using the built-in commands of the simulator for fault injection is that this does not require the modification of the VHDL code. However, the applicability of these techniques depends strongly on the command languages of the simulators. The values of either signals or variables may be manipulated by stopping and restarting the simulation. For example, a temporary stuck at fault may be injected by application of the following sequence of pseudo commands:

- 1) SimulateUntil <fault injection time>
- 2) FreezeSignal <signal name> <signal value>
- 3) SimulateFor <fault duration>
- 4) UnFreezeSignal <signal name>
- 5) SimulateFor <observation time>

For a permanent fault, steps 3 and 4 are skipped. Intermittent faults can be injected using a more complex command sequence.

2.2.3 Hybrid Fault Emulation

Yount and Siewiorek [70][71] developed a fault injection methodology for processors based on a register transfer level fault model. The approach, called hybrid fault emulation, uses the actual circuit to perform fault injection. A transient fault occurs during the interval $T = [t_j, t_k]$. The system is allowed to run until some time, t_j , and the state of the machine is captured. A low level model is then used over a limited interval with the captured state. Reverse fault emulation is used to generate output and set the new state at t_k to match the effects of the desired fault. Since fault simulations are only run over short periods of time, many more fault injection experiments can be run using the same simulator resources. This method is, however, limited to only transient faults and applies to the evaluation of an actual system rather than one still in the design process.

2.2.4 Summary

Though brief, this survey of fault injection techniques provides sufficient insight into methods for manipulating behavioral models. Fault injection can be as simple as starting and stopping the simulation to change desired values or as complicated as developing *mutants* that imitate a component's behavior in the presence of faults. The bus resolution function technique is simply an adaptation of the concept of a *saboteur* that alters the value of a signal in the model. Test designers now have a range of fault models and injection techniques from which to develop behavioral test generation algorithms.

2.3 Test Generation Techniques

Test generation techniques at higher levels of abstraction can be based on either functional or behavioral descriptions and their corresponding fault models. This dissertation investigates research efforts which have produced automated test pattern generation (ATPG) methods and tools to support circuit designers at or near the register transfer level. Results of these test generators can be used to evaluate the effectiveness of the underlying behavioral fault models. Lin and Su [49][50] developed a functional test generation algorithm which uses a register transfer level fault model based on the functional fault work of Abraham [2][17][66]. Armstrong and his collaborators [13][14][15][21][22][46][55][56] [57][58] have produced a series of test generators based on their ever evolving behavioral fault models. Several other researchers [20][54] have also developed ATPG algorithms which utilize the fault models of Ward and Armstrong [69]. Finally, Santucci and Giambiasi et al. [59][61] have produced a prototype model of a test pattern generator which uses a fault model claimed equivalent to that of Ghosh and Chakraborty [18][27]. However, since their research focus is on optimization of the test generation algorithm, no results are published that can help evaluate the effectiveness of their fault model.

Most fault-oriented techniques use some form of a three step approach to the test generation process. First, a fault must be *activated* at the desired location in the circuit model. Then, the effect of the fault must be *propagated* to a point where it can be observed and, hence, detected. Finally, the inputs of the model must be determined to *justify* the desired signal values throughout the circuit. Variations of these techniques attempt to utilize the information available in these higher level models to more efficiently accomplish the computationally intensive tasks of fault *propagation* and *justification*.

2.3.1 S-Algorithm

The S-Algorithm [49][50] is based on a register transfer (RT) level fault model similar to the functional fault models discussed by Abraham and Fuchs [2]. The reduced fault set is divided into three groups for ease of fault identification:

- Register decoding faults and operator decoding faults regular registers and ALU operators are "global" among RT-statements.
- 2) *Condition* faults, *jump* faults, and *data transfer* faults with constant source registers - they are "local" within an RT-statement.
- 3) Data transfer faults with regular source registers.

The overall test generation algorithm also includes a preprocess stage which includes partitioning the system under test and a postprocess stage that evaluates fault coverage.

Major parts of the heuristic test generation algorithm were implemented in IBM Pascal and experimental results have been published for several example circuits. For a hardware multiplier described by 16 RT-statements, the program generated 66 test patterns and claims a 96.4% fault coverage. For the SIMPLE-CALCULATOR, 134 test patterns were generated resulting in a 97.2% fault coverage. No comparisons are provided to evaluate the resulting fault coverages versus a traditional gate level stuck at model.

2.3.2 B-Algorithm

Early test generation algorithms based on VHDL behavioral descriptions included "A Heuristic Chip-Level Test Generation Algorithm" [13] and "The E-Algorithm, an Automatic Test Generation Algorithm for Hardware Description Languages." [55] Both were strongly influenced by the pioneering work of Levendel and Menon in "Test Generation Algorithms for Computer Hardware Description Languages." [48] The fault models used were evolving versions of the model published by Ward and Armstrong [69]. Reported results on medium complexity circuits were in the range of 90% gate level fault coverage.

Further along in the evolution, the Behavioral Test Generator (BTG) [57][58] uses a VHDL subset and the same behavioral fault model [69]. The faulted operation is first *activated* (fault sensitization), then the effect of the fault is *propagated* to the output (fault *propagation*). Examples are provided for *propagation* through behavioral control constructs and *propagation* through data paths. Behavioral fault coverage is then evaluated as the ratio of detected behavioral faults to the total number of behavioral faults.

BTG was tested using a set of 11 logic circuits representing a cross section of generic types of logic. The average equivalent gate level coverage for the experiments on these circuits was 92%. A major drawback was what they called the "big micro-operation problem." Faults can be applied to micro-operations that represent large blocks for logic. For example ACUM \leq ADD(A,B,CIN) implies a multi-bit adder. It is impossible for a single vector to detect the majority of gate level faults in such a complex structure. O'Neill et al., thus, resorted to heuristics to supplement their test vector set to bring up their equivalent gate level coverage. The necessity to add test vectors to those generated by their algorithm points back to a fundamental deficiency in their underlying fault models.

Continuing efforts by the same research group produced the B-algorithm [21][22]. This test generation algorithm uses the reduced behavioral fault model discussed earlier in this chapter: *behavioral stuck-at* faults, *behavioral stuck-open* faults, and *micro-operation* faults. Again, a three step approach is applied to the test generation process: *activation*, *propagation*, and *justification*. As with BTG, the B-algorithm cannot generate sufficient test vectors for micro-operation faults. In order to raise their equivalent gate level coverage numbers to acceptable levels, an additional 4n-1 test vectors are generated by a heuristic test generator for each *n*-bit micro-operation. Even with the stated improvements and the modified fault model, the B-algorithm still suffers from the same underlying deficiencies concerning the modeling of faults in complex operations.

2.3.3 Other Research

In "Analysis of the Gap Between Behavioral and Gate Level Fault Simulation," [20] Chen and Perumal describe the details of an ATPG system for VHDL behavioral models. The behavioral fault models based on Ward and Armstrong [69] are expanded and classified into six different categories:

- 1) Input stuck-at fault
- 2) *If stuck then* fault
- 3) *If stuck else* fault
- 4) Assignment statement fault
- 5) *Dead clause* fault
- 6) Local stuck data fault.

The problems associated with micro-operation faults have been avoided by eliminating such constructs from their behavioral models.

For the simple example discussed in their paper, 34 of the 37 non redundant gate level faults are detected by behavioral fault simulation, resulting in a fault efficiency of 92%. The results of a more complicated example of a vending control unit are also presented. A total of 250 test patterns are required to test for 217 faults in the synthesized gate level circuit. Detection of 189 faults results in a fault coverage of 87%. If scan design techniques are applied to the circuit, then a fault coverage of 98% is claimed to be achieved.

The last example to be discussed is the "Behavioral Fault Simulation and ATPG System for VHDL." [54] For purposes of comparison, VHDL behavioral code is synthesized to gate-level implementations and analyzed. The underlying fault models are again based those proposed by Ward and Armstrong [69], this time including micro-operation faults. A linear feedback shift register (LFSR) algorithm was utilized to generate the test patterns in this research.

Nine circuits were used to evaluate the performance of the system. Actual behavioral fault coverages ranged from 18% to 100%, with sequential descriptions, such as counters and controllers, performing worst and combinational circuits performing best. These results were compared to randomly generated test patterns applied to synthesized gate level circuits. Many of the resulting gate level descriptions proved to be difficult to test with the random test patterns and, therefore, also had low fault coverage numbers. Hence, the authors were able to claim success based on a different figure of merit, relative detection of testable gate-level faults. Using this questionable comparison, Noh et. al. claims an overall result of detecting around 98% of all testable gate level faults. There is dubious

merit in claiming that detecting only 18% of the behavioral faults in a vending machine controller versus 19% in the synthesized gate level circuit implies a 95% success rate.

2.4 Conclusions

This survey of the state of the art in high level fault modeling and test generation clearly indicates that there is no widely accepted solution to the problem. Modeling techniques range from the functional analysis employed by Abraham and Hayes, to the procedural data flow of behavioral descriptions used by Ghosh and Armstrong. Test generation algorithms contained some functional techniques, but mainly relied on the behavioral fault models developed by Armstrong. Examples provided a figure of merit by which to judge their effectiveness, but few common measures could be found. When results did not meet expectations, alternate methods, such as heuristics or testability, were invoked to improve the statistics.

In the functional arena, Abraham has attempted to describe accurately the effects of faults within higher functional modules and thus make complex systems tractable. Though largely based on actual circuit descriptions, care has been taken to make these models as implementation independent as possible. With his *physically-induced* fault techniques, Hayes has been able to derive minimal test sets for several medium-scale examples. Key to this research are the concepts of *functional equivalence*, *fault dominance*, and *compatible fault sets*. Both of these functional methods show promise for application to behavioral faults and hardware description languages.

For behavioral models, though Ghosh proposes relationships to possible hardware faults, there is no detailed analysis to justify these assertions. A further shortcoming of these models is the restriction that multiple bit signals must all be *stuck at 0* or *stuck at 1* rather than allowing for only a *single stuck line*. Similarly, the evolving set of behavioral fault models by Armstrong and their subsequent test generation algorithms seem to move too far away from the hardware which they attempt to describe. In order to compensate for what they call the "big micro-operation problem," the researchers resorted to heuristics to supplement their test vector set to increase their equivalent gate level coverage. The necessity to add test vectors to those generated by their algorithm points back to a fundamental deficiency in the underlying fault models.

The behavioral fault models developed in this dissertation only affect the *activation* step of the test generation process. Hence, the high level algorithms developed to handle the *propagation* and *justification* steps still remain valid. Integration of new fault models with an existing behavioral test generation algorithm can be of mutual benefit. Such advanced test generation algorithms already address problems such as reconvergent fanout [3], while use of more complex fault models can eliminate the need to supplement test vector sets via heuristics.

Though previous research provides no clear cut solution to modeling faults at higher levels of abstraction, valuable insights have been gained by the examination of each of these techniques. Certain key concepts from past efforts have immediate applicability to further research. Other works can also serve as benchmarks for comparison of future results. Behavioral fault modeling remains an active research area which requires continued exploration.

Chapter 3 A New Control Fault Model

Previous research has proposed fault models for the control constructs *if-then-else* and *case*, such as *stuck-then/stuck-else* and *dead clause*, based solely on perturbing the language without a well defined link to the underlying hardware. This dissertation proposes a new behavioral fault model based on a functional analysis of gate level implementations. By combining VHDL synthesis information with industry standard *single-stuck-line* (*SSL*) faults, a new control fault model can be obtained which is more closely linked to the underlying hardware.

3.1 IF-THEN-ELSE

The first control construct to be modeled will be *if-then-else*, common to most hardware description languages. In Appendix E Armstrong and Gray identify the link between the *if* statement and the functional building block of a multiplexer. Likewise, Mentor Graphics demonstrates how the selection activity of an *if-then-else* construct is implemented in hardware as a multiplexer. These examples reinforce the underlying intuition that a language construct that involves selection leads naturally to a hardware construct that implements selection, the multiplexer.

3.1.1 Synthesis of a Simple Example

As a first example, an *if* statement is used to select one of two input signals to be assigned to an output signal. The VHDL behavioral description is shown in Figure 3-1.

```
if SEL = `0' then
    Z <= Y0;
else
    Z <= Y1;
end if;</pre>
```

Figure 3-1 Behavioral description for example IF1.

The VHDL code was compiled using the Mentor Graphics' Design Architect and then synthesized using AutoLogic II. The resulting circuit shows the expected multiplexer architecture in Figure 3-2.





Figure 3-2 Synthesized hardware for example IF1.

The synthesized hardware implementing example IF1 has the two inputs, Y0 and Y1, connected to inputs A and B respectively of a 2-to-1 multiplexer, MUX21. In subsequent discussions, these will be referred to as the channels of the multiplexer, Channel A (CHA) and Channel B (CHB).

3.1.2 Functional Analysis

In order to perform a functional analysis similar to the method used by Hansen and Hayes [29] in their work on physically-induced faults, a gate level design of the multiplexer architecture is needed. To make the resulting models independent of any specific implementation, several different gate level realizations will be examined and compared.

The first gate level multiplexer was obtained by expanding the functional element MUX21 one level lower in the design hierarchy. The resulting gate level circuit is recognized as a sum-of-products (SOP) implementation shown in Figure 3-3.



Figure 3-3 SOP gate level circuit for MUX21.

To analyze the gate level circuit, the effect of single-stuck-line (SSL) gate level faults will be examined. This analysis will determine a set of functional faults which are induced by the lower level SSL faults. Testing for these functional faults will then ensure complete testing for the original gate level faults.

The gate level circuit contains 10 logical lines. Applying the SSL fault model where each line can be either stuck-at-0 or stuck-at-1, there are a total of 20 gate level SSL faults

in the circuit. The line labeled A being *stuck-at-0* and *stuck-at-1* will be indicated as A-0 and A-1 respectively. By activating each SSL fault individually in the gate level circuit and evaluating the resulting output response to changing inputs, a fault table is obtained. In Table 3-1 the fault free behavior of the circuit is shown in column Z. For simplicity, only outputs due to a SSL fault that differ from the fault free behavior of the circuit are shown.

SEL	Y1	Y0	Ζ	A-0	A-1	B-0	B-1	C-0	C-1	D-0	D-1	Е-0	E-1	F-0	F-1	G-0	G-1	0-H	H-1	S-0	S-1	Z-0	Z-1
0	0	0	0		1						1								1				1
0	0	1	1	0				0		0		0									0	0	
0	1	0	0		1						1	1		1			1		1		1		1
0	1	1	1	0				0		0												0	
1	0	0	0				1				1								1				1
1	0	1	0				1		1		1		1						1	1			1
1	1	0	1			0						:	0		0	0		0		0		0	
1	1	1	1			0									0	0		0				0	

Table 3-1 SSL fault table for SOP MUX.

Faults which cause the same faulty output can be considered *functionally equivalent* [45]. Such faults can be combined in the fault table, since there is no way to distinguish between these faults by observing the circuit's output behavior. To further reduce the functional faults, consider the concept of *dominance* of one fault over another. The fault (column) F1 is said to *dominate* the fault (column) F2 if F1 has a faulty output in at least every row in which F2 has a faulty output [45]. The dominating fault (column) F1 may be removed from the fault table, since any test which detects fault F2 will also detect fault F1. Fault reductions due to *functional equivalence* and *dominance* are shown in Table 3-2.

Faults	Remarks	Faults	Remarks
A-1, E-0, H-1, S-1, Z-1	Dominate G-1	C-0, D-0	Equivalent to A-0
B-1, D-1, E-1, S-0	Dominate C-1	F-0	Equivalent to G-1
Z-0	Dominate B-0	F-1, G-0, H-0	Equivalent to B-0

Table 3-2 Fault reductions.

Only four functional faults remain after applying the indicated fault reductions. Testing for this reduced set of faults will ensure complete coverage of all 20 original *SSL* gate level faults. The reduced fault table and appropriate test vectors are shown in Table 3-3. The two test vectors labeled mandatory are the only ones that cover a specific fault. For the other faults two options are available; selecting one test vector from each group will provide coverage for the faults in question.

Test Vector	SEL	Y1	Y0	Z	A-0	G-1	B-0	C-1
	0	0	0	0				
Option 1	0	0	1	1	0			
Mandatory	0	1	0	0		1		
Option 1	0	1	1	1	0			
	1	0	0	0				
Mandatory	1	0	1	0				1
Option 2	1	1	0	1			0	
Option 2	1	1	1	1			0	

 Table 3-3 Reduced fault table for SOP MUX.

3.1.3 Alternate Implementation

In order to investigate an alternate gate level implementation of the multiplexer architecture, consider the Karnaugh map for the output function Z, shown in Figure 3-4. The groupings of minterms (1's), indicated by the dashed lines, produce an *SOP* representation consistent with the gate level circuit previously analyzed from Figure 3-3.



Figure 3-4 Karnaugh map for MUX.

To obtain a *product-of-sums* (*POS*) implementation of the multiplexer, the maxterms (0's) are grouped, as indicated with the solid lines. The resulting function for the output is: $Z = (SEL + Y0) \cdot (\overline{SEL} + Y1)$. A gate level realization is shown in Figure 3-5.



Figure 3-5 POS gate level MUX.

A functional analysis can now be performed on the SSL gate level faults for this circuit. After appropriate reductions for *functional equivalence* and *fault dominance*, the resulting fault table is shown in Table 3-4.

Test Vector	SEL	Υ1	Y0	Ζ	A-1	F-0	B-1	C-0
Option 1	0	0	0	0	1			
Mandatory	0	0	1	1		0		
Option 1	0	1	0	0	1			
	0	1	1	1				
Option 2	1	0	0	0			1	
Option 2	1	0	1	0			1	
Mandatory	1	1	0	1				0
	1	1	1	1				

 Table 3-4 Reduced fault table for POS MUX.

It should be noted here that a NAND-NAND realization of the *SOP* circuit and a NOR-NOR realization of the *POS* circuit produce the same reduced fault tables shown in Table 3-3 and Table 3-4, respectively. The reduced set of functional faults only affect the controlling inputs to the multiplexer channels, which remain unchanged due to conversions to NAND-only or NOR-only circuits using DeMorgan's theorem [41].

3.1.4 Generalized Functional Fault Model

Comparison of the reduced fault tables for both the *SOP* and *POS* implementations of the multiplexer can now yield a generalized functional fault model, not tied to a specific realization. There are no contradictions among the test vectors indicated in the two fault tables. Selection of the mandatory test vectors from each table provides complete coverage of the functional faults from both tables.

A generalized functional fault model is, therefore, presented in Table 3-5. The functional faults have been renamed *SOP_A*, *SOP_B*, *POS_A*, and *POS_B* to indicate the origin of their mandatory test vector and the channel of the multiplexer which they corrupt. Testing based on the indicated vectors should provide complete coverage of gate level *SSL* faults for multiple multiplexer implementations.

Test Vector	SEL	$\mathbf{Y1}$	Y0	Z	SOP_A	POS_A	SOP B	POS B
	0	0	0	0				
Mandatory	0	0	1	1		0		
Mandatory	0	1	0	0	1			
	0	1	1	1				
	1	0	0	0				
Mandatory	1	0	1	0			1	
Mandatory	1	1	0	1				0
	1'	1	1	1				

Table 3-5 Generalized functional fault model.

3.1.5 Development of a Behavioral Fault Model

Examination of the relationship between the generalized functional fault model and the initial behavioral description will result in a behavioral fault model for the *if-then-else* construct in example IF1. This final step in the abstraction of *SSL* gate level faults into the behavioral domain provides the link between lower and higher level fault models, which has been lacking in previous research.

The *then* clause in example IF1, $Z \le Y0$, corresponds to the upper half of the truth table in Table 3-5. It can be seen from the fault table, that two of the functional faults, SOP_A and POS_A , uniquely affect the *then* clause. The SOP fault table in Table 3-3 shows that SOP_A was derived from the SSL gate level fault G-1. Referring to the gate level SOP circuit in Figure 3-3, it can be seen that the fault G-1 causes undesired activation of *Channel B*, while attempting to select *Channel A*. The functional fault SOP_A , therefore, causes a corruption of *Channel A* by ORing it with *Channel B* in the final stage of the multiplexer. Behaviorally, this fault can be described as a corruption of the *then* clause, resulting in the definition of the behavioral fault *Clause-CORRUPT (OR)*, or specifically *THEN-CORRUPT (OR)*. The faulty version of the *then* clause can be modeled by ORing the right hand side of its assignment statement with the right hand side of the assignment statement from the *else* clause, resulting in Z $\leq Y0$ **OR Y1**.

The other functional fault affecting the *then* clause, POS_A , was derived from the SSL gate level fault *F-0*, shown in Table 3-4. Referring to the gate level *POS* circuit in Figure 3-5, fault *F-0* also causes undesired activation of *Channel B*, while attempting to select *Channel A*. The functional fault *POS_A*, therefore, causes a corruption of *Channel A* by ANDing it with *Channel B* in the final stage of the multiplexer. Behaviorally, this fault also causes a corruption of the *then* clause, resulting in the definition of the behavioral fault *THEN-CORRUPT (AND)*. The faulty version of the *then* clause can be modeled by ANDing the right hand side of its assignment statement with the right hand side of the assignment statement from the *else* clause, resulting in Z <= Y0 **AND Y1**.

In a manner identical to the previous discussion, behavioral faults can be defined for the two functional faults that affect the *else* clause. The functional fault *SOP_B* corresponds to the behavioral fault *ELSE-CORRUPT (OR)*. Finally, the functional fault *POS_B* produces the behavioral fault *ELSE-CORRUPT (AND)*.

The generalized set of four functional faults derived for example IF1, therefore, result in the definition of four behavioral faults. These four faults form an initial behavioral fault model for the control construct *if-then-else*: Each clause of an *if-then-else* statement can be affected by two behavioral faults, *Clause-CORRUPT* (*OR*) and *Clause-CORRUPT* (*AND*). The new control fault model is summarized in Figure 3-6.

THEN-CORRUPT (OR)	ELSE-CORRUPT (OR)
if SEL = $0'$ then	if SEL = $0'$ then
Z <= Y0 OR Y1 ;	Z <= Y0;
else	else
Z <= Y1;	Z <= Y1 OR Y0 ;
end if;	end if;
THEN-CORRUPT (AND)	ELSE-CORRUPT (AND)
if SEL = $0'$ then	if SEL = $0'$ then
Z <= Y0 AND Y1;	Z <= Y0;
else	else
Z <= Y1;	Z <= Y1 AND Y0;
end if;	end if;

Figure 3-6 Control fault model for if-then-else.

The test vectors for the new behavioral faults follow directly from the functional faults from which they were derived. The behavioral fault *THEN-CORRUPT (OR)* corresponds to the functional fault *SOP_A*. From Figure 3-6, the test vector (*SEL Y1 Y0*) 010, causes the *then* clause of the *if* statement to be selected. The fault free response assigns the 0 from input Y0 to the output Z, regardless of the value on input Y1. Under the behavioral fault *THEN-CORRUPT (OR)*, the *then* clause assigns Y0 **OR Y1** to Z, resulting in Z=1, contrary to the fault free output. Thus, the test vector 010 constitutes a valid test for the behavioral fault *THEN-CORRUPT (OR)*.

The test vector for the behavioral fault *THEN-CORRUPT* (AND) comes from the functional fault *POS_A*, 001. The fault free behavior again selects the *then* clause, resulting in Z=1. Under the fault *THEN-CORRUPT* (AND), the *then* clause assigns YO AND Y1 to Z, resulting in Z=0. Similarly, the test vector for *ELSE-CORRUPT* (OR) is 101 and the test vector for *ELSE-CORRUPT* (AND) is 110.

3.1.6 Evaluation of the New Behavioral Fault Model

In order to evaluate the effectiveness of the new behavioral fault model, it will be compared to the underlying gate level faults it was meant to encompass. The four test vectors required to detect the behavioral faults in Figure 3-6 are 001, 010, 101, and 110. Applying these test vectors to the gate level circuits in Figure 3-3 (*SOP*) and Figure 3-5 (*POS*) results in the detection of the *SSL* gate level faults indicated in Table 3-6. Careful examination of the results confirms that the behavioral test vectors achieve complete coverage of *SSL* gate level faults in either the *SOP* or *POS* implementation of the multiplexer.

Behavioral fault	Test Vector	SOP SSL Faults Detected	POS SSL Faults Detected
THEN-CORRUPT (OR)	010	A-1, D-1, E-0, F-0, G-1, H-1, S-1, Z-1	A-1, C-1, D-1, S-1, Z-1
THEN-CORRUPT (AND)	001	A-0, C-0, D-0, E-0, S-1, Z-0	A-0, D-0, E-1, F-0, G-0, S-1, Z-0
ELSE-CORRUPT (OR)	101	B-1, C-1, D-1, E-1, H-1, S-0, Z-1	B-1, E-0, F-1, G-1, S-0, Z-1
ELSE-CORRUPT (AND)	110	B-0, E-1, F-1, G-0, H-0, S-0, Z-0	B-0, C-0, D-0, G-0, S-0, Z-0

Table 3-6 SSL faults detected by behavioral test vectors.

3.1.7 Comparison with Previous Behavioral Fault Models

As was discussed in Chapter 2 of this dissertation, most higher level fault models address the control construct *if-then-else*. The common fault model proposed by previous research is *stuck-then/stuck-else*. This fault model has been used by Armstrong [7][8][9] [13][58][69], Ghosh [18][27], Riesgo [60], and Chen [20][54]. All these researchers also use *stuck-data* faults on their non-control signals, so for completeness, these will be considered in combination with the control faults. Armstrong and Cho [22] also proposed the *Behavioral Stuck-at* (BSA) fault, combining *stuck-data* and *stuck-control* faults by defining *stuck-at* faults for unnamed signals corresponding to the conditional expressions of an *if* statement. All these behavioral fault models use the same fault technique, applying *stuck-at* faults to the data and control inputs of the circuit.

The effectiveness of the *stuck-data* and *stuck-control* fault models will now be evaluated by applying them to the behavioral description in example IF1. Possible test vector sets will be developed and their ability to detect *SSL* gate level faults will be compared. There are three data signals (*Y0*, *Y1*, *Z*) and one control signal (*SEL*) in the example. Applying the *stuck-data* and *stuck-control* fault models implies a *stuck-at-0* and *stuck-at-1* fault for each signal, resulting in the eight behavioral faults shown in Table 3-7. Note that the two control faults *SEL-0* and *SEL-1* are equivalent to *stuck-then* and *stuck-else*, respectively.

Test Vector	SEL	$\mathbf{Y1}$	Y0	Z	V0-0	Y0-1	Y1-0	Y1-1	Z-0	Z-1	SEL-0	SEL-1
Option 2	0	0	0	0		1				1		
Options 1 & 6	0	0	1	1	0				0			0
Options 2 & 6	0	1	0	0		1				1		1
Option 1	0	1	1	1	0				0			
Option 4	1	0	0	0				1		1		
Options 4 & 5	1	0	1	0				1		1	1	
Options 3 & 5	1	1	0	1			0		0		0	
Option 3	1	1	1	1			0		0			

 Table 3-7 Stuck-data and stuck-control behavioral faults.

None of the test vectors in Table 3-7 are mandatory; each behavioral fault has at least two possible tests that detect it. For completeness, all possible combinations of test vectors will be examined. There are $2^6 = 64$ possible test vector sets, but due to overlapping coverage among test vectors, only 49 of them are unique.

First, consider the effectiveness of a test vector set of size four at detection of gate level *SSL* faults. The test vectors for Set 1 are listed in Table 3-8 along with the behavioral faults from Table 3-7 that they detect. As expected, all behavioral faults are detected. Also indicated in Table 3-8 are the *SOP* circuit gate level *SSL* faults, from Figure 3-3 and Table 3-1, which are detected by each of the test vectors. Examination of the gate level fault coverage shows that the *SOP SSL* faults *C-1*, *F-0*, and *G-1* are left undetected by test vector Set 1.

	Test Vector	Behavioral Faults Detected	SOP Gate Level SSL Faults Detected
	000	Y0-1, Z-1	A-1, D-1, H-1, Z-1
	001	Y0-0, Z-0, SEL-1	A-0, C-0, D-0, E-0, S-1, Z-0
Set I	100	Y1-1, Z-1	B-1, D-1, H-1, Z-1
	110	Y1-0, Z-0, SEL-0	B-0, E-1, F-1, G-0, H-0, S-0, Z-0

Table 3-8 Faults detected by test vector Set 1.

Similar analysis shows that any test vector set that does not contain the test vector 101 will fail to detect the gate level fault *C-1*. Likewise, any test vector set that does not contain the test vector 010 will fail to detect the gate level faults *F-0* and *G-1*. Of the nine test vector sets of size four, only four sets contain both test vectors 010 and 101. Coverage does not improve greatly with test vector set size. Half of the 24 sets containing five test vectors do not contain both test vectors necessary to ensure complete gate level fault coverage. Even when the size of the test vector set is increased to six, there are still seven sets out of a possible 16 that fail to detect all gate level faults.

Similar results are obtained when the previous fault models are compared to a *POS* gate level circuit. A large number of possible test vector sets do not contain the test vectors necessary to ensure complete gate level fault coverage. Hence, the new behavioral fault model, based on functional analysis of control constructs, gives improved gate level fault coverage compared to the previous *stuck-then/stuck-else* fault model.

3.1.8 Expansion of the Fault Model

The previous example involved selection of one of two options, hence its implementation with a 2-to-1 multiplexer. Selection from a larger set of input options must now be examined and the effects on the behavioral fault model developed. The *if-then-else* construct contains an optional *elsif* clause to allow selection based on multiple conditions. Multiple *elsif* clauses can be used to allow selection among any number of inputs.

```
if SEL = "00" then
    Z <= Y0;
elsif SEL = "01" then
    Z <= Y1;
elsif SEL = "10" then
    Z <= Y2;
elsif SEL = "11" then
    Z <= Y3;
end if;</pre>
```

Figure 3-7 Behavioral description for example IF2.

Example IF2, in Figure 3-7, selects one of four inputs (Y3, Y2, Y1, Y0) for assignment to a single output (Z), based on the value of two control bits (SEL). For example, the control bits SEL = "10" represent the binary encoding for 2, hence input Y2 is selected for

assignment to Z. Note the lack of a final *else* clause, due to complete elaboration of values for *SEL* in the *if-elsif* clauses. Synthesis of example IF2 by AutoLogic II results in the expected 4-to-1 multiplexer architecture shown in Figure 3-8.



Figure 3-8 Synthesized hardware for example IF2.

3.1.8.1 Functional Analysis

In order to perform a functional analysis similar to example IF1, a gate level implementation of the entity MUX41 is needed. Mentor Graphics' Design Architect was used to provide the gate level detail shown in Figure 3-9. Note that the resulting *SOP* gate level structure agrees with the functional block diagram of the 4-line-to-1-line data selector/ multiplexer found in *The TTL Data Book* [67].



Figure 3-9 Gate level circuit for example IF2.

The gate level circuit contains 18 distinct internal lines (labeled E through V) in addition to the six inputs (A, B, C, D, S0, S1), and one output (Z) for a total of 25 logical lines.

Again, applying the SSL fault model where each line can be either *stuck-at-0* or *stuck-at-1*, there are a total of 50 gate level SSL faults in the circuit. The 2^6 possible input combinations result in a 64 by 50 fault table which will not be reproduced here. Reductions in the fault table due to *functional equivalence* and *fault dominance* are listed in Table 3-9.

Faults	Remarks	Faults	Remarks
A-1, I-0, J-1, P-1	Dominate N-1	K-0, L-0, M-0	Equivalent to A-0
B-1, M-1, Z-1	Dominate L-1	N-0, O-0, P-0	Equivalent to B-0
C-1, V-1	Dominate U-1	Q-0, R-0, S-0	Equivalent to C-0
D-1, S-1	Dominate R-1	T-0, U-0, V-0	Equivalent to D-0
I-1, J-0, Z-0	Dominate B-0		
F-0, G-1	Dominate Q-1		
F-1, G-0	Dominate C-0		

 Table 3-9 Fault reductions for example IF2.

The gate level faults of primary interest are those controlling the switching of the multiplexer channels, the inputs to the channel AND gates. Activating the external control line fault SEL(0)-0 has the effect of simultaneously activating the faults L-1, N-0, R-1, and U-0. Each of these gate level faults corresponds to, or is equivalent to, an undominated functional fault. Testing for each of these functional faults provides complete test coverage for the fault SEL(0)-0. Combinations of undominated faults for each of the external control line faults are shown in Table 3-10.

Faults	Covering Faults			
S0-0, H-1	B-0	D-0	L-1	R-1
S0-1, H-0	A-0	C-0	N-1	U-1
S1-0, E-1	C-0	D-0	K-1	O-1
S1-1, E-0	A-0	B-0	Q-1	T-1

 Table 3-10 Covering faults for external control line faults.

The resulting reduced fault set for example IF2 contains three distinct faults for each of the four channels of the multiplexer, for a total of 12 functional faults. The set of func-

tional faults is summarized in Table 3-11 along with the associated test vectors; don't care values are indicated by an X. For ease of comparison with the remarks column, the vectors are labeled with the multiplexer inputs (S1 S0 D C B A). This notation for test vectors is equivalent to the external input combination (SEL Y3 Y2 Y1 Y0).

Fault	Remarks	Test Vector (S1 S0 D C B A)
A-0	CHA = 0	00 XXX1
B-0	CHB = 0	01 XX1X
C-0	CHC = 0	10 X1XX
D-0	CHD = 0	11 1XXX
N-1	CHA = CHA OR CHB	00 XX10
Q-1	CHA = CHA OR CHC	00 X1X0
L-1	CHB = CHB OR CHA	01 XX01
T-1	CHB = CHB OR CHD	01 1X0X
K-1	CHC = CHC OR CHA	10 X0X1
U-1	CHC = CHC OR CHD	10 10XX
O-1	CHD = CHD OR CHB	11 0X1X
R-1	CHD = CHD OR CHC	11 01XX

 Table 3-11 Reduced functional faults for example IF2.

Examination of the remarks column of Table 3-11, indicates that each channel can be corrupted by two different sources. Each of a channel's logically adjacent neighbors can cause a corruption, where logical adjacency means that the combination of control inputs (*S1 S0*) differ by only one bit. For example, *Channel A* is selected by control inputs 00 and is, therefore, logically adjacent to *Channel B* (01) and *Channel C* (10).

Further examination of the test vectors associated with any pair of corruptions shows that, due to the don't cares, these vectors are not inconsistent with each other. It is possible to chose a single test vector which will detect both corruptions of a channel by its adjacent neighbors. Using the terminology of Hansen and Hayes [29], a *compatible fault set* is defined as a set of functional faults that can be detected by a single test. Hence, the two corruptions of a channel by its logically adjacent neighbors form a *compatible fault set*.

3.1.8.2 Generalized Functional Fault Model

Comparison of the reduced functional faults for the *SOP* 4-to-1 multiplexer with a corresponding set of functional faults for a *POS* implementation yields a generalized functional fault model, not tied to a specific realization. Recall from Sections 3.1.4 and 3.1.5 that the channel corruptions originating in an *SOP* realization produced an ORing of adjacent channels, while a *POS* circuit caused an ANDing of those channels. A generalized set of functional faults and their corresponding test vectors are presented in Table 3-12. Note the combination of test vectors to form *compatible fault sets*.

Functional Fault	Remarks	Test Vector (S1 S0 D C B A)	
CHA-CORRUPT (by CHB)(AND)	CHA = CHA AND CHB	00 X001	
CHA-CORRUPT (by CHC)(AND)	CHA = CHA AND CHC	00 1001	
CHB-CORRUPT (by CHA)(AND) CHB = CHB AND CHA		01 0810	
CHB-CORRUPT (by CHD)(AND)	CHB = CHB AND CHD		
CHC-CORRUPT (by CHA)(AND)	CHC = CHC AND CHA		
CHC-CORRUPT (by CHD)(AND)	CHC = CHC AND CHD	IO OINO	
CHD-CORRUPT (by CHB)(AND)	CHD = CHD AND CHB	11 100X	
CHD-CORRUPT (by CHC)(AND)	CHD = CHD AND CHC	II IUUA	
CHA-CORRUPT (by CHB)(OR)	CHA = CHA OR CHB	00 ¥110	
CHA-CORRUPT (by CHC)(OR)	CHA = CHA OR CHC	UU AIIU	
CHB-CORRUPT (by CHA)(OR)	CHB = CHB OR CHA	01 1 1 1 0 1	
CHB-CORRUPT (by CHD)(OR)	CHB = CHB OR CHD	UT IAUI .	
CHC-CORRUPT (by CHA)(OR)	CHC = CHC OR CHA	10 10X1	
CHC-CORRUPT (by CHD)(OR)	CHC = CHC OR CHD	IO IOXI	
CHD-CORRUPT (by CHB)(OR)	CHD = CHD OR CHB	11 0118	
CHD-CORRUPT (by CHC)(OR)	CHD = CHD OR CHC		

Table 3-12 Generalized functional faults for example IF2.

3.1.8.3 Behavioral Fault Model

Examination of the relationship between the generalized set of functional faults and the initial behavioral description for example IF2 will result in a behavioral fault model for the expanded *if-then-else* construct.

The first *then* clause, $Z \ll Y0$, is activated when SEL="00", corresponding to *Channel A* of the multiplexer. This clause will be referred to as the *00-THEN* clause. From the functional analysis of the multiplexer in Section 3.1.8.2, *Channel A* was affected by four functional faults, *CHA-CORRUPT* (by *CHB*)(*AND*), *CHA-CORRUPT* (by *CHC*)(*AND*), *CHA-CORRUPT* (by *CHB*)(OR), and *CHA-CORRUPT* (by *CHC*)(*OR*). These faults can be mapped into the behavioral domain just like those from example IF1.

The functional fault CHA-CORRUPT (by CHB)(AND) represents corruption of Channel A by Channel B, which maps directly to the behavioral fault 00-THEN-CORRUPT (by 01)(AND). The faulty version of the then clause can be represented by ANDing the right hand side of the assignment statement corresponding to Channel A with the right hand side of the assignment statement from Channel B. The resulting faulty version of the assignment statement for the 00-THEN clause becomes $Z \leq YO$ **AND Y1**. Likewise, the corruption of Channel A by Channel C results in the definition of the behavioral fault 00-THEN-CORRUPT (by 10)(AND).

The (OR) corruptions of the 00-THEN clause are formed in a similar manner. Finally, the remaining *then* clauses each have four behavioral faults, corresponding to their respective channels. A subset of the resulting 16 behavioral faults, four for each of the four clauses/channels, is presented in Figure 3-10.

00-THEN-CORRUPT (by 01)(OR)	00-THEN-CORRUPT (by 10)(OR)
if SEL = "00" then	if SEL = "00" then
Z <= Y0 OR Y1 ;	Z <= Y0 OR Y2;
elsif SEL = "01" then	elsif SEL = "01" then
Z <= Y1;	Z <= Y1;
elsif SEL = "10" then	elsif SEL = "10" then
Z <= Y2;	Z <= Y2;
elsif SEL = "11" then	elsif SEL = "11" then
Z <= Y3;	Z <= Y3;
end if;	end if;

Figure 3-10 Control fault model for expanded if-then-else.

01-THEN-CORRUPT (by 00)(OR)	01-THEN-CORRUPT (by 11)(OR)
if SEL = "00" then	if SEL = "00" then
Z <= Y0;	Z <= Y0;
elsif SEL = "01" then	elsif SEL = "01" then
Z <= Y1 or y0 ;	Z <= Y1 OR Y3;
elsif SEL = "10" then	elsif SEL = "10" then
Z <= Y2;	Z <= Y2;
elsif SEL = "11" then	elsif SEL = "11" then
Z <= Y3;	Z <= Y3;
end if;	end if;

Figure 3-10 Control fault model for expanded if-then-else.

3.1.8.4 Evaluation of the Behavioral Fault Model

To evaluate the effectiveness of the behavioral fault model for the expanded *if-thenelse* construct, it will be compared to the underlying *SSL* gate level faults it was meant to encompass. Test vectors will be determined for each behavioral fault and the gate level fault coverage of these test vectors will be examined.

Testing for the first behavioral fault from Figure 3-10, 00-THEN-CORRUPT (by 01)(OR), requires activation of the first then clause with SEL="00". The fault free behavior, $Z \le Y0$, must be compared to the faulty behavior, $Z \le Y0$ OR Y1. Setting Y0=0 will result in Z=0 for an uncorrupted channel. If Y1=1, then a corruption of the channel by the adjacent channel, 01-THEN, will result in Z=1, thus detecting the fault. Since the other two clauses do not cause any corruptions with this fault, the other inputs (Y3, Y2) can remain don't cares. The resulting test vector (SEL Y3 Y2 Y1 Y0) is 00XX10. By similar analysis, the other (OR) corruption fault for this clause, 00-THEN-CORRUPT (by 10)(OR), requires the test vector 00X1X0.

Recall that the functional faults for *Channel-CORRUPT* formed a compatible fault set, one that could be detected by a single test vector. Likewise, the two *Clause-CORRUPT* behavioral faults should also form a compatible fault set. The previous analysis determined that the two test vectors necessary to detect the two *Clause-CORRUPT* (*OR*) faults for the *00-THEN* clause were 00XX10 and 00X1X0. Comparison of the two test vectors confirms that they are, in fact, compatible. Elimination of don't cares for *Y2* and *Y1* produces a combined test vector 00X110. The behavioral faults and their corresponding test vectors are summarized in Table 3-13.

Behavioral Fault	Test Vector (SEL Y3 Y2 Y1 Y0)	
00-THEN-CORRUPT (by 01)(AND)	00 X001	
00-THEN-CORRUPT (by 10)(AND)		
01-THEN-CORRUPT (by 00)(AND)	01 0X10	
01-THEN-CORRUPT (by 11)(AND)		
10-THEN-CORRUPT (by 00)(AND)	10 01 ¥0	
10-THEN-CORRUPT (by 11)(AND)		
11-THEN-CORRUPT (by 01)(AND)	11 100 X	
11-THEN-CORRUPT (by 10)(AND)		
00-THEN-CORRUPT (by 01)(OR)	00 X110	
00-THEN-CORRUPT (by 10)(OR)		
01-THEN-CORRUPT (by 00)(OR)	01 1X01	
01-THEN-CORRUPT (by 11)(OR)		
10-THEN-CORRUPT (by 00)(OR)	10 1021	
10-THEN-CORRUPT (by 11)(OR)	TO TOXT	
11-THEN-CORRUPT (by 01)(OR)	11 011X	
11-THEN-CORRUPT (by 10)(OR)		

 Table 3-13 Behavioral test vectors for example IF2.

The behavioral test vectors from Table 3-13 provide complete *SSL* gate level fault coverage for both *SOP* and *POS* implementations of example IF2. The results confirm that the control fault model for the expanded *if-then-else* construct is a valid abstraction into the behavioral domain of the original gate level *SSL* faults.

3.1.9 Summary

A new behavioral fault model has been developed for the control construct *if-then-else*. This new control fault model specifies that each clause of an *if-then-else* statement can be affected by two different types of faults, *Clause-CORRUPT (OR)* and *Clause-CORRUPT (AND)*. The actual number of *Clause-CORRUPT* faults depends on the size of the model and the resulting number of logical adjacencies between clauses.

In the VHDL behavioral description, a *Clause-CORRUPT (OR)* fault is represented by considering the effect of the corrupting clause. The right hand side of the assignment statement for the corrupted clause is ORed with the right hand side of the assignment statement for the corrupting clause. A test vector for this fault is determined by setting the fault free behavior of the uncorrupted clause to '0'. The corrupting clause is then set to '1', thus producing a conflict with the fault free behavior. Multiple *Clause-CORRUPT* (*OR*) faults may affect the same clause, due to logical adjacencies between clauses. These faults can form a compatible fault set and their test vectors can, therefore, be combined to produce a single test for the corruption of that clause.

The faulty operation of a clause due to a *Clause-CORRUPT (AND)* fault is represented in a similar manner. The right hand side of the assignment statement for the corrupted clause is ANDed with the right hand side of the assignment statement for the corrupting clause. A test vector forces the fault free behavior of the uncorrupted clause to produce an output of '1', while the corrupting clause is set to produce an output of '0'.

Test vectors generated from these behavioral faults can be applied to gate level implementations of the behavioral descriptions. Examples have shown the ability of these test vectors to detect underlying gate level *SSL* faults in synthesized circuits. Analysis has also shown that the test vectors from this new control fault model do provide improved gate level fault coverage over previous behavioral fault models.

3.2 CASE

The other VHDL control construct closely related to *if-then-else* is the *case* statement. The *case* statement allows selection of statements to execute depending on the value of a selection *expression*. Multiple *when* clauses can be used to allow selection among any number of *choices*. All *choices* must be distinct and all values must be represented in the *choice* lists, or the special *choice others* must be included as the last alternative.

The *case* statement is really just an alternative representation of the *if-then-else* construct with more restrictions. Use of *if-then-else* versus *case* is usually just a matter of programming style; any *case* statement can be rewritten as an equivalent *if-then-else*. The example in Figure 3-11 shows two equivalent behavioral descriptions, one using an *if* statement and the other using a *case* statement. Note that the *when* clauses perform the same function for the *case* statement as the *then* clauses in the *if* statement. The *when oth-*

	case SEL is
if SEL = "00" then	when "00" =>
Z <= Y0;	Z <= Y0;
elsif SEL = "01" then	when "01" =>
Z <= Y1;	Z <= Y1;
else	when others =>
Z <= Y2;	Z <= Y2;
end if;	end case;

ers clause ensures complete elaboration of *choices* for the *case* statement, just like the *else* clause for the *if* statement.

Figure 3-11 Equivalent if-then-else and case statements.

In Appendix E Armstrong and Gray and Mentor Graphics demonstrate the link between the *case* statement and the functional building block of a multiplexer. These examples reinforce the similarities between the control constructs *if-then-else* and *case*.

3.2.1 Application of the Control Fault Model

The simple example, CASE1, from Figure 3-11 will be used to demonstrate the application of the new control fault model to the *case* construct. The fault model specifies that each clause of an *if-then-else* statement, and hence of a *case* statement, can be affected by two different types of faults, *Clause-CORRUPT (OR)* and *Clause-CORRUPT (AND)*. The actual number of *Clause-CORRUPT* faults depends on the size of the model and the resulting number of logical adjacencies between clauses. *Clause-CORRUPT* faults are now designated *WHEN-CORRUPT* faults, identified by the appropriate *choice* as well as the corrupting clause.

In order to specify the WHEN-CORRUPT faults, the logical adjacencies between clauses must be identified. Figure 3-12 provides a graphical representation of the logical adjacencies between the clauses for example CASE1. The when others clause defines the behavior for all choices not explicitly specified in previous clauses. Figure 3-12 indicates that each clause of the case statement is adjacent to two other clauses. Hence, there are two WHEN-CORRUPT (OR) faults and two WHEN-CORRUPT (AND) faults for each of the three clauses in the example.


Figure 3-12 Logical adjacencies among clauses.

The when others clause from example CASE1 corresponds to the control input combination SEL="1X". In order to specify an adjacency between the when others clause and another clause, the don't care must be eliminated. For example, the clause when "00" should be logically adjacent to the clauses when "01" and when "10". Though the clause when "10" does not explicitly exist, it is created by the designation of the don't care for the when others clause as SEL(0)=0. The resulting adjancies and WHEN-CORRUPT (AND) faults are shown in Figure 3-13. An additional six behavioral faults for WHEN-CORRUPT (OR) faults are determined in a similar manner.

```
WHEN-00-CORRUPT (by 10)(AND)
WHEN-00-CORRUPT (by 01)(AND)
                                     case SEL is
   case SEL is
                                        when "00" =>
     when "00" =>
        Z <= Y0 AND Y1;
                                           Z \leq Y0 AND Y2;
                                       when "01" =>
     when "01" =>
        Z \ll Y1;
                                           Z \leq Y1;
                                        when others =>
     when others =>
        Z \leq Y2;
                                           Z \leq Y2;
                                     end case;
  end case;
                                  WHEN-01-CORRUPT (by 11)(AND)
WHEN-01-CORRUPT (by 00)(AND)
   case SEL is
                                     case SEL is
     when "00" =>
                                        when "00" =>
        Z \leq Y0;
                                           Z \leq Y0;
                                        when "01" =>
     when "01" =>
                                           Z <= Y1 AND Y2;
        Z \iff Y1 \text{ AND } Y0;
     when others =>
                                       when others =>
        Z <= Y2;
                                           Z \leq Y2;
                                     end case;
  end case;
```

Figure 3-13 WHEN-CORRUPT (AND) faults for example CASE1.

WHEN-10-CORRUPT (by 00)(AND)	WHEN-11-CORRUPT (by 01)(AND)
case SEL is	case SEL is
when "00" =>	when "00" =>
Z <= Y0;	Z <= Y0;
when "01" =>	when "01" =>
Z <= Y1;	Z <= Y1;
when others =>	when others =>
Z <= Y2 AND Y0;	Z <= Y2 AND Y1;
end case;	end case;

Figure 3-13 WHEN-CORRUPT (AND) faults for example CASE1.

Test vectors for the behavioral faults are derived using the methodology described in Section 3.1.9. Recall that vectors derived from corruptions to the same channel can form a *compatible fault set* with a single test. The resulting test vectors are listed in Table 3-14.

Behavioral Fault	Test Vector (SEL Y2 Y1 Y0)
WHEN-00-CORRUPT (by 01)(AND)	00 001
WHEN-00-CORRUPT (by 10)(AND)	00 001
WHEN-01-CORRUPT (by 00)(AND)	01 010
WHEN-01-CORRUPT (by 11)(AND)	01 010
WHEN-10-CORRUPT (by 00)(AND)	10 1X0
WHEN-11-CORRUPT (by 01)(AND)	11 10X
WHEN-00-CORRUPT (by 01)(OR)	00 110
WHEN-00-CORRUPT (by 10)(OR)	00 110
WHEN-01-CORRUPT (by 00)(OR)	01 101
WHEN-01-CORRUPT (by 11)(OR)	01 101
WHEN-10-CORRUPT (by 00)(OR)	10 OX1
WHEN-11-CORRUPT (by 01)(OR)	11 01X

 Table 3-14 Behavioral test vectors for example CASE1.

3.2.2 Evaluation of the Fault Model

The VHDL behavioral description for example CASE1 was synthesized using Mentor Graphics' AutoLogic II. The resulting multiplexer architecture is shown in Figure 3-14.



Figure 3-14 Synthesized Structure1 for example CASE1.

Fault simulations were then performed on the gate level circuit for Structure1 using the behavioral test vectors shown in WAVES format in Figure 3-15.

8 9	SEL Y	72	Y1	Y0	Z	: time	;
	Claus	se-	-CO	RRUI	?т(й	AND)	
00	001	1	:	500	ns	;	
01	010	1	:	500	ns	;	
10	1X0	1	:	500	ns	;	
11	10X	1	:	500	ns	;	
୫ (Claus	se-	-C0	RRUI	?T ((DR)	
00	110	0	:	500	ns	;	
01	101	0	:	500	ns	;	
10	0X1	0	:	500	ns	;	
11	01X	0	:	500	ns	;	

Figure 3-15 WAVES test vectors for example CASE1.

According to MIL-STD 883D [52], Structure1 contains 34 unique gate level *SSL* faults. Simulations are performed for each of the gate level faults and the fault is reported as detected when the circuit's output differs from the expected output. All *SSL* faults were detected by the behavioral test vectors in the multiplexer implementation of example CASE1. The fault coverage was next evaluated for an alternate gate level realization.

AutoLogic II was again used to synthesize and then optimize the VHDL behavioral description for example CASE1. The resulting circuit for Structure2 is shown in Figure 3-16. Note that the circuit contains a mix of NAND, OR, and NOT gates and does not directly relate to any of the circuits analyzed in the development of the behavioral fault models for the control constructs *if-then-else* or *case*.



Figure 3-16 Synthesized Structure2 for example CASE1.

Fault simulations were performed on the gate level circuit using the behavioral test vectors from Figure 3-15. The resulting fault coverage as a function of the input test vectors is shown in Figure 3-17. Though the actual shape of the graph may vary with the order of the application of the test vectors, the resulting fault coverage will be the same. Fault coverage for Structure2 was 34/34 = 100%, complete *SSL* gate level fault coverage.



Figure 3-17 Fault coverage for Structure2 of example CASE1.

3.2.3 Comparison with Previous Behavioral Fault Models

As was discussed in Chapter 2 of this dissertation, most higher level fault models also address the control construct *case*. However, contrary to *if-then-else* where there was a general consensus, varying fault models are proposed for the *case* statement. Armstrong [7][8][9][13][58][69] proposes the *dead clause* fault in which each clause in a *case* state-

ment fails to a no-operation. The *dead clause* fault model is also used by Chen [20][54]. Armstrong and Cho [22] later define the Behavioral Stuck-open (*BSO*) fault, combining *assignment control* faults and *dead clause* faults. In a *BSO* fault the value of the source expression of an assignment statement is not correctly transferred to its target.

Since Ghosh [18][27] bases his fault models on the programming language C, instead of VHDL, the equivalent to the *case* statement is the *switch* (*Id*). The corresponding fault model states that the *switch* may select either the cases corresponding to the lower or upper extremes of the *switch* identifier's value system, all, or none of the specified cases. Finally, Riesgo [60] proposes a fault model for *case-is-when* in which the expression which controls the *case* statement may be *stuck-at-"all possible values."*

Recall that the new behavioral fault model for the *case* statement was based on the fact that *if-then-else* and *case* represent similar selection activities leading to multiplexer architectures. Any *case* statement can be rewritten as an equivalent *if-then-else*, therefore, the faults models for the equivalent statements should be the same. Only Riesgo proposes a fault model for the *case* statement which is equivalent to the one proposed for *if-then-else*. The equivalent *stuck-control* fault model is *stuck-then/stuck-else*, which was shown to have deficiencies in detecting gate level faults by the analysis in Section 3.1.7.

Ghosh's fault model is based on multiple *stuck-at* faults on the control inputs resulting in only the upper and lower extremes of the *switch (Id)* statement. Individual *stuck-at* faults are not considered on control lines, causing intermediate values to be neglected. Hence, faults developed for a *switch* statement, will not be consistent with those developed for an equivalent *if-then-else*. Also, the hardware analogy, that the decoder for *Id* may fail such that a logic *1* is always asserted at all of the output ports, would correspond to the unlikely scenario of all the *channels* of the multiplexer being simultaneously *stuck-on*.

The *dead clause* and *BSO* faults proposed by Armstrong represent the farthest divergence from those proposed for *if-then-else*. This occurs even though Armstrong and Gray [10] demonstrate the similarities between *if-then-else* and *case* in their discussion of hardware implementation of VHDL constructs. The faulty version of a clause is represented by replacing the right hand side (source) of an assignment statement with the left hand side (target). For example, an assignment statement from example CASE2, $Z \leq Y1$,

would be faulted as $Z \ll Z$. The result is inconsistent with the faults that would be defined for an equivalent *if-then-else* construct.

3.3 Conclusions

A new behavioral fault model has been developed for the control constructs *if-thenelse* and *case*. The new fault model is more closely linked to underlying hardware by combining VHDL synthesis information with the industry standard *single-stuck-line* (SSL) fault model. Each clause of an *if-then-else* or *case* statement can be affected by 2 different types of faults, *Clause-CORRUPT* (*OR*) and *Clause-CORRUPT* (*AND*). The actual number of *Clause-CORRUPT* faults depends on the size of the model and the resulting number of logical adjacencies between clauses.

Test vectors derived from these control faults can be applied to gate level implementations of the VHDL behavioral descriptions. Examples have shown the ability of these test vectors to detect underlying gate level *SSL* faults in synthesized circuits. Because the behavioral faults were derived from a functional analysis of the selection activity of multiplexers, they provide complete gate level fault coverage over a broad range of implementations. Detailed analysis has shown that the test vectors from this new control fault model do provide improved gate level fault coverage over previous behavioral fault models.

The new control fault model developed in this chapter provides two improvements over previous behavioral fault models. First, detection of low level *SSL* faults is improved by linking the selection activity of control constructs to the functional building block of a multiplexer. Finally, the control constructs *if-then-else* and *case* have been brought together in a single consistent fault model, where each clause can be affected by a some number of *Clause-CORRUPT* faults.

Chapter 4 Relational Operators

The only *relational operator* used thus far has been the "=" as part of the *condition* which controlled the *if* statement. The inclusion of other *relational operators* such as ">" and "<" implies the use of a comparator module in hardware. The comparison function will first be analyzed, using the techniques developed for the multiplexing function, in order to determine a generalized set of functional faults. These faults will then be abstracted into the behavioral domain by examination of the relationship between the functional faults and the behavior of the *relational operators*.

Because of similarities in their functions, the predefined VHDL relational operators from Table 1-2 can be divided into two groups for analysis. The relation A > B (GT) is the same as B < A (LT), hence only one of these operators needs to be analyzed. The relation A > B (GT) is the opposite of A <= B (LE); the hardware need only differ by a single inverter. Likewise A < B (LT) is related to A >= B (GE). Therefore, the set of generalized functional faults developed for the GT function can be used as a basis for all of the above relations (GT, LT, GE, LE). The same argument implies that the relation A = B (EQ) can be analyzed to find functional faults which also apply to the function $A \neq B$ (NE).

4.1 Greater Than (GT)

Several common implementations of the *GT* function were first analyzed to assess the "worst case" for functional faults. Abstraction of these faults into the behavioral domain should provide complete gate level *SSL* fault coverage over a broad range of possible realizations of the comparison functions. Gate level circuits were examined for 2-level (*SOP* and *POS*), factored, and cascade implementations.

A comparison of functional faults indicates that the test vectors required to cover all gate level faults in either the factored or cascade implementations are a subset of the vectors for the 2-level realizations. Furthermore, a common set of test vectors provides complete gate level fault coverage in either the *SOP* or *POS* circuits. Therefore, just like the analysis of the multiplexing function, the *SOP* and *POS* implementations will again be used to form a generalized set of functional faults for the GT function.

4.1.1 Generalized Functional Faults

The SOP and POS implementations of the 2-bit A > B function can be obtained from the analysis of the Karnaugh map shown in Figure 4-1.



Figure 4-1 Karnaugh map for 2-bit GT function.

The groupings of minterms are shown with dashed lines and produce the SOP function $GT = A_1\overline{B_1} + A_0\overline{B_1}\overline{B_0} + A_1A_0\overline{B_0}$. A functional analysis of the gate level SOP circuit produces a reduced set of functional faults shown in Table 4-1. Test vectors are shown in base-4 for ease of magnitude comparison of A and B.

Fault	Faulty Output	Test Vectors (AB)
G-1	1	00
G-0	0	10
I-1	1	11
H-1	1	12
E-0	0	20, 21, 31
K- 1	1	22
J-0	0	32
L-1	1	33

 Table 4-1 Reduced functional faults for SOP GT.

Note that a faulty output of 0 occurs when A is in fact greater than B (e.g. 32), yet the gate level circuit fails to give a proper indication of GT = 1. Likewise a faulty output of 1 occurs when A is not greater than B (e.g. 22).

Grouping of the maxterms in the Karnaugh map in Figure 4-1, shown with solid lines, produces the POS function $GT = (A_1 + A_0)(A_1 + \overline{B_1})(A_1 + \overline{B_0})(A_0 + \overline{B_1})(\overline{B_1} + \overline{B_0})$. A functional analysis of the gate level POS circuit produces the reduced set of functional faults shown in Table 4-2.

Fault	Faulty Output	Test Vectors
E-1	1	00
F-0	. 0	10
I-1	1	11
G-1	1	12
E-0	0	20, 21
I-0	0	21, 31
K-1	1	22
G-0	0	32
M-1	1	33

Table 4-2 Reduced functional faults for POS GT.

Examination of the two sets of functional faults indicates that the seven mandatory test vectors are the same for both the *SOP* and *POS* circuits. Note that a single input combination (21) will detect the remaining faults in either implementation. Thus, a set of eight generalized functional faults and test vectors is produced.

4.1.2 Classification of Functional Faults

In order to aid in the analysis and classification of the generalized set of functional faults, an alternate representation is shown in Figure 4-2. The columns of the figure indicate the base-4 value of input A, while the rows specify the value of B. The heavy line cutting through the figure represents the GT function; the shaded area above the line indicates

A is greater than B. The diagonal just below the heavy line indicates where A and B are equal.



Figure 4-2 Fault classes for 2-bit GT function.

The roman numerals in Figure 4-2 indicate the location of the test vectors for the eight functional faults derived in Section 4.1.1. Class I has been assigned to the test vectors where A and B are equal (00, 11, 22, 33). For this class, the faulty function gives an erroneous *TRUE* for the relation A > B. The other fault that produces an erroneous *TRUE* (12) has been designated as Class II. In this class, A is less than, rather than equal to, B. Finally, Class III has been assigned to the test vectors where A is in fact greater than B (10, 21, 32), but the faulty function gives an erroneous *FALSE*.

Classes I and III follow easily identifiable patterns, however, Class II is still somewhat vague. To gain additional insight into the make-up of Class II and to ensure that the initial classification of functional faults is valid, a functional analysis was performed on a 3-bit GT function. The resulting functional faults are classified in Figure 4-3.

As expected, the functional faults for Classes I and III follow the same patterns as the 2-bit case. The octal test vectors in Group II (12, 34, 56) provide the additional information necessary to identify a pattern. Note that in both Classes II and III, A and B differ by only 1; A + 1 = B or A = B + 1 respectively (+ indicates addition). By induction, these fault classifications can now be generalized for an n-bit GT function. The resulting functional faults and fault classes are presented in Table 4-3.

					A	ł			
·		0	1	2	3	4	5	6	7
	0	Ι	III			÷			
	1		Ι	Ш					
	2		II	Ι	Ш				
D	3				Ι	Ш			
Б	4				Π	Ι	Ш		
	5						Ι	Ш	
	6						II	I ·	III
	7								Ι

Figure 4-3 Fault classes for 3-bit GT function.

Class	Faulty Output	A vs. B	# Test Vectors
Ι	TRUE	A = B	2 ⁿ
II	TRUE	A + 1 = B(A odd)	2 ⁿ⁻¹ - 1
III	FALSE	$\mathbf{A} = \mathbf{B} + 1$	2 ⁿ - 1

Table 4-3 Functional faults for n-bit GT.

4.1.3 Behavioral Fault Model

Now that the three classes of functional faults have been identified, they can be abstracted into the behavioral domain by examining the relationship between the faults and the original VHDL operator (>). Appendix C gives the details of usage for *relational operators* in *expressions*, which form the *conditions* for the *if* statement. The *expressions* yield Boolean results, which control the selection of the appropriate clause. In order to model a fault in a *relational operator*, the controlling *expression* needs to be modified such that it produces an erroneous result (*TRUE* or *FALSE*) corresponding to that fault.

First consider the four Class I faults for a 2-bit *GT* function. The fault-free controlling *condition* can be written as:

if A > B then

Class I faults produce an erroneous TRUE when A = B, so the initial inclination would be to simply modify the *condition* to read:

if (A > B) OR (A = B) then

However, this has the effect of producing a single behavioral fault which can be detected by any one of the four required test vectors. What is needed is a distinct behavioral fault for each test vector. Hence, the first Class I behavioral fault can be rewritten as:

if (A > B) OR ((A = B) AND (A = "00") then

The other three Class I faults are modeled by enumeration of the appropriate values for A.

Class II faults also produce an erroneous TRUE, this time when A + 1 = B (A odd). Hence the single Class II fault for the 2-bit GT function can be modeled as:

if (A > B) OR ((A + 1 = B) AND (A = "01")) then

Finally, Class III faults produce an erroneous *FALSE* when A = B + 1. To model these faults in the behavioral domain, the original expression can be ANDed with 0 for the appropriate input combination. The NAND function produces the required behavior, giving the following model for the first Class III fault:

if (A > B) AND ((A = B + 1) NAND (B = "00")) then

Class	Faulty Expression	Faults		
I	(A > B) OR ((A = B) AND (A = "00"))	A = "00" A = "01" A = "10" A = "11"		
П	(A > B) OR $((A + 1 = B)$ AND $(A = "01"))$	A = "01"		
III	(A > B) AND ((A = B + 1) NAND (B = "00"))	B = "00" B = "01" B = "10"		

The complete set of Class III faults is modeled by enumeration of the 2^n - 1 values for B.

Table 4-4 Behavioral faults for 2-bit GT.

4.1.4 Adapting the Model for GE, LT, and LE

The three fault classes and their associated behavioral fault models can be easily adapted for the other *relational operators* in this group (*GE*, *LT*, *LE*). As an example, the functional faults for the relation $A \ge B$ (*GE*) are presented in Figure 4-4. The thick line representing the *GE* function now lies below the diagonal of the figure. The locations of the Class I faults remain unchanged, however they now represent an erroneous *FALSE* for the *GE* function. Class II faults still produce the same erroneous result as Class I faults and the Class III faults now result in an incorrect *TRUE*.



Figure 4-4 Fault classes for 2-bit GE function.

The functional faults for the GE function can now be summarized in Table 4-5 and result in the definition of the behavioral faults in Table 4-6.

Class	Faulty Output	A vs. B	# Test Vectors
Ι	FALSE	A = B	2 ⁿ
Π	FALSE	A = B + 1 (B odd)	2 ⁿ⁻¹ - 1
III	TRUE	A + 1 = B	2 ⁿ - 1

Table 4-5 Functional faults for n-bit GE.

Similar analysis produces the generalized functional faults for the LE and LT functions. Since these functions differ by only a single inverter from GT and GE respectively, the behavioral fault models follow directly by inversion of the faulty output values.

Class	Faulty Expression	Faults		
I	(A >= B) AND ((A = B) NAND (A = "00"))	A = "00" A = "01" A = "10" A = "11"		
Π	$(A \ge B)$ AND $((A = B + 1)$ NAND $(B = "01"))$	B = "01"		
ш	(A >= B) OR ((A + 1 = B) AND (A = "00"))	A = "00" A = "01" A = "10"		

 Table 4-6 Behavioral faults for 2-bit GE.

4.1.5 Summary

A set of functional and behavioral faults have been developed for the first group of relational operators (*GT*, *GE*, *LT*, *LE*). Each operator is affected by three classes of faults. Class I faults occur when the *operands* of the relation are equal. Class II faults produce the same faulty output as Class I faults. Finally, Class III faults produce the opposite faulty output from Classes I and II.

4.2 Threshold Detection

In the previous analysis, both *operands* for the *relational operators* were *signals/vari-ables*. When one *operand* is a *constant*, the behavior changes to that of a *unary operator* or threshold detector. Treated individually, each threshold value represents a separate function with its own set of generalized faults. The analysis presented here examines those faults as a whole in order to identify patterns in the required test vectors.

4.2.1 Greater Than Signed Threshold

For this development, a signed comparison of a 4-bit number was chosen. The $2^4 = 16$ threshold values provide enough data to identify patterns in the test vectors. Use of 2's complement numbers helps demonstrate the general applicability of the functional analysis techniques.

A 4-bit 2's complement number represents the integer range from -8 to +7. Each threshold function (A > -8, A > -7, ..., A > 7) was analyzed (*SOP* and *POS*) to produce a generalized set of functional faults. A subset of these results is presented in Figure 4-5.

A > -5	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > -4	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > -3	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > -2	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > -1	-8	-7	-6	-5	-4	-3	-2	-1	.0	+1	+2	+3	+4	+5	+6	+7
A > 0	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > +1	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > +2	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7
A > +3	-8	-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7

Each threshold is indicated by a heavy vertical line, while the functional test vectors are shaded.

Figure 4-5 Functional test vectors for signed GT threshold.

From these test vectors it is possible to identify a pattern which will produce a behavioral fault model for the threshold functions. Every function requires the two test vectors bracketing the threshold. Each pattern of test vectors is then based on the binary value of the distance the threshold function is from the center of the range of values, indicated by the double lines. Additional test vectors are determined by moving left and right of the initial two test vectors at step sizes starting with the location of the 1s in a binary representation of the distance from center and increasing by powers of 2. The pattern is symmetrical about the threshold, but truncated beyond the far side of the center line.

For example, the threshold for A > -2 requires the test vectors -2 and -1 and is one space from the center. Therefore, the additional test vectors to the left of the threshold are -2 - 1 = -3 and -3 - 2 = -5. To the right of the threshold produces -1 + 1 = 0, with the pattern truncated beyond. For a threshold of A > 4, the initial test vectors are +4 and +5. The threshold is five spaces from center indicating step sizes of one and four. Thus, additional test vectors to the left are 4 - 1 = 3 and 3 - 4 = -1. The only test vector to the right is 5 + 1 = 6, because 6 + 4 = 12 is beyond the range of a 4-bit 2's complement number.

4.2.2 A Quick Example

A quick example will be used to demonstrate application of the behavioral fault patterns for threshold functions. Example LE5 in Figure 4-6 defines the behavior of a signed *less than or equal to (LE)* threshold function. The integer range -16 to +15 will be synthesized as a 5-bit 2's complement number.

```
entity LE5 is
  port(A: in INTEGER range -16 to +15; LE: out BIT);
end LE5;
architecture BEHAVE of LE5 is
begin
  process(A)
    begin
    if A <= 5 then
       LE <= `1';
    else
       LE <= `0';
    end if;
    end process;
end BEHAVE;
```

Figure 4-6 Behavioral description for example LE5.

The *LE* function places the threshold between +5 and +6. The threshold lies six spaces from the center of the range of values, implying step sizes of two and four. Therefore, additional test vectors to the left are 5 - 2 = 3 and 3 - 4 = -1. Test vectors to the right are 6 + 2 = 8 and 8 + 4 = 12. The test vector pattern is represented graphically in Figure 4-7.

A <= 5	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
			•						-			x 104			

Figure 4-7 Behavioral test vectors for example LE5.

Example LE5 was synthesized to the gate level circuit shown in Figure 4-8. Fault simulations using the behavioral test vectors from Figure 4-7 resulted in a *SSL* gate level fault coverage of 18/18 = 100%.



Figure 4-8 Synthesized circuit for example LE5.

The behavioral test vector patterns presented here are equally applicable to unsigned threshold comparisons. For more examples using comparison functions see Appendix A.

4.3 Equal (EQ)

The development of functional and behavioral faults for the EQ (and NE) function(s) follows the same process as that for GT. In addition, since EQ is often used in conjunction with other *relational operators*, the function is sometimes formed by a combination of the outputs of the other comparison modules.

4.3.1 Functional Faults

As was the case with the GT function, the patterns for the EQ function are somewhat vague in the 2-bit case. Therefore, the functional faults and classes will be directly presented for the 3-bit case and then generalized for n-bits. A functional analysis of gate level implementations of the EQ function yields the fault classes in Figure 4-9 (A and B shown in octal).

					A	ł			
		0	1	2	3	4	5	6	7
	0	I	III	Ш		Ш			
	1	Π							
	2	п							
D	3								
Б	4	Π							
	5			•					
	6								
	7								I

Figure 4-9 Fault classes for 3-bit EQ function.

Class I faults are again defined along the diagonal of the table, however, only two test vectors are required (00 and 77). Class II faults are selected to be those below the diagonal, while Class III faults are above. Both Class II and III fault sets are of size 3 (n).

The patterns apparent in the 3-bit case imply the definition of the functional faults in Table 4-7. The test vectors for Class I faults consist of the all 0's and all 1's cases. Both

Class II and III faults have test vectors where one *operand* is 0 and the other has a single 1, hence the n possible combinations.

Class	Faulty Output	A vs. B	# Test Vectors
Ĺ	FALSE	A = B = 0 $A = B = 2n - 1$	2
П	TRUE	$A = 0, B_i = 1$ i = 0, 1,, n-1	n
III	TRUE	$A_i = 1, B = 0$ i = 0, 1,, n-1	n

 Table 4-7 Functional faults for n-bit EQ.

4.3.2 Behavioral Fault Model

The functional faults developed for the EQ function can now be abstracted into the behavioral domain, just like those for GT. For the Class I faults, the faulty output of FALSE implies the use of an AND/NAND combination. Likewise, the Class II and III faults employ the OR/AND structure seen in the other fault models. The resulting behavioral faults for a 2-bit EQ function are presented in Table 4-8.

Class	Faulty Expression	Faults	
Ι	(A = B) AND $((A = 00'') NAND (B = 00''))$	A,B = "00" A,B = "11"	
П	(A = B) OR ((A = "00") AND (B = "01"))	B = "01" B = "10"	
III	(A = B) OR ((A = "01") AND (B = "00"))	A = "01" A = "10"	

Table 4-8 Behavioral faults for 2-bit EQ.

The functional and behavioral faults developed for the EQ function can now be readily adapted to NE. Due to the symmetry of the functions, the location of all the test vectors remains the same. Only the reversal of the faulty outputs causes a change in the behavioral fault model.

4.4 Comparison with Previous Fault Models

In contrast to *if-then-else* and *case*, few previous fault models address *relational operators*. As part of the development of the "B-algorithm: A Behavioral Test Generation Algorithm," [21][22] Cho and Armstrong developed a new behavioral fault model which included *relational operators* as part of *micro-operation* faults. Such faults perturb a *relational operator* to another operator as indicated in Table 4-9. Recall that this fault model suffered from what the authors called the "big micro-operation problem," where only a single test vector is generated for a complex block of logic. In order to raise their equivalent gate level coverage numbers to acceptable levels, an additional 4n-1 test vectors are generated by a heuristic test generator for each *n*-bit micro-operation.

Fault-free Operator	Faulty Operator
BVLT	BVGE
BVLE	BVGT
BVEQ	BVNEQ
BVNEQ	BVEQ

 Table 4-9 Micro-operation Faults

A similar fault model was proposed by Al Hayek and Robach [4] as part of a mutationbased testing strategy in which VHDL behavioral faults are considered as software faults. For Relational Operator Replacement (*ROR*), each operator (<, >, <=, >=, =, /=) is replaced by each of the other relational operators. Mutation analysis does not take into account the size of the hardware implementation, because it considers each relational operator as a software operation and consequently generates only one test vector. In order to improve the performance of their technique, the authors also resorted to heuristics to generate additional test vectors for their complex operators.

The new behavioral fault models developed for *relational operators* are based on the size of the hardware implementation and, therefore, eliminate the need to supplement test vector sets. While the new fault models are definitely more complex than previous ones, this is because they more accurately reflect the underlying complexity of the hardware faults which they attempt to model.

4.5 Application of the New Fault Models

A simple example is now presented to demonstrate the application of the new fault models developed for *relational operators*. A test vector set will be formed based on behavioral faults and then applied to synthesized gate level realizations. Gate level fault coverage of the behavioral test vectors will be used to evaluate the effectiveness of the new behavioral fault models.

Example COMPARE in Figure 4-10 uses two 2-bit signals (A, B) to control the selection of input signals (Y2, Y1, Y0). Based on the relative magnitudes of the control signals, a single input signal is assigned to the output signal (Z). Test vectors can be determined for behavioral faults on the *relational operators* and the control construct *if-then-else*.

```
if A > B then
    Z <= Y2;
elsif A < B then
    Z <= Y0;
else
    Z <= Y1;
end if;</pre>
```

Figure 4-10 Behavioral description for example COMPARE.

4.5.1 Faults on Relational Operators

The two *relational operators* (>, <) in example COMPARE are affected by the three classes of behavioral faults developed earlier in this chapter. Application of each of these fault classes to each of the *relational operators* will determine an appropriate set of behavioral test vectors.

The first relational operator in the expression A > B controls the *then* clause of the *if-then-else* statement. According to the new behavioral fault model, a Class I fault occurs when the relation A > B produces an erroneous *TRUE* when A = B. This fault causes the selection of the A > B clause instead of the desired A = B clause. The test vectors necessary to detect these Class I faults are shown in Table 4-10. For ease of reference when determining compatible fault sets, the faults have been numbered.

Next, a Class II fault for the relation A > B also produces an erroneous *TRUE*, when A + 1 = B (A odd). This fault causes the selection of the A > B clause instead of the

desired A < B clause. Test vectors for the single Class II fault are shown in the middle of Table 4-10.

Finally, a Class III fault produces an erroneous *FALSE* when A = B + 1. This fault causes the selection of clauses corresponding to A <= B instead of the desired A > B. Note that the faulty selection of either the A < B clause or the A = B clause results in the elimination of don't care values seen in previous test vectors. The Class III faults and their corresponding test vectors are shown at the bottom of Table 4-10.

Fault Class	Fault #	А	В	(A>B) Y2	(A=B) Y1	(A <b) Y0</b) 	Fault-free Z
	1	00	00	1	0	Х	0
	1	00	00	0	1	X	1
	2	01	01	1	0	X	0
Close I	4	01	01	0	1	X	1
Class I	2	10	10	1	0	X	0
	3	10	10	0	1	X	1
	4	11	11	1	0	X	0
	4	11	11	0	1	X	1
Class II	5	01	10	1	X	0	0
	3	01	10	0	X	1	1
	6	01	00	0	1	1	0
	U	01	00	1	0	0	1
Class III	7	10	01	0	1	1	0
Class III	/	10	01	1	0	0	1
	0	11	10	0	1	1	0
	o		10	1	0	0	1

Table 4-10 Test vectors for behavioral faults for A > B.

A similar application of Class I, II, and III faults to the relation A < B in the *elsif* clause produces the test vectors shown in Table 4-11.

Fault Class	Fault #	А	В	(A>B) Y2	(A=B) Y1	(A <b) Y0</b) 	Fault-free Z
	0	00	00	X	0	1	0
	,	00	00	X	1	0	1
	10	01	01	X	0	1	0
Close I	10	01	01	X	1	0	1
Class I	11	10	10	X	0	1	0
	11	10	10	X	1	0	1
	10	11	11	X	0	1	0
	14	11		X	1	0	1
Class II	13	⁻ 10	01	0	Х	1	0
	15	10		1	X	0	1
	14	00	01	1	1	0	0
	17	00	01	0	0	1	1
Class III	15	01	10	1	1	0	0
	15	01	10	0	0	1	1
	16	10	11	1	1	0	0
	10		TT	0	0	1	1

Table 4-11 Test vectors for behavioral faults for A < B.

Each behavioral fault still has two possible test vectors which will detect it. Application of the control fault model to the *if-then-else* construct will provide additional guidance on selection of a final set of test vectors.

4.5.2 Control Faults

As was shown in Chapter 3, the control fault model specifies that each clause of an *if*then-else statement can be affected by two different types of faults, *Clause-CORRUPT* (*OR*) and *Clause-CORRUPT* (*AND*). Corruptions are caused by clauses which are *logically adjacent* to the affected clause. In the case of *relational operators*, it has been shown that the ">" operator can fault to "=" (Class I) or "<" (Class II), thus causing the *then* clause in example COMPARE to corrupt the either *else* clause or the *elsif* clause. By establishing adjacency using the appropriate values for A and B, determined by faults to

the *relational operators*, compatible test vectors can be formed for these *Clause-COR-RUPT* faults.

First consider the corruptions to the *then* clause by the *elsif* clause. These corruptions occur when the A < B expression in the elsif clause produces an erroneous indication that A > B, corresponding to a Class II fault for A < B. Table 4-11 shows that this Class II fault, 13, has only one combination of A and B which will produce the proper adjacency between clauses. Using those values for A and B results in the test vectors shown in Table 4-12 for the first two of the *Clause-CORRUPT (OR)* and *Clause-COR-RUPT (AND)* faults for the *then* clause.

Control Fault	Fault Class	A	В	then Y2	else Y1	elsif Y0	Fault-free Z
THEN-CORRUPT (by ELSIF) (OR)	A < B	10	01	0	X	1	0
THEN-CORRUPT (by ELSIF) (AND)	Class II	10	01	1	X	0	1
THEN-CORRUPT (by ELSE) (OR)	A > B	11	10	0	1	x	0
THEN-CORRUPT (by ELSE) (AND)	Class III	01	00	1	0	х	1

 Table 4-12 Test vectors for THEN-CORRUPT faults.

The corruptions of the *then* clause by the *else* clause, correspond to the other combinations from Table 4-10 and Table 4-11 where the *then* clause is activated, Class III faults for A > B. Table 4-10 shows that faults **6** through **8** provide three combinations of A and B which will produce the proper adjacency between clauses. To reduce the total number of test vectors using *compatible fault* sets, the vectors for the *Clause-CORRUPT (OR)* and *Clause-CORRUPT (AND)* faults are chosen with different values for A and B. These test vectors form the remainder of Table 4-12.

The *Clause-CORRUPT* faults for the *elsif* and *else* clauses are formed in a similar manner. For the *elsif* clause, the test vectors must set A < B, corresponding to Class II faults for A > B and Class III faults for A < B. Finally, the *else* clause corresponds to

Control Fault	Fault Class	A	В	then Y2	else Y1	elsif Y0	Fault-free Z
ELSIF-CORRUPT (by THEN) (OR)	A > B	01	10	1	X	0	0
ELSIF-CORRUPT (by THEN) (AND)	Class II	UI		0	X	1	1
ELSIF-CORRUPT (by ELSE) (OR)	A < B	10	11	X	1	0	0
ELSIF-CORRUPT (by ELSE) (AND)	Class III	00	01	X	0	1	1
ELSE-CORRUPT (by THEN) (OR)	A > B	00	00	1	0	x	0
ELSE-CORRUPT (by THEN) (AND)	Class I	01	01	0	1	x	1
ELSE-CORRUPT (by ELSIF) (OR)	A < B	10	10	X	0	1	0
ELSE-CORRUPT (by ELSIF) (AND)	Class I	11	11	x	1	0	1

A = B and the Class I faults. Test vectors for the remaining *Clause-CORRUPT* faults are, therefore, shown in Table 4-13.

 Table 4-13 Test vectors for ELSIF-CORRUPT and ELSE-CORRUPT faults.

4.5.3 Final Behavioral Test Vector Set

The behavioral faults for the *relational operators* can now be combined with the control faults to form *compatible fault* sets. Recall that Class I faults occur along the diagonal where A = B. Notice that the don't care values in the test vectors for faults 1 through 4 are compatible with the test vectors for faults 9 through 12. Plus, due to the symmetry of the behavioral faults defined for *relational operators*, the Class II faults for the relation A > B are a compatible subset of the Class III faults for the relation A < B. Likewise, the Class II faults for A < B are a compatible subset of the Class III faults for A > B.

Note that the *THEN-CORRUPT* (by *ELSIF*) and *ELSIF-CORRUPT* (by *THEN*) faults dictate that both options of test vectors for faults **13** and **5** be chosen. For each of the other *relational operator* faults, only one of the two possible test vectors is needed. For this

example, the final set of test vectors has been chosen with a pattern of alternating fault free values of Z = 0 and Z = 1. This pattern can be easily repeated for vectors that are not specified by control faults, as will be the case where A and B are larger than two bits.

Application of this selection pattern to each group of faults for the *relational operators* results in a final behavioral test vector set containing 12 test vectors. The test vectors and their corresponding groups are shown in Table 4-14. Note that other test vector sets of size 12 are possible. The required values of A and B are set by the Class I, II and III faults for the *relational operators*, however several choices are possible for Y2, Y1, and Y0. As long as both a Z = 0 and Z = 1 option are chosen for each fault class, sufficient coverage will be provided for the *Clause-CORRUPT* control faults.

Group # (Control Fault)	Fault #s	A	В	(A>B) Y2	(A=B) Y1	(A <b) Y0</b) 	Z
	1,9	00	00	1	0	1	0
Ι	2, 10	01	01	0	1	0	1
(ELSE-CORRUPT)	3, 11	10	10	1	0	1	0
	4, 12	11	11	0	1	0	1
	6	01	00	1	0	0	1
II	7 12	10	01	0	1	1	0
(THEN-CORRUPT)	7,15	10	01	1	0	0	1
	8	11	10	0	1	1	0
	14	00	01	0	0	1	1
III	5 15	01	10	1	1	0	0
(ELSIF CORRUPT)	5, 15	01	10	0	0	1	1
	16	10	11	1	1	0	0

 Table 4-14 Final behavioral test vector set for example COMPARE.

4.6 Evaluation of Behavioral Test Vectors

The test vectors derived from the behavioral faults for example COMPARE are next applied to several synthesized gate level implementations. *SSL* fault coverage will be determined and used to judge the effectiveness of the behavioral test vectors.

4.6.1 Gate Level Realizations

The VHDL behavioral description for example COMPARE, from Figure 4-10, was synthesized to a gate level implementation using AutoLogic II. The first structural description was produced using minimal optimization in order to produce the most direct realization of the circuit. The gate level circuit for Structure1 is presented in Figure 4-11. Note that the circuit contains a mix of AND, OR, NOT, NAND, and NOR gates and does not directly relate to any of the circuits analyzed in the development of the behavioral fault models for the *relational operators*.



Figure 4-11 Synthesized Structure1 for example COMPARE.

According to MIL-STD 883D, Structure1 contains 74 unique gate level SSL faults. Fault simulations were performed using the behavioral test vectors from Table 4-14. The behavioral test vector set detected 73 of the 74 SSL gate level faults, resulting in a fault coverage of 98.65%. The undetected fault was on input B to NAND gate zg3 shown in Figure 4-11. Exhaustive testing of the circuit Structure1 reveals that this fault is, in fact, undetectable by any test vector due to redundant logic produced by the synthesis tool. In this case, an alternate measure of effectiveness can also be used to account for redundant logic. The fault efficiency is defined as the ratio of detected faults to detectable faults. For this example the fault efficiency is 73/73 = 100%.

An alternate test vector set was formed by reversing the order of the Z = 0, Z = 1 pattern in Table 4-14. Application of this alternate set of behavioral test vectors, shown in Figure 4-12, to the circuit Structure1 produced an identical fault coverage of 98.65%. Either set of test vectors developed from the behavioral fault models, therefore, achieved a *fault efficiency* of 100%.

```
% AB Y2 Y1 Y0 Z : time ;
% Group I
0000 010 1 : 500 ns ;
0101 101 0 : 500 ns ;
1010 010 1 : 500 ns ;
1111 101 0 : 500 ns ;
Group II
0100 011 0 : 500 ns ;
1001 100 1
             500 ns
           :
1001 011 0 : 500 ns ;
1110 100 1 : 500 ns ;
% Group III
0001 110 0 : 500 ns ;
0110 001 1 : 500 ns ;
0110 110 0 : 500 ns ;
1011 001 1 : 500 ns ;
```

Figure 4-12 Alternate set of behavioral test vectors for example COMPARE.

A second gate level implementation of example COMPARE was produced by allowing AutoLogic II to perform logic optimizations. Using these optimizations in a synthesis environment allows a designer to remove redundancies and reduce the number of undetectable faults. The resulting Structure2 is shown in Figure 4-13.



Figure 4-13 Synthesized Structure2 for example COMPARE.

Structure2 contains 72 unique SSL gate level faults. Fault simulations using the original and alternate behavioral test vectors from Table 4-14 and Figure 4-12 both result in a fault coverage of 72/72 = 100%. The fault coverage as a function of the alternate behavioral test vectors is shown in Figure 4-14.

When redundancies are not present in the gate level circuit, the test vector sets developed using the behavioral fault models achieve a gate level *SSL* fault coverage of 100%.





These results help validate the new behavioral fault models through practical application. The effects on fault coverage due to expansion of the control signals and data path will now be investigated.

4.6.2 Expansion of the Data Path

In the VHDL behavioral description for example COMPARE, the input signals (Y2, Y1, Y0) and the output signal (Z) are each only a single bit wide. To demonstrate the effects of a wider data path on the new behavioral fault models, example COMPARE4 was created by changing the above signals to four bits wide, BIT_VECTOR(3 downto 0). For this example, the control signals A and B remained two bits each.

The faults on the *relational operators* are unchanged due to the widening of the data path. Hence, the values of the control signals A and B are the same as those given in Table 4-14. The only change to the test vectors is the widening of the 1-bit signals to four bits. This is done by replication of the appropriate signal values, since the wider data path simply represents multiple copies of the 1-bit case implemented in parallel.

The control faults on adjacent clauses developed in Section 4.5.2 are, likewise, unaffected by the widening of the data path. The only modifications necessary to the test vec-

tors is again replication of the appropriate signal values. Hence, the behavioral test vectors for example COMPARE4 are presented in Figure 4-15.

% AB	Y2	Y1	Y0	Z	:	time	e ;	
% Gro	oup I		•					
0000	1111	0000	1111	0000	:	500	ns	;
0101	0000	1111	0000	1111	:	500	ns	;
1010	1111	0000	1111	0000	:	500	ns	;
1111	0000	1111	0000	1111	:	500	ns	;
% Gro	oup II	[
0100	1111	0000	0000	1111	:	500	ns	;
1001	0000	1111	1111	0000	:	500	ns	;
1001	1111	0000	0000	1111	:	500	ns	;
1110	0000	1111	1111	0000	:	500	ns	;
% Gro	oup II	II						
0001	0000	0000	1111	1111	:	500	ns	;
0110	1111	1111	0000	0000	:	500	ns	;
0110	0000	0000	1111	1111	:	500	ns	;
1011	1111	1111	0000	0000	:	500	ns	;

Figure 4-15 Behavioral test vectors for example COMPARE4.

Example COMPARE4 was synthesized and optimized with AutoLogic II to produce the gate level Structure shown in Figure 4-16. Fault simulations were performed using the test vectors derived from the behavioral fault models. The results show a fault coverage of 150/150 = 100%; all SSL gate level faults are detected.



Figure 4-16 Synthesized Structure for example COMPARE4.

As was done previously, an alternate test vector set was formed for example COMPARE4 by reversing the order of the Z = 0, Z = 1 pattern in Figure 4-15. Fault simulation using these alternate behavioral test vectors again resulted in gate level fault coverage of 100%. Though the number of unique gate level faults more than doubled, the same number of test vectors, 12, were able to provide complete *SSL* fault coverage.

4.6.3 Expansion of the Control Signals

Example COMPARE was next modified by increasing the width of the control signals A and B from two bits to three bits each. The change in size of the control signals does affect the behavioral faults for the *relational operators*. Recall the number of faults on a comparison is related to the number of bits being compared. The increase from two bits to three bits causes the number of Group I faults from Table 4-14 to increase from $2^2 = 4$ to $2^3 = 8$. The number of Group II and Group III faults likewise increases to $2^3 - 1 = 7$.

In contrast to the faults on *relational operators*, the control faults are affected only by adjacency among clauses. Recall from Table 4-13 that the *ELSIF-CORRUPT (by THEN)* faults corresponded to the A > B Class II faults. In the 2-bit case, only a single combination of A and B produced the proper adjacency, forcing the selection of both test vector options to cover both the *Clause-CORRUPT (OR)* and *Clause-CORRUPT (AND)* faults. In the 3 bit case, there are now $2^{3-1} - 1 = 3$ different combinations of A and B that produce the proper adjacency between clauses (see Figure 4-3). Through proper selection of test vectors from the faults on *relational operators*, it is possible to provide coverage for all control faults without replication of values for A and B. A final set of behavioral test vectors for example COMPARE3 is, hence, presented in Figure 4-17.

```
% A B Y2Y1Y0 Z : time ;
% Group I
000 000 101 0 : 500 ns ;
001 001 101 0 : 500 ns ;
010 010 010 1 : 500 ns ;
011 011 010 1 : 500 ns ;
100 100 101 0 : 500 ns ;
101 101 101 0 : 500 ns ;
110 110 010 1 : 500 ns ;
```



	% G1	coup	II						
	001	000	011	0	:	500	ns	;	
	010	001	011	0	:	500	ns	;	
	011	010	100	1	:	500	ns	;	
	100	011	100	1	:	500	ns	;	
	101	100	011	0	:	500	ns	;	
	110	101	011	0	:	500	ns	;	
	111	110	100	1	:	500	ns	;	
•	8 G1	coup	III						
	000	001	110	0	:	500	ns	;	
	001	010	110	0	:	500	ns	;	
	010	011	001	1	:	500	ns	;	
	011	100	001	1	:	500	ns	;	
	100	101	110	0	:	500	ns	;	
	101	110	110	0	:	500	ns	;	
	110	111	001	1	:	500	ns	;	

Figure 4-17 Behavioral test vectors for example COMPARE3.

Example COMPARE3 was then synthesized and optimized by AutoLogic II to produce the gate level implementation shown in Figure 4-18.



Figure 4-18 Synthesized Structure for example COMPARE3.

Fault simulations were conducted on the gate level Structure for example COMPARE3 using the derived set of behavioral test vectors. A *SSL* gate level fault coverage of 92/92 = 100% was achieved. As in previous examples, fault simulation with an alternate set of behavioral test vectors again resulted in complete gate level fault coverage.

4.7 Conclusions

New behavioral fault models have been developed for the predefined VHDL relational operators from Table 1-2. These fault models are based on a functional analysis of the comparison functions GT and EQ. The symmetry of these comparison functions allowed the resulting generalized functional faults to be easily adapted for all the relational operators.

The new behavioral fault models developed for *relational operators* are based on the size of the hardware implementation and, therefore, eliminate the need to supplement test vector sets via methods such as heuristics. While the new fault models are more complex than previous ones, this is because they more accurately reflect the underlying complexity of the hardware faults which they attempt to model.

A simple example was presented to demonstrate the application of the new fault models. Test vector sets were formed based on behavioral faults to the *relational operators* and the control construct *if-then-else*. These behavioral test vectors were then applied to synthesized gate level realizations. Gate level fault coverage was used to evaluate the effectiveness of the new behavioral fault models.

When redundancies were not present in the synthesized gate level circuits, both the primary and alternate test vector sets developed using the behavioral fault models produced a gate level *SSL* fault coverage of 100%. Even with undetectable faults, the behavioral test vectors were able to achieve a *fault efficiency* of 100%.

Chapter 5 Arithmetic Operators

Like the *relational operators*, *arithmetic operators* also generate large blocks of combinational logic. The predefined VHDL operators ADD (+) and SUB (-) are normally implemented by synthesis tools with standard library modules. Optimizations for speed or chip area may modify these building blocks, however, the basic function of the *arithmetic operators* remains unchanged. A fault modeling technique is proposed here based on complete functional testing of the arithmetic building blocks. By concentrating on functional testing, complete gate level *SSL* fault coverage should be obtained over a broad range of hardware implementations.

Previous behavioral modeling approaches, based on perturbing language constructs such as *ADD* to *SUB*, do not accurately reflect underlying hardware faults. In order to compensate for this "big micro-operation problem," alternate methods such as heuristics were used to supplement test vector sets to increase the equivalent gate level fault coverage. The new modeling technique presented in this chapter increases the complexity of the fault models for the *arithmetic operators*, providing a better representation of the faults which occur in actual hardware.

5.1 Addition

The ADD operation has several basic forms which will be investigated in succession. A two-level network would be the fastest, however, this circuit would require a large number of gates and gate inputs. It would be necessary to have 2^{2n} NAND gates of 2n + 1 inputs and one NAND gate of 2^{2n} inputs to add two *n*-bit numbers [41]. This number of gates and inputs is quite significant for even small values of *n*.

In contrast to this direct approach, adders are most commonly implemented by the interconnection of smaller functional building blocks. In its simplest from, a *half adder* (*HA*) is a multiple output combinational circuit which adds two bits to produce a *sum* and a *carry-out*. A *full adder* (*FA*) adds two binary digits and a *carry-in* from a previous stage. To speed up the combinational addition process, by reducing the rippling of carries between stages, methods such as *carry look-ahead* (*CLA*) are used.

5.1.1 Ripple Carry Adder

Behavioral modeling of the *ADD* operation has two basic forms, depending on the presence of an overall *carry-in* and *carry-out* for the resulting adder circuit. In its simplest form, the addition of two *n*-bit binary numbers can be represented as:

 $S \leq A + B;$

A 3-bit ripple carry implementation of this ADD operation is shown in Figure 5-1.



Figure 5-1 Ripple carry adder.

The *full adders* (FA_i) are 3-input 2-output combinational circuits, where CI and CO represent the *carry-in* and *carry-out* respectively. A truth table for the *full adder* function is presented in Table 5-1; **Test #** shown in octal. Similarly, the *half adder* (HA_0) is a 2-input 2-output circuit that can be considered a subset of the *full adder* function. The upper half of Table 5-1, where CI = 0, represents the truth table for the *half adder* function.

Test #	CI	A	В	S	CO
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

 Table 5-1
 Truth table for full adder.

5.1.1.1 Functional Testing

A functional testing strategy is presented which will ensure complete gate level *SSL* fault coverage for a broad range of *ripple carry* adder implementations. Because such realizations are one-dimensional cellular logic arrays, made up of 2- and 3-input functional building blocks, complete functional testing can be achieved by exhaustive testing of each module [44].

Table 5-2 presents the Phase I functional tests proposed for the 3-bit *ripple carry* adder. The left hand side of the table shows the *ADD* operation with the resulting *sum*. Non-zero carries between stages of the adder are shown by arrows above the columns. The right hand side of the table indicates the **Test #**, from Table 5-1, which is applied to each functional module (*FA*, *HA*) by the input test vectors. For example, the first row shows that A_i and B_i are 0 for every stage of the adder. Since the resulting *carry-out* for each stage is also 0, **Test 0** is applied to all modules of the adder.

A + B	Test #		
S S	FA ₂	FA ₁	HA ₀
$+ \frac{000}{000}$	0	0	0
$+ \frac{000}{111}$	1	1	1
$+\frac{111}{000}$ 111	2	2	2
$+\frac{111}{110}$	7	7	3

 Table 5-2 Phase I functional tests.

Note that the last row of Table 5-2 shows that both C_0 and C_1 are 1, resulting in Test 7 being applied to FA_1 and FA_2 . Since HA_0 only has 2 inputs, the last test from Phase I represents Test 3 for this module and concludes complete functional testing of the *half adder*. To continue the testing of the *full adder* modules, further vectors must be generated which will produce the remaining input and carry combinations. Consider **Test 5** and **Test 6**, which require opposite inputs for A and B with CI = 1. In order to start the *ripple carry* process between stages, HA_0 must generate a CO = 1. Since **Test 5** and **Test 6** produce a CO = 1, the carries are correctly propagated through the stages. Table 5-3 summarizes the results for the Phase II functional tests.

A + <u>B</u> S	Test #		
	FA ₂	FA ₁	HA ₀
$+\frac{11}{000}$	5	5	3
$+ \frac{001}{000}$	6	6	3

 Table 5-3 Phase II functional tests.

Only Test 3 and Test 4 remain to complete the testing of the *full adder* modules. Since Test 3 requires a CI = 0, yet produces a CO = 1, no single vector can provide Test 3 inputs to all stages. Likewise, Test 4 requires CI = 1 and produces CO = 0. Due to the observed symmetry, the functional tests for Phase III can be formed by interleaving Test 3 and Test 4 as shown in Table 5-4.

+ <u>B</u> S	Test #		
	FA ₂	FA ₁	HA ₀
$+\frac{101}{010}$	3	4	3
$+ \frac{010}{100}$	4	3	0

 Table 5-4 Phase III functional tests.
Hence, complete functional testing of all modules of the *ripple carry* adder in Figure 5-1 has been achieved with eight test vectors. Next, the scalability of the functional tests will be evaluated for larger adders.

5.1.1.2 Scalability

Because the *ripple carry* structure is made up entirely of cascaded HA and FA modules, the functional tests developed in Section 5.1.1.1 can be readily adapted to larger adders. Since the functional tests are based on the complete testing of each individual module, only eight test vectors are required regardless of the size of the inputs A and B [44]. A 4-bit addition will be used to demonstrate the scaling of the test vectors.

Table 5-5 presents the test vectors for a 4-bit *ripple carry* adder. The tests for Phases I and II are created by replication of the highest order bits of the 3-bit case. The test vectors for Phase III are produced by continuation of the alternating patterns caused by the interleaving of **Test 3** and **Test 4** for the *FA* modules. Note that both extensions of the functional tests can be continued for larger values of n, still requiring only eight test vectors.

				Test #			
Phase	А	В	S	FA ₃	FA ₂	FA ₁	HA ₀
	0000	0000	0000	0	0	0	0
т	0000	1111	1111	1	1	1	1
1	1111	0000	1111	2	2	2	2
	1111	1111	1110	7	7	7	3
п	0001	1111	0000	5	5	5	3
ш	1111	0001	0000	6	6	6	3
TTT	0101	0101	1010	4	3	4	3
111	1010	1010	0100	3	4	3	0

Table 5-5 Functional tests for 4-bit ripple carry adder.

5.1.1.3 Behavioral Fault Model

Comparison of the functional test vectors with the gate level faults detected provides some insight into the performance of the functional testing technique. For example, the test vector A = 0000, B = 0000 covers faults which correspond to the *sum* and *carry* bits of each stage producing an erroneous 1. Due to the relative bit positions of the various stages, this causes the resulting *sum* to be in error by either +1, +2, +4, or +8. Similar observations reveal that the functional faults for the *ripple carry* adder result in an output *sum* which is in error by $\pm 2^i$ (i = 0, 1, 2, 3).

A one-to-one correspondence cannot be established between the large number of functional faults which are covered by the relatively small number of eight functional test vectors. Therefore, a direct mapping cannot be made to produce a simple behavioral fault model. Rather, the approach taken here simply seeks to map the functional test vectors into error vectors which corrupt the *ADD* operation for the appropriate input combinations.

Again consider the functional test vector A = 0000, B = 0000. Corruption of the resulting *sum* can be achieved by producing an erroneous I in any of the bit positions. For this fault model, the corruptions are chosen to be to the least significant bit position. The *XOR* operator provides the desired corruption properties by inverting the appropriate bit when presented with an error vector of 0001.

A behavioral fault model for the first functional test vector is therefore proposed as:

 $D \le (A + B)$ XOR "000" & (A = "0000" AND B = "0000")

The concatenation operator (&) combines the TRUE/FALSE from the AND operator with leading 0's to produce the appropriate error vector. While this is not syntactically correct VHDL due to type differences, it presents the concept of the behavioral fault model. A complete implementation of the behavioral fault model using functions from the Mentor Graphics *std_logic_arith* library is presented in Figure 5-2.

D	<=	(A	+ 1	B)	XOR z	eı	co_exte	nđ(to	ວ_ຄ	stċ	llogic(
					(A	=	``0000 ″	AND	в		" 0000″)	OR
					(A	=	~ 0000″	AND	в	=	``1111″)	OR
					(A	=	`1111 ″	AND	в	=	``0000″)	OR
					(A	=	`1111 ″	AND	в	=	<i>"1111")</i>	OR
					(A	=	``0001 ″	AND	в	=	"1111 <i>"</i>)	OR
					(A	=	`1111 ″	AND	в	=	``0001 <i>"</i>)	OR
					(A	=	``0101 ″	AND	в	=	``0101 <i>''</i>)	OR
					(A	=	`1010 ″	AND	в	=	`1010″)), 4);



5.1.1.4 Evaluation of the Behavioral Test Vectors

The behavioral test vectors derived in the preceding sections will now be applied to gate level implementations of *ripple carry* adders. Fault simulations will determine *SSL* fault coverage and demonstrate the effectiveness of this functional testing technique.

The VHDL behavioral description for example ADD4 is shown in Figure 5-3. Example ADD4 was synthesized with AutoLogic II to produce the *ripple carry* circuit shown in Figure 5-4.

```
entity add4 is
  port(A, B: in std_logic_vector(3 downto 0);
       D: out std_logic_vector(3 downto 0));
end add4;
architecture behave of add4 is
begin
  process(A,B)
  begin
       D <= A + B;
  end process;
end behave;
```

Figure 5-3 Behavioral description for example ADD4.



Figure 5-4 Synthesized circuit for example ADD4.

According to MIL-STD 883D, the synthesized circuit contains 102 unique gate level *SSL* faults. Fault simulations using the behavioral test vectors derived from Table 5-5 result in a fault coverage of 102/102 = 100%.

An alternate realization of the 4-bit *ripple carry* adder is presented in Figure 5-5. The NOR-only circuit, based on *POS* implementations of the *FA* and *HA* modules, contains



142 unique gate level *SSL* faults. Fault simulations using the eight behavioral test vectors again resulted in complete gate level fault coverage.

Figure 5-5 NOR-only realization of example ADD4.

5.1.1.5 Carry-in and Carry-out

The more complex form of behavioral addition includes a *carry-in* and/or a *carry-out*. Since the predefined VHDL *ADD* operator combines two *n*-bit operands to form an *n*-bit result, some additional manipulation is required to deal with the extra carries. The *carry-out* is produced by simply extending the *ADD* operation to n+1 bits and extracting the most significant bit of the result. The *carry-in* can be modeled by an extra addition of a single bit. Example ADD4wc, in Figure 5-6, demonstrates the behavioral description of a 4-bit addition with carries.

```
entity add4wc is
   port(A, B: in
                   std_logic_vector(3 downto 0);
           D: out std_logic_vector(3 downto 0);
                   std_logic;
         CIN: in
        COUT: out std_logic);
end add4wc;
architecture behave of add4wc is
begin
process(A, B, CIN)
  variable op1,op2,sum: std_logic_vector(4 downto 0);
  variable carry_in : std_logic_vector(1 downto 0);
  begin
      op1 := 0' \& A;
      op2 := `0' & B;
      carry_in := `0' & CIN;
      sum := op1 + op2 + carry_in;
      D \le sum(3 \text{ downto } 0);
      COUT <= sum(4);
   end process;
end behave;
```

Figure 5-6 Behavioral description for example ADD4wc

Rather than treating the two (+) operators separately in example ADD4wc, a synthesis tool will combine the operators to form a single adder. Recognition of this carry-in structure can be used to produce an appropriate behavioral fault model. Recall from the functional testing strategy in Section 5.1.1.1 that the least significant stage of the adder was only a *half-adder*. The only change when a carry-in is present is the conversion of this stage to a *full-adder*. Minor adjustments to the test vectors will ensure complete functional testing of this new module.

The test vectors for A and B remain the same for both Phase I and Phase III tests to the adder. The value of the *CIN* is simply set to match the other carry patterns for that test. For example, the last test for Phase I applies **Test 7** to each *FA*, thus CIN = 1. The test vectors for Phase II originally set the least significant bit of A or B to 1 to initiate the *ripple carries* through the adder. *CIN* now serves as the least significant input the adder and can assume that role. The resulting functional test vectors are presented in Table 5-6.

The behavioral fault model for the *add with carry* operation follows the same approach shown in Section 5.1.1.3. The functional test vectors are converted to error vectors which

						Test #			
Phase	Α	В	CIN	COUT	S	FA ₃	FA ₂	FA ₁	FA ₀
	0000	0000	0	0	0000	0	0	0	0
т	0000	1111	0	0	1111	1	1	1	1
	1111	0000	0	0	1111	2	2	2	2
	1111	1111	1	1	1111	7	7	7	7
TT	0000	1111	1	1	0000	5	5	5	5
	1111	0000	1	1	0000	6	6	6	6
тт	0101	0101	·0	0	1010	4	3	4	3
	1010	1010	1	1	0101	3	4	3	4

Table 5-6 Functional tests for example ADD4wc.

corrupt the resulting sum. Application of the eight behavioral test vectors derived from Table 5-6, to a synthesized gate level implementation of example ADD4wc, results in the expected complete gate level fault coverage.

5.1.2 Carry Look-Ahead Adder

Carry look-ahead (CLA) speeds up the process of combinational addition by determining carries for higher order stages of the adder without having to wait for them to ripple through lower order stages. From the truth table for the FA, Table 5-1, it can be seen that the carry-out is the same as the carry-in as long as one of the other inputs is a 1. Also, the carry-out is always a 1 independent of the carry-in when both of the other inputs are 1s, and a 0 if both are 0. Consequently, two useful functions can be defined: the carry-propagate, P_i , and the carry-generate, G_i [41].

$$P_{i} = A_{i} \oplus B_{i}$$
$$G_{i} = A_{i}B_{i}$$

The FA equations can then be written as:

 $S_i = P_i \oplus C_{i-1}$ $C_i = G_i + P_iC_{i-1}$

A CLA realization of a 3-bit adder is shown in Figure 5-7, where PG_i represents propagate-generate modules and SU_i implements the sum functions.



Figure 5-7 Carry look-ahead adder.

The carries for the various stages of a 4-bit adder are formed from the terms shown in Table 5-7. The columns indicate the contributions of the various stages, which are combined to produce the appropriate carry. Presence or absence of a *carry-in* (C_{-1}), to the least significant stage of the adder, determines whether or not to use the terms in the farthest right column.

		Stage						
Carry	3	2	1	0	-1			
C ₀				G ₀	P ₀ C ₋₁			
C ₁			G ₁	P ₁ G ₀	$P_1P_0C_{-1}$			
C ₂		G ₂	P_2G_1	$P_2P_1G_0$	$P_2P_1P_0C_{-1}$			
C ₃	G ₃	P ₃ G ₂	$P_3P_2G_1$	$P_3P_2P_1G_0$	$P_{3}P_{2}P_{1}P_{0}C_{-1}$			

Table 5-7 Carries for 4-bit CLA adder.

5.1.2.1 Functional Testing

Carry-propagate and carry-generate are both 2-input functions which will be completely tested by the functional test vectors developed for the ripple carry adder. The sum (S_i) is still a function of A_i , B_i , and C_{i-1} and, therefore, will also be exhaustively tested by the ripple carry test vectors. Examination of the carries from Table 5-7 indicates that any faults to lower order carries will *dominate* faults in higher order carries. Hence, testing for all the faults in the highest order carry in a *CLA* adder will provide coverage for all lower order faults.

A simple 4-bit addition will be used to evaluate the coverage of carry faults by the *rip*ple carry test vectors. Since this circuit has neither a carry-in nor a carry-out, the highest order carry is C_2 , which contains three terms: G_2 , P_2G_1 , and $P_2P_1G_0$. Possible functional faults include a missing carry, due to one of the terms producing an erroneous 0, and an extra carry, due to a term producing an unwanted 1.

Missing carry faults are evaluated in Table 5-8. In order to establish test vectors, G_i is set to 1 by $A_i = B_i = 1$ and P_i is set to 1 by $A_i = \overline{B}_i$. The right hand column indicates whether or not the indicated fault is covered by a *ripple carry* functional test vector.

Stage	Term	Test Set-up	Test Vector	Covered
0	$P_2P_1G_0$	$+\frac{X \ 0 \ 0 \ 1}{P_2 P_1 G_0}$	$+ \frac{0001}{1111} \\ 0000$	Yes
1	P ₂ G ₁	$+\frac{X 0 1 X}{X 1 1 X}$	$+ \frac{0010}{1111} \\ 0001$	No
2	G ₂	$+\frac{\begin{array}{c} x & 1 & x & x \\ + & x & 1 & x & x \\ \hline G_2 \end{array}$	$+ \frac{1111}{1110}$	Yes

Table 5-8 Missing carry faults.

The missing carry fault for stage 1 is not covered by the functional test vectors for the ripple carry adder. Examination of larger carry look-ahead circuits reveals that only the missing carry faults for the most and least significant stages will be covered. Hence, an additional n - 3 test vectors will be required to provide complete fault coverage.

Extra carry faults are evaluated in Table 5-9. An erroneous I in a term can be caused by any element in that term producing a I when it has been set to 0. Two *extra carry* faults are not covered by the functional test vectors for the 4-bit *ripple carry* adder. All of the erroneous generation faults (G_i) are covered as well as the erroneous propagation fault for the highest order *carry-propagate* (P_2).

Stage	Term	Test Set-up	Test Vector	Covered
	P ₂ P ₁ G ₀	$+\frac{X \ 0 \ 0 \ 0}{P_2 P_1 \overline{G}_0}$	$+ \frac{0000}{1111} \\ + \frac{1111}{1111}$	Yes
0		$+\frac{X \ 0 \ 0 \ 1}{P_2 \overline{P}_1 G_0}$	$+ \frac{0001}{1101} \\ + \frac{1101}{1110}$	No
		$+\frac{X \ 0 \ 0 \ 1}{\overline{P}_2 P_1 G_0}$	$+ \frac{0001}{1011} \\ + \frac{1011}{1100}$	No
1	PC	$+\frac{X 0 0 X}{X 1 1 X}$ $+\frac{X 1 1 X}{P_2\overline{G}_1}$	$+\frac{0000}{1111}\\1111$	Yes
· 1 · ·	P_2G_1	$+\frac{X 0 1 X}{\overline{P}_2 G_1}$	$+\frac{1010}{1010}$	Yes
2	G ₂	$+\frac{\begin{array}{ccc} x & 0 & x & x \\ + & x & 1 & x & x \\ \hline \overline{G}_2 \end{array}$	0000 + <u>1111</u> 1111	Yes

 Table 5-9 Extra carry faults.

The behavioral fault model for the *CLA* adder follows directly from the base *ripple carry* fault model. Additional functional test vectors for *missing carry* and *extra carry* faults are also mapped into error vectors which corrupt the resulting sum. The behavioral test vectors, therefore, consist of the eight *ripple carry* tests supplemented by some number of *CLA* tests. The additional behavioral test vectors for a 4-bit *CLA* adder are summarized in Table 5-10.

Fault	Stage	А	В	S
Missing Carry	1	0010	1111	0001
Extra Commu	0	0001	1101	1110
Extra Carry	U	0001	1011	1100

Table 5-10 Additional behavioral test vectors for CLA adder.

5.1.2.2 Application of the Behavioral Test Vectors

Behavioral test vectors are now applied to a *CLA* implementation of example ADD4. The NAND-only circuit in Figure 5-8 contains 130 unique *SSL* gate level faults. Fault simulation using the original eight *ripple carry* test vectors from Table 5-5 produces a gate level fault coverage of 127/130 = 97.69%. Examination of the results confirms that the three uncovered faults are, in fact, from signals forming the highest order carry, C_2 . Application of the additional *CLA* test vectors from Table 5-10 then achieves complete fault coverage.



Figure 5-8 CLA implementation of example ADD4.

5.1.2.3 Scalability

Gate level fault coverage can now be evaluated for a larger example, ADD8. Fault simulations using behavioral test vectors will be performed for both *ripple carry* and *CLA* implementations. Optimizations can also be investigated when detailed knowledge of the target technology is available.

Example ADD8 was synthesized to a gate level circuit producing the *ripple carry* implementation as shown in Figure 5-9. Fault simulations with eight behavioral test vec-

tors, extrapolated from example ADD4, produce a *SSL* gate level fault coverage of 234/234 = 100%. The fault coverage plot is shown in Figure 5-10.



Figure 5-9 Ripple carry implementation of example ADD8.



Figure 5-10 Fault coverage for ripple carry ADD8.

Next a *CLA* implementation of example ADD8 will be considered. Without specific knowledge of the details of the target technology, behavioral test vectors are first formed on the basis of a full 8-bit *CLA* structure.

The highest order carry in an 8-bit *CLA* adder, without a *carry-in* or *carry-out*, is C_6 . Extrapolating from Table 5-7, the terms for C_6 are $P_6P_5P_4P_3P_2P_1G_0$, $P_6P_5P_4P_3P_2G_1$, $P_6P_5P_4P_3G_2$, $P_6P_5P_4G_3$, $P_6P_5G_4$, P_6G_5 , and G_6 . According to the new behavioral fault model, test vectors for *extra carry* faults are required for all but the two most significant stages. Each stage has a separate fault for each *carry-propagate* (P_i) signal in that term. Behavioral test vectors for the *extra carry* faults are presented in Table 5-11. Test vectors are necessary for *missing carry* faults for all but the lowest and highest order terms. The *missing carry* behavioral test vectors are presented in Table 5-12.

Stage	Term	А	В	S
		00000001	11111101	11111110
		00000001	11111011	11111100
		00000001	11110111	11111000
	P ₆ P ₅ P ₄ P ₃ P ₂ P ₁ O ₀	00000001	11101111	11110000
		00000001	11011111	11100000
		00000001	10111111	11000000
		00000010	11111011	11111101
		00000010	11110111	11111001
1	$P_6P_5P_4P_3P_2G_1$	00000010	11101111	11110001
		00000010	11011111	11100001
		00000010	10111111	11000001
		00000100	11110111	11111011
		00000100	11101111	11110011
	P ₆ P ₅ P ₄ P ₃ O ₂	00000100	11011111	11100011
		00000100	10111111	11000011
		00001000	11101111	11110111
3	$P_6P_5P_4G_3$	00001000	11011111	11100111
		00001000	10111111	11000111
Δ	D D C	00010000	11011111	11101111
4	P ₆ P ₅ G ₄	00010000	10111111	11001111

Table 5-11 Behavioral test vectors for extra carry faults.

Stage	Term	А	В	S
1	$P_6P_5P_4P_3P_2G_1$	00000010	11111111	00000001
2	$P_6P_5P_4P_3G_2$	00000100	11111111	00000011
3	P ₆ P ₅ P ₄ G ₃	00001000	11111111	00000111
4	P ₆ P ₅ G ₄	00010000	11111111	00001111
5	P ₆ G ₅	00100000	11111111	00011111

 Table 5-12 Behavioral test vectors for missing carry faults.

A common modular *CLA* implementation was chosen, which cascades individual 4-bit *CLA* adders to form the *n*-bit addition. A block diagram of an 8-bit adder, therefore, has the structure shown in Figure 5-11.



Figure 5-11 Block diagram of modular CLA adder.

Fault simulations were performed on a NAND-only realization of the modular *CLA* adder using the combined behavioral test vectors. The original eight *ripple carry* test vectors produce a *SSL* gate level fault coverage of 290/310 = 93.55%. Application of the additional 25 *carry look-ahead* test vectors results in the expected complete gate level fault coverage.

Note the relative inefficiency of the *CLA* test vectors versus the original eight *ripple carry* vectors in the fault coverage plot in Figure 5-12. In addition, there is a large flat portion of the graph, between vectors 21 to 31, where fault coverage does not improve. This is due to the lack of specific knowledge about the modular *CLA* implementation. A designer equipped with details of the functional elements used in the target technology can optimize the behavioral test vectors.



Figure 5-12 Fault coverage for modular CLA adder.

5.1.2.4 Optimization of CLA Behavioral Faults

Given the additional detail that adders in a certain design are implemented by cascading 4-bit *CLA* modules, an optimized set of behavioral test vectors can be derived. From the block diagram of the modular *CLA* adder in Figure 5-11, highest order carries can be determined for both the upper, S_{7-4} , and lower, S_{3-0} , *CLA* modules.

The highest order carry in the lower CLA adder, with a carry-out and no carry-in, is C_3 . From Table 5-7, the terms for C_3 are $P_3P_2P_1G_0$, $P_3P_2G_1$, P_3G_2 , and G_3 . The highest order carry in the upper CLA adder, with a carry-in and no carry-out, is C_6 . Extrapolating from Table 5-7, the terms for C_6 are $P_6P_5P_4C_3$, $P_6P_5G_4$, P_6G_5 , and G_6 .

According to the new behavioral fault model, test vectors are necessary for missing carry faults for all but the lowest and highest order terms for each CLA module. The missing carry behavioral test vectors are, therefore, presented in Table 5-13. Test vectors for extra carry faults are required for all but the two most significant stages for each module. Each stage has a separate fault for each carry-propagate (P_i) . Behavioral test vectors for the extra carry faults are shown in Table 5-14.

Fault simulations were performed on the modular *CLA* adder using the optimized behavioral test vectors from Table 5-13 and Table 5-14. A *SSL* gate level fault coverage of

Stage	Term	А	В	S
1	$P_3P_2G_1$	00000010	111111111	00000001
2	P ₃ G ₂	00000100	11111111	00000011
4	P ₆ P ₅ G ₄	00010000	111111111	00001111
5	P ₆ G ₅	00100000	11111111	00011111

 Table 5-13 Optimized test vectors for missing carry faults.

Stage	Term	А	В	S
		00000001	11111101	11111110
0	$P_3P_2P_1G_0$	00000001	11111011	11111100
		00000001	11110111	11111000
1		00000010	11111011	11111101
1	P ₃ P ₂ O ₁	00000010	11110111	11111001
		00001000	11101111	11110111
3	$P_6P_5P_4C_3$	00001000	11011111	11100111
		00001000	10111111	11000111
	P P C	00010000	11011111	11101111
4	P ₆ P ₅ G ₄	00010000	10111111	11001111

Table 5-14 Optimized test vectors for extra carry faults.

310/310 = 100% was achieved with only 8 + 14 = 22 behavioral test vectors as opposed to the 8 + 25 = 33 non-optimized vectors. Hence, additional knowledge about the target architecture has allowed optimization of the behavioral test vector set while still achieving complete gate level fault coverage.

5.1.3 Summary

A behavioral fault model for addition has been derived using a complete functional testing technique. Both simple addition and the more complex form including a *carry-in* and/or a *carry-out* have been considered. The functional tests can be readily extended to *n*-bits, still requiring only eight behavioral test vectors. If the target hardware uses *carry look-ahead* circuits, additional behavioral faults are defined.

5.2 Subtraction

Subtraction is closely related to addition and all the techniques previously discussed in this chapter are applicable. Also, the subtraction operation is often implemented indirectly using adders and 2's complement arithmetic. A *ripple borrow* subtractor and a 2's complement addition will be used to demonstrate extension of functional testing to the subtraction operation.

5.2.1 Direct Subtraction

Subtraction, like addition, can be performed by the interconnection of functional modules. Consider the subtraction operation $D \le M - S$. The difference (D) is formed by the subtraction of the subtrahend (S) from the minuend (M). Like ripple carries in addition, full subtractors (FS) and half subtractors (HS) can be interconnected via borrows between stages.

The truth table for a *full subtractor* is presented in Table 5-15, where *BI* and *BO* represent *borrow-in* and *borrow-out* respectively [35]. Like the *half adder*, the *half subtractor* represents a subset of the *full subtractor*, where BI = 0.

Test #	BI	М	S	D	BO
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	0	0	1	1
5	1	0	1	0	1
6	1	1	0	0	0
7	1	1	1	1	1

Table 5-15 Truth table for full subtractor.

5.2.1.1 Functional Testing

The functional testing strategy presented for the *ripple carry* adder can now be applied to direct subtraction. Complete functional testing can be achieved by exhaustive testing of each subtractor module.

The functional test vectors proposed for direct subtraction are presented in Table 5-16. The Phase I tests represent complete testing of the *half subtractor* (HS_0) module. Phase II continues testing of the *full subtractor* (FS_i) modules where BI = 1. Finally, Phase III completes testing of the *full subtractor* modules by interleaving the remaining tests from Table 5-15.

					Tes	st #	
Phase	М	S	D	FS ₃	FS ₂	FS ₁	HS ₀
	0000	0000	0000	0	0	0	0
т	0000	1111	0001	5	5	5	1
L	1111	0000	1111	2	2	2	2
	1111	1111	0000	3	3	3	3
TT	0000	0001	1111	4	4	4	1
11	1110	1111	1111	7	7	7	1
ттт	0101	1010	1011	1	6	1	2
	1010	0101	0101	6	1	6	1

Table 5-16 Functional tests for 4-bit direct subtraction.

5.2.1.2 Application of the Behavioral Test Vectors

Again, the behavioral fault model follows directly from the error vector approach presented earlier in this chapter. Behavioral test vectors are derived from the error vectors formed by the functional tests from Table 5-16.

A 4-bit subtractor will now be used to evaluate the effectiveness of the behavioral test vectors. The VHDL behavioral description for example SUB4 is shown in Figure 5-13. The example subtracts B from A to produce a 4-bit *difference* D.

```
architecture behave of sub4 is
begin
  process(A,B)
  begin
      D <= A - B;
  end process;
end behave;
```

Figure 5-13 Behavioral description for example SUB4.

Example SUB4 was synthesized with AutoLogic II to produce the subtraction circuit shown in Figure 5-14. Fault simulations were then performed using the behavioral test vectors derived from Table 5-16. As expected, the behavioral test vectors achieved a *SSL* gate level fault coverage of 112/112 = 100%.



Figure 5-14 Synthesized circuit for example SUB4.

5.2.2 Subtraction Using Addition Circuitry

Subtraction can also be performed by taking the negative of the *subtrahend* and performing an addition [35][41]. With 2's complement arithmetic, the negation can be performed by taking the logical complement of B and adding 1. Thus, the subtraction operation becomes:

 $A - B = A + (-B) = A + \overline{B} + 1$

The behavioral test vectors developed for direct subtraction can now be evaluated for their performance on a subtractor realized with addition circuitry. A block diagram for a 4-bit subtractor implemented with *full adders* is presented in Figure 5-15.



Figure 5-15 Subtractor implemented with full adders.

Table 5-17 shows the functional tests applied to the *full adders* (FA_i) by the subtraction test vectors from Table 5-16. Since CIN = 1 for all vectors, **Test 4** through **Test 7** represents all possible tests for FA_0 . Hence, complete functional testing for each stage of the adder is achieved by the behavioral test vectors for subtraction. Additional fault simulation results can be found in Appendix B.

	Subtraction			Addition				Tes	st #		
Phase	М	S	D	A	В	CIN	S	FA ₃	FA ₂	FA ₁	FA ₀
	0000	0000	0000	0000	1111	1	0000	5	5	5	5
т	0000	1111	0001	0000	0000	1	0001	0	0	0	4
1	1111	0000	1111	1111	1111	1	1111	7	7	7	7
	1111	1111	0000	1111	0000	1	0000	6	6	6	6
TT	0000	0001	1111	0000	1110	1	1111	1	1	1	4
11	1110	1111	1111	1110	0000	1	1111	2	2	2	4
TTT	0101	1010	1011	0101	0101	1	1011	4	3	4	7
111	1010	0101	0101	1010	1010	1	0101	3	4	3	4

 Table 5-17 Functional tests for adder by subtraction test vectors.

5.2.3 Summary

A behavioral fault model for subtraction has been derived using a complete functional testing technique. Though based on direct subtraction, the behavioral test vectors also provide complete gate level fault coverage when implemented with adders. The functional tests can be readily extended to *n*-bits, still requiring only eight behavioral test vectors. Supplemental behavioral faults for a *CLA* implementation can be easily derived via the relationships of 2's complement arithmetic. Such an example can be found in Appendix A.

5.3 Constants as Operands

When one *operand* for an *arithmetic operator* is a *constant*, the behavior changes to that of a *unary operator* such as *increment* or *decrement*. Controllability is lost over the constant's input patterns, so the previously developed functional tests cannot be applied.

A 4-bit increment function, $Z \le Y + 1$, is used here as an example. Applying the constant 0001 to a *ripple carry* adder structure, the FA equations can be rewritten as:

$$S_0 = \overline{A}_0 \qquad S_i = A_i \oplus C_{i-1}$$

$$C_0 = A_0 \qquad C_i = A_i C_{i-1}$$

Stage 0 (ST₀) of the resulting incrementor is a single input function, while the remaining stages are 2-input functions that can be recognized as *half adders*. A functional testing strategy, like that in Section 5.1.1.1, can now be applied.

5.3.1 Functional Testing

Complete functional testing of the *unary operator increment* will now be achieved by exhaustive testing of every stage. The test vectors and resulting tests for each stage are presented in Table 5-18.

			Test #			
Phase	Y	Z	HA ₃	HA ₂	HA ₁	ST ₀
т	0000	0001	0	0	0	0
1	1111	0000	3	3	3	1
	1110	1111	2	2	2	0
ττ	1101	1110	2	2	1	1
п	1011	1100	2	1	3	1
	0111	1000	1	3	3	1

 Table 5-18 Functional tests for 4-bit increment function.

As can be seen from the patterns in the test results, complete functional testing of an nbit *increment* function can be achieved with n+2 test vectors. Similar analysis on a 4-bit *decrement* function, $Z \le Y - 1$, achieves similar results.

5.3.2 Generalized Behavioral Fault Model

By examining the functional test vector patterns for each *unary operator* function, (Y + 1, Y + 2,...) and (Y - 1, Y - 2,...), a generalized behavioral fault model can be developed. In the case of positive increments, the *n* test vectors for the Phase II tests can be derived by starting with a test pattern formed by the complement of the increment value.

For example, a 5-bit implementation of $Z \le Y + 3$ would start with a Phase II test vector of 11100. The starting test pattern is then rotated *n*-1 times to produce the remaining behavioral test vectors.

For negative numbers, the starting test pattern is based on the positive representation of the number. For example, a 6-bit implementation of $Z \le Y - 5$ would start with a Phase II test pattern of 000101. The functional tests produced by the resulting behavioral test vectors are summarized in Table 5-19.

				Test #				
Phase	Y	Z	ST ₅	ST ₄	ST ₃	ST ₂	ST ₁	ST ₀
т	000000	111011	0	0	0	0	0	0
I	111111	111010	3	3	3	3	3	1
	000101	000000	1	1	1	3	1	1
	001010	000101	1	1	2	1	2	0
тт	010100	001111	1	2	0	2	0	0
11	101000	100011	3	1	2	0	0	0
	010001	001100	1	2	0	1	1	1
	100010	011101	2	0	0	1	2	1

Table 5-19 Functional tests for 6-bit function Z <= Y - 5.

Complete functional testing, of the unary operators formed by arithmetic operators with a constant operand, can be achieved with n+2 behavioral test vectors. The resulting gate level fault coverage, however, depends on optimizations performed by a synthesis tool which may affect the underlying *ripple carry* structure. Further application of these new behavioral fault models can be found in the examples in Appendix A.

5.4 Comparison with Previous Fault Models

As was the case with *relational operators*, few previous fault models address *arithmetic operators*. As part of the development of the "B-algorithm: A Behavioral Test Generation Algorithm," [21][22] Cho and Armstrong developed a new behavioral fault model which included *arithmetic operators* as part of *micro-operation* faults. Such faults perturb an *arithmetic operator* to another operator as indicated in Table 5-20.

Fault-free Operator	Faulty Operator
ADD	SUB, XOR
SUB	ADD, XOR

Table 5-20	Micro-o	peration	Faults
-------------------	---------	----------	--------

Recall that this fault model suffered from what the authors called the "big micro-operation problem," where only a single test vector is generated for a complex block of logic. To raise their equivalent gate level fault coverage to acceptable levels, an additional 4n-1test vectors are generated by a heuristic test generator for each *n*-bit micro-operation.

A similar fault model was proposed by Al Hayek and Robach [4] as part of a mutationbased testing strategy in which VHDL behavioral faults are considered as software faults. For Arithmetic Operator Replacement (AOR), ADD(+) is replaced by SUB(-). Mutation analysis does not take into account the size of the hardware implementation, because it considers each *arithmetic operator* as a software operation and consequently generates only one test vector. In order to improve the performance of their technique, the authors also resorted to heuristics to generate additional test vectors for their complex operators.

The new behavioral fault models developed for *arithmetic operators* eliminate the need to supplement test vector sets. While the new fault models are definitely more complex than previous ones, this is because they more accurately reflect the underlying complexity of the hardware faults which they attempt to model.

5.5 Conclusions

New behavioral fault models have been developed for the predefined VHDL arithmetic operators ADD(+) and SUB(-). The fault models are based on complete functional testing of arithmetic building blocks. Though optimizations may modify the building blocks, the basic function of the arithmetic operators remains unchanged. By concentrating on functional testing, complete gate level SSL fault coverage can be obtained over a broad range of hardware implementations.

The base fault model is derived from the *ripple carry* connection of *full* and *half adders*. Because such realizations are made up of 2- and 3-input functional building blocks, complete functional testing can be achieved by exhaustive testing of each module.

Only eight behavioral test vectors are required for complete gate level fault coverage, regardless of the size of the *operands*.

If the target hardware uses *carry look-ahead* circuits, additional behavioral faults are defined. Testing for all the faults in the highest order carry in a *CLA* adder will provide coverage for all lower order faults. Test vectors are added for *missing carry* faults and *extra carry* faults for specific stages. Knowledge of the target architecture will allow optimization of the test vector set while still achieving complete gate level fault coverage.

Subtraction is closely related to addition and all the techniques previously discussed are applicable. The functional tests can be readily extended to *n*-bits, still requiring only eight behavioral test vectors. Though based on direct subtraction, they also provide complete functional testing when the operation is realized indirectly with adders. Additional behavioral faults for a *CLA* implementation can be easily derived via the relationships of 2's complement arithmetic.

The new behavioral fault models developed for *arithmetic operators* eliminate the need to supplement test vector sets via methods such as heuristics. While the new fault models are more complex than previous ones, this is because they more accurately reflect the underlying complexity of the hardware faults which they attempt to model.

Chapter 6 Other Operators

The remaining VHDL operators from Table 1-2 include *logical*, *unary*, *multiplying*, and *miscellaneous*. *Logical operators* provide a close link between behavioral and gate level descriptions. Mapping SSL gate level faults into the behavioral domain is, therefore, a fairly straight forward process. However, differences in actual gate level structures, due to optimization and synthesis tools, must also be taken into account.

In contrast to previously discussed *binary operators*, *unary operators* affect only a single *operand*. This distinction does not alter the analysis of gate level faults and their mapping to behavioral faults. Though classified in the *miscellaneous* category, the *operator ABS* will be considered with this group.

Due to power of 2 restrictions placed on *multiplying operators*, detailed in Figure 1-1, implementation becomes simply a shifting of lines, rather than any additional hardware. Since no more gates are implied by such *operations*, no additional gate level faults are introduced. The same synthesis guidelines apply to the miscellaneous operator (**), hence, no behavioral faults will be defined.

6.1 Logical Operators

The predefined VHDL *logical operators* include AND, OR, NAND, NOR, and XOR. All these operators are binary, therefore only 2-input gate level structures are implied. The miscellaneous operator NOT is a unary operator that does not introduce additional SSL gate level faults, not covered by other operators. Further, the logical pairs AND/OR and NAND/NOR differ by only a single inversion, hence detailed analysis of one group will provide the insight necessary to develop the behavioral fault models for the entire set of operators.

The remaining *logical operator, XOR*, will be examined separately due to the unique nature of its functional faults. Additionally, previous research on *XOR* structures, such as parity trees, can provide optimizations for the behavioral tests necessary to provide complete gate level fault coverage.

6.1.1 AND/OR

The *logical operators AND/OR* provide a close link between behavioral and gate level descriptions. Mapping *SSL* gate level faults into the behavioral domain is, therefore, a fairly straight forward process. However, differences in actual gate level structures, due to optimization and synthesis tools, must also be taken into account.

6.1.1.1 Functional Faults

The behavioral description of a 2-operand AND operation can be expressed as:

 $Z \ll A AND B;$

A direct gate level implementation results in a 2-input AND gate which can be analyzed for SSL faults. A reduced set of functional faults for the AND operation is presented in Table 6-1. The three test vectors (AB) necessary to detect all functional faults are, therefore, 01, 10, and 11.

A	В	Z	$\mathbf{Z} = 0$	$\mathbf{Z} = \mathbf{A}$	Z = B
0	0	0			
0	1	0			1
1	0	0		1	
1	1	1	0		

 Table 6-1 Functional faults for AND operation.

A similar analysis of the *OR operation* produces the reduced set of functional faults shown in Table 6-2. The required functional test vectors are 00, 01, and 10.

A	В	Z	Z = 1	Z = A	Z = B
0	0	0	1		
0	1	1		0	
1	0	1			0
1	1	1			

Table 6-2 Functional faults for OR operation.

The functional faults for the NAND and NOR operators follow directly from the above analysis. The functional tests for the NAND operation are the same as for the AND operation, while the tests for the NOR operation are the same as the OR.

6.1.1.2 Complex Expressions

The functional faults for a single *logical operator* have been determined quite easily. However, interactions among these *operators* in more complex *expressions* must also be addressed. Since all the *logical operators* are binary, these interactions can be investigated with the use of a binary tree.

Consider the behavioral description of a logical *expression* presented in Figure 6-1.

```
entity SOP1 is
    port( A, B, C, D: in std_logic;
        Z: out std_logic );
end SOP1;
architecture behave of SOP1 is
begin
    process(A,B,C,D)
    begin
        Z <= (A AND B) OR (C AND D);
end process;
end behave;
```

Figure 6-1 Behavioral description for example SOP1.

The *expression* on the right hand side of the assignment statement can be parsed into a binary tree shown in Figure 6-2. The nodes (1,2,3) are formed by the *logical operators*, while the leaves of the tree are the signals A, B, C, and D.



Figure 6-2 Binary tree representing example SOP1.

According to the previous analysis, three functional faults affect each of the three *logical operators* in the *expression*. For example, the first functional fault to the AND operator at node 1 requires a test vector (AB) of 01 and produces an erroneous output of A AND B = 1. In order for this erroneous output to be observable at the *expression* output Z, appropriate values for the other signals (C,D) must be determined. This can be accomplished using the following set of Boolean identities:

According to the first identity, setting the right hand *operand* of the *OR operator* at node 2 to 0 will allow the left hand *operand* to propagate up the tree unchanged. Using the last identity, setting either *operand* of the *AND operator*, at node 3, to 0 will produce the desired input to node 2. Applying this set of identities along with the functional faults for the *AND* and *OR operators* produces the test vectors shown in Table 6-3.

Node	Functional Test	Test Requirements	Test Vector (ABCD)
	01	A = 0, B = 1, (C AND D) = 0	010X
1	10	A = 1, B = 0, (C AND D) = 0	100X
	11	A = 1, B = 1, (C AND D) = 0	110X
	00	(A AND B) = 0, (C AND D) = 0	0X0X
2	01	(A AND B) = 0, (C AND D) = 1	0X11
	10	(A AND B) = 1, (C AND D) = 0	110X
	01	C = 0, D = 1, (A AND B) = 0	0X01
3	10	C = 1, D = 0, (A AND B) = 0	0X10
	11	C = 1, D = 1, (A AND B) = 0	0X11

 Table 6-3 Functional test vectors for example SOP1.

The set of functional test vectors can be reduced by combination of the don't care values. The final test vectors for example SOP1 are presented in Table 6-4. Also listed are the functional tests covered for each node of the parse tree.

Test Vector (ABCD)	Functional Tests (Node [Test])
0101	1 [01], 2 [00], 3 [01]
0X10	3 [10]
0X11	2 [01], 3 [11]
100X	1[10]
110X	1 [11], 2 [10]

 Table 6-4 Reduced test vectors for example SOP1.

6.1.1.3 Scalability

Example SOP4 was created by expanding the width of the signals A, B, C, and D to std_logic_vector(3 downto 0). It has been previously shown that expansion of the data path simply causes replication of the single bit case implemented in parallel. Hence, extra test vectors are not required since the additional hardware can be tested at the same time. The expanded test vectors for example SOP4 are shown in WAVES format in Figure 6-3.

8 A	В	С	D	\mathbf{Z}	:	time	∋;
0000	1111	0000	1111	0000	:	500	ns;
0000	XXXX	1111	0000	0000	:	500	ns;
0000	XXXX	1111	1111	1111	:	500	ns;
1111	0000	0000	XXXX	0000	:	500	ns;
1111	1111	0000	XXXX	1111	:	500	ns;

Figure 6-3 WAVES test vectors for example SOP4.

6.1.1.4 Behavioral Fault Model

As was the case with the *arithmetic operators*, an error vector approach is taken for abstracting the *logical operator* functional faults into the behavioral domain. The *XOR operator* provides the desired corruption properties by inverting the appropriate bit(s) when presented with a non-zero error vector.

Recall from Table 6-3, the first functional fault to the AND operator at node 1 required that A = 0, B = 1, and (C AND D) = 0. This fault criteria can be directly translated to an error vector for example SOP1 as:

 $Z \le (A AND B) OR (C AND D) XOR (A = `0' AND B = `1' AND (C AND D) = `0')$

Again, as was the case for the *arithmetic operators*, this expression is not syntactically correct due to VHDL type differences. A complete implementation of the behavioral fault model for example SOP1 is presented in Figure 6-4.

```
Z <= (A AND B) OR (C AND D) XOR to_stdlogic(
(A = `0' AND B = `1' AND (C AND D) = `0') OR
(A = `1' AND B = `0' AND (C AND D) = `0') OR
(A = `1' AND B = `1' AND (C AND D) = `0') OR
((A AND B) = `0' AND (C AND D) = `0') OR
((A AND B) = `0' AND (C AND D) = `0') OR
((A AND B) = `0' AND (C AND D) = `1') OR
((A AND B) = `1' AND (C AND D) = `0') OR
(C = `0' AND D = `1' AND (A AND B) = `0') OR
(C = `1' AND D = `0' AND (A AND B) = `0') OR
```

Figure 6-4 Behavioral fault model for example SOP1.

6.1.1.5 Application of the New Fault Models

A simple example is now presented to demonstrate the application of the new fault models developed for *logical operators*. A test vector set will be formed based on behavioral faults and then applied to synthesized gate level realizations.

Example GT in Figure 6-5 uses *logical operators* to describe a Boolean *expression* for the 2-bit *greater than* function, examined in detail in Chapter 4. Example GT presents two minor differences from example SOP1. First, the inclusion of the *unary operator NOT* means that the resulting parse tree will not be completely binary. This should have no

Figure 6-5 Behavioral description for example GT.

effect on the test generation process since functional faults are only defined for the binary *logical operators*. Second, not all of the pairings are explicitly defined for the binary *logical operators* in example GT. Hence, the precise implementation by the synthesis tool cannot be determined. Since the new fault models are based on a functional analysis, they should provide complete gate level fault coverage over a broad range of realizations.

A parse tree for example GT is presented in Figure 6-6. As before, the binary nodes are formed by the *logical operators*, while the leaves of the tree represent the signals.



Figure 6-6 Parse tree for example GT.

Application of the new behavioral fault models implies three behavioral faults for each of the five binary *logical operators*. The resulting test vectors are presented in Table 6-5. Again, the set of behavioral test vectors can be reduced by combination of the don't care values. The final test vectors for example GT are presented in Table 6-6. It is worth noting that the behavioral test vectors derived here are consistent with the functional faults for the 2-bit GT function presented in Figure 4-2.

Example GT was synthesized with AutoLogic II to produce the gate level Structure1 shown in Figure 6-7. Note that the groupings for the *AND* gates do not match the parse tree in Figure 6-6. According to MIL-STD 883D, Structure1 contains 30 unique gate level *SSL* faults. Fault simulations using the behavioral test vectors from Table 6-6 resulted in complete gate level fault coverage.

Node	Behavioral Test	Test Requirement	Test Vector $(A_1A_0B_1B_0)$
	01	$A_1 = 0, \text{ not } B_1 = 1,$ $A_0 \text{ AND not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 0$	000x
1	10	$A_1 = 1, \text{ not } B_1 = 0,$ $A_0 \text{ AND not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 0$	101X
	11	$A_1 = 1, \text{ not } B_1 = 1,$ $A_0 \text{ AND not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 0$	100X
	00	$A_1 \text{ AND not } B_1 = 0,$ $A_0 \text{ AND not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 0$	00XX
2	01	$A_1 \text{ AND not } B_1 = 0,$ $A_0 \text{ AND not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 1$	0100
	10	$A_1 \text{ AND not } B_1 = 1,$ $A_0 \text{ AND not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 0$	100X
	01	$A_0 = 0, \text{ not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 1,$ $A_1 \text{ AND not } B_1 = 0$	1010
3	10 $A_0 = 1, \text{ not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 0, \\A_1 \text{ AND not } B_1 = 0$		01X1
	11	$A_0 = 1, \text{ not } B_0 \text{ AND } (A_1 \text{ OR not } B_1) = 1,$ $A_1 \text{ AND not } B_1 = 0$	1110
	01	not $B_0 = 0$, A_1 OR not $B_1 = 1$, $A_0 = 1$, A_1 AND not $B_1 = 0$	1111
4	10	not $B_0 = 1$, A_1 OR not $B_1 = 0$, $A_0 = 1$, A_1 AND not $B_1 = 0$	0110
	11	not $B_0 = 1$, A_1 OR not $B_1 = 1$, $A_0 = 1$, A_1 AND not $B_1 = 0$	1110
	00	$A_1 = 0$, not $B_1 = 0$, not $B_0 = 1$, $A_0 = 1$, A_1 AND not $B_1 = 0$	0110
5	01	$A_1 = 0$, not $B_1 = 1$, not $B_0 = 1$, $A_0 = 1$, A_1 AND not $B_1 = 0$	0100
	10	$A_1 = 1$, not $B_1 = 0$, not $B_0 = 1$, $A_0 = 1$, A_1 AND not $B_1 = 0$	1110

Table 6-5	Behavioral te	st vectors f	or examp	le (ЪТ.
------------------	---------------	--------------	----------	------	-----

-

Test Vector $(A_1A_0B_1B_0)$	Functional Tests (Node [Test])
000x	1 [01], 2 [00]
0100	2 [01], 5 [01]
01X1	3 [10]
0110	4 [10], 5 [00]
100X	1 [11], 2 [10]
1010	1 [10], 3 [01]
1110	3 [11], 4 [11], 5 [10]
1111	4[01]

Table 6-6 Reduced test vectors for example GT.



Figure 6-7 Synthesized Structure1 for example GT.

Example GT was next synthesized and optimized for an alternate target technology. The resulting Structure2 is shown in Figure 6-8. Fault simulations using the same behavioral test vectors achieved a *SSL* gate level fault coverage of 35/35 = 100%. The fault coverage graph is shown in Figure 6-9.



Figure 6-8 Synthesized Structure2 for example GT.



Figure 6-9 Fault coverage for Structure2 of example GT.

6.1.2 XOR

The remaining *logical operator, XOR*, is examined here separately, due to the unique nature of its functional faults. Additionally, previous research on *XOR* structures, such as parity trees [16][53], can provide optimizations for the behavioral tests necessary to provide complete gate level fault coverage.

6.1.2.1 Functional Faults

The XOR gate has numerous logical implementations producing several different sets of functional faults. In order to test an XOR gate whose internal structure is unknown, an exhaustive test set, four patterns, is needed to detect all SSL faults [30]. A generalized set of functional faults is presented in Table 6-7.

The set of generalized functional faults and the following Boolean identities allow functional tests to be generated for the *XOR operator* used in complex *expressions* with the other *logical operators*.

Y XOR
$$0 = Y$$

Y XOR $Y = 0$
Y XOR $\overline{Y} = 1$

Though not optimal, the functional tests ensure complete SSL fault coverage of the XOR gates regardless of their internal structure.

Α	B	Ζ	Z = A NAND B	$Z = A AND \overline{B}$	$Z = \overline{A} AND B$	Z = A OR B	
0	0	0	1				
0	1	1		0			
1	0	1			0		
1	1	0				1	

Table 6-7 Generalized functional faults for XOR operation.

6.1.2.2 Optimized Test Generation

Expressions containing only *XOR operators* are common for circuits such as parity networks. Previous research has demonstrated the generation of optimum test patterns for such parity networks with fixed structures [16][53]. However, when working with behavioral descriptions, the actual gate level structure is often unknown and may ultimately be determined by a synthesis tool. Still, modifications to the algorithms are possible to allow for complete gate level fault coverage over a broad range of implementations.

The Bossen algorithm [16] applies an exhaustive test to each *XOR* gate by using the labeling scheme shown in Figure 6-10. The test sequences are labeled R, S, and T. Each of the sequences is the modulo-2 sum of the other two. That is, $T = R \oplus S$, $S = T \oplus R$, and $R = S \oplus T$.



Figure 6-10 Labeling scheme for Bossen test.

Test generation using the Bossen algorithm will be demonstrated via a simple example. The behavioral description for XOR5 is shown below:

Z <= A XOR B XOR C XOR D XOR E

Grouping the terms from left to right produces a linear tree or cascade implementation shown in Figure 6-11.



Figure 6-11 Structure Cascade1 for example XOR5.

The test set is developed by first assigning one of the three sequences to the output of the tree. Other sequences are then determined moving from right to left, ensuring that each XOR gate is exhaustively tested. The resulting test vectors are shown in Table 6-8. Note that the labeling sequences are not unique and the test vectors generated may not completely test other implementations of the same expression.

Signal	Label	Sequence
Α	Т	0101
В	S	0011
C	Т	0101
D	R	0110
E	S	0011

Test Vector (ABCDE)	Z
00000	0
10110	1
01011	1
11101	0

Table 6-8 Bossen test vectors for Cascade1.

If the behavioral description for XOR5 is instead implemented by grouping terms from right to left, a second cascade structure is formed. Figure 6-12 shows that the labeling scheme from Figure 6-11 cannot be applied to this alternate implementation.



Figure 6-12 Structure Cascade2 for example XOR5.

In fact, no labeling will provide exhaustive testing for all XOR gates in both circuits Cascade1 and Cascade2. What is needed is a modification to the Bossen algorithm that will take into account the most likely implementations of an XOR-only expression. Consider the following groupings for example XOR5:

Z <= (((A XOR B) XOR C) XOR D) XOR E	(Cascade1)
Z <= ((A XOR B) XOR (C XOR D)) XOR E	(Balanced1)
$Z \iff A XOR (B XOR (C XOR (D XOR E)))$	(Cascade2)
Z <= A XOR ((B XOR C) XOR (D XOR E))	(Balanced2)

The first and third groupings produce structures Cascade1 and Cascade2 respectively, while the second and forth groupings produce balanced trees [53]. If we consider these structures to represent a broad range of possible implementations of the behavioral description, a generalized Bossen algorithm can then provide complete gate level fault coverage.

An extra restriction is added to the Bossen algorithm to account for multiple possible structures: According to the assumed groupings for a cascade and its corresponding balanced implementation (Cascade i, Balanced i), no two inputs that are grouped together can be assigned the same test sequence.

Applying this modified algorithm to structure Cascade2 produces the labeling sequences shown in Figure 6-13 and the test vectors shown in Table 6-9. Since the test vectors developed for structure Cascade1 already meet the additional restriction for the modified Bossen algorithm, the generalized set of test vectors from Table 6-8 and Table 6-9 will now provide exhaustive testing for all four structures of XOR5.



Figure 6-13 Modified Bossen test for Cascade2.
Signal	Label	Sequence
А	S	0011
В	R	0110
С	Т	0101
D	S	0011
Е	Т	0101

Test Vector (ABCDE)	Z
00000	0
01101	1
11010	1
10111	0

 Table 6-9 Modified Bossen test vectors for Cascade2.

6.1.2.3 Evaluation of the Generalized Test Vectors

The generalized Bossen test vectors for example XOR5 are presented in WAVES format in Figure 6-14. Fault simulations were conducted on multiple implementations including Structure4 in Figure 6-15. Complete *SSL* gate level fault coverage was achieved for each realization. An example fault coverage graph for Structure4 is shown in Figure 6-16. Note that though the behavioral test vectors are optimized compared to those generated by a parse tree, they are still generalized to apply to multiple gate level structures.

```
% ABCDE Z : time ;
00000 0 : 500 ns;
01011 1 : 500 ns;
01101 1 : 500 ns;
10110 1 : 500 ns;
10111 0 : 500 ns;
11010 1 : 500 ns;
11101 0 : 500 ns;
```

Figure 6-14 Generalized Bossen test vectors for example XOR5.



Figure 6-15 Structure4 for example XOR5.



Figure 6-16 Fault coverage for Structure4 of example XOR5.

6.1.3 Comparison with Previous Fault Models

Behavioral fault models for *logical operators* are addressed by Armstrong et al., where early models replaced one micro-operation with any other in its class [9][69], while later studies tried to determine which perturbations produced the best fault coverage [19]. Finally, the B-algorithm eliminated micro-operation faults for *logical operators* by defining bit-wise *stuck-at* faults for any one of its arguments (a *signal* or an unnamed *signal* for an *expression*) [21][22]. This method amounts to exhaustive testing of each *logical operator* in an *expression*.

In their mutation based testing strategy, Al Hayek and Robach [4] define Logical Operator Replacement (*LOR*) in which each *logical operator* is replaced by each of the other operators. This method treats the VHDL description as software and has little relation to actual hardware faults. Finally, other fault models [18][27][60] completely neglect *logical operators* and instead rely on *stuck-signals* to provide fault coverage.

The new behavioral fault models developed for *logical operators* are based on functional faults that require less than exhaustive testing for all operators except XOR. For the special case of XOR-only *expressions*, a generalized Bossen algorithm is presented that allows for optimization of test sequences. The new fault models and algorithms thus provide complete SSL gate level fault coverage for a broad range of implementations.

123

6.2 Unary Operators

The unary operator for negation (-) performs a function related to the miscellaneous operator ABS; each affects the sign of a 2's complement number. By analyzing each operator's functional faults in tandem with the other, a consistent behavioral fault model can be developed. Note that no previous behavioral fault models even address these operators.

6.2.1 Absolute Value

The *ABS operator* computes the absolute value of a 2's complement number. The operator's functional faults will be investigated for the 4-bit case, then generalized to n-bits. The following Boolean equations describe the 4-bit absolute value function:

 $Z_{0} = X_{0}$ $Z_{1} = X_{1} \oplus (X_{3}X_{0})$ $Z_{2} = X_{2} \oplus (X_{3}X_{1} + X_{3}X_{0})$ $Z_{3} = 0$

Analysis of the faulty behavior of Z = ABS X produces the reduced set of functional faults shown in Table 6-10. The faults are generically labeled F1-F5 and are shown with the integer value of the appropriate test vectors. Before proceeding further with the development of a behavioral fault model, the *negation operator* will first be examined for common functional faults.

Functional Fault	Test Vectors (integer)
F1	1, 3, 5, 7
F2	2, 3, 6, 7
F3	-3, -7
F4	-2, -6
F5	-4

Table 6-10 Reduced functional faults for 4-bit ABS.

6.2.2 Negation

The *negation operator* (-) changes the sign of a 2's complement number. The following Boolean equations describe the 4-bit negation function:

$$Z_{0} = X_{0}$$

$$Z_{1} = X_{1} \oplus X_{0}$$

$$Z_{2} = X_{2} \oplus (X_{1} + X_{0})$$

$$Z_{3} = X_{3} \oplus (X_{2} + X_{1} + X_{0})$$

Analysis of the faulty behavior of Z = -X produces the reduced set of functional faults shown in Table 6-11. The faults are generically labeled F6-F10 and are shown with the integer value of the appropriate test vectors.

Functional Fault	Test Vectors (integer)		
F6	0		
F7	-3, -7, 1, 5		
F8	-5, -6, -7, 1, 2, 3		
F9	-2, -6, 2, 6		
F10	-4, 4		

 Table 6-11 Reduced functional faults for 4-bit negation.

6.2.3 Generalized Functional Faults

Examination of the reduced set of functional faults for the *absolute value* and *negation operators* provides the necessary insight for developing a common fault model. A generalized set of functional faults is shown in Table 6-12. The faults are covered by three tests spanning the range of integer values (0, -7, 7) combined with a readily identifiable pattern from the 4-bit test vectors. These patterns are easily replicated for the n-bit case.

Functional Faults	Test Vector (integer)	Test Vector $(X_3X_2X_1X_0)$
F6	0	0000
F1, F2	7	0111
F3, F7, F8	-7	1001
F4, F9	-6	1010
F5, F10	-4	1100

 Table 6-12 Generalized functional faults for absolute value and negation.

6.2.4 Behavioral Fault Model

Since *ABS* and (-) only operate on integers, the form of their behavioral fault models will be slightly different. One method would be to convert the resulting integer to a bit_vector, which could then be corrupted using an error vector and the *XOR* operator. The corrupted bit_vector would then have to be converted back to an integer to match the type of the original operation.

Since the purpose of the error vector approach is to simply corrupt the result of the operation, another operator that works directly with integers could just as easily be used. Hence, the *addition operator* (+) is used here instead of the *XOR* to reduce the number of type conversions necessary. An implementation of the behavioral fault model for the *ABS operator* is presented in Figure 6-17. The *negation operator* (-) can also be corrupted using the same method.

Z <= (ABS X) + to_integer(X = 0 OR X = 7 OR X = -7 OR X = -6 OR X = -4);

Figure 6-17 Behavioral fault model for ABS.

6.2.5 Evaluation of Behavioral Test Vectors

The generalized functional faults from Section 6.2.3 can be readily extrapolated for a larger range of integer values. For examples ABS8 and NEG8, X is declared as an integer with range from -127 to +127. Thus, a synthesis tool will generate hardware with eight bits to represent the 2's complement value of X. The WAVES test vectors for example ABS8 are shown in Figure 6-18.

% X	Z	:	time ;
00000000	0000000	:	500 ns;
01111111	1111111	:	500 ns;
10000001	1111111	:	500 ns;
10000010	1111110	:	500 ns;
10000100	1111100	:	500 ns;
10001000	1111000	:	500 ns;
10010000	1110000	:	500 ns;
10100000	1100000	:	500 ns;
11000000	1000000	:	500 ns;



Example ABS8 was synthesized to a gate level implementation using AutoLogic II. Structure1 is shown in Figure 6-19. Fault simulations using the behavioral test vectors produced a *SSL* gate level fault coverage of 132/132 = 100%. The fault coverage graph is shown in Figure 6-20.



Figure 6-19 Synthesized Structure1 of example ABS8.



Figure 6-20 Fault coverage for Structure1 of example ABS8.

Due to the common behavioral fault model, the test vectors for example NEG8 are the same as those for ABS8. Example NEG8 was synthesized to the gate level circuit shown in Figure 6-21. According to MIL-STD 883D, the synthesized circuit contains 114 unique

gate level SSL faults. Fault simulations using the common behavioral test vectors also result in complete gate level fault coverage.



Figure 6-21 Synthesized circuit for example NEG8.

6.3 Conclusions

New behavioral fault models have been developed for the remaining predefined VHDL *operators*. The fault models are based on a functional analysis of the *logical, unary, multiplying,* and *miscellaneous operators*. Though not necessarily optimal, the new fault models provide complete gate level *SSL* fault coverage over a broad range of hardware implementations.

The new behavioral fault models developed for the *logical operators (AND, OR, NAND, NOR, XOR)* are based on functional faults that require less than exhaustive testing for all operators except *XOR*. For the special case of *XOR*-only *expressions*, a generalized Bossen algorithm is presented that allows for optimization of test sequences.

The unary operator for negation (-) performs a function related to the miscellaneous operator ABS; each affects the sign of a 2's complement number. By analyzing each operator's functional faults in tandem with the other, a consistent behavioral fault model was developed. Note that no previous behavioral fault models even address these operators.

Chapter 7 Other Programming Constructs

VHDL also includes other constructs drawn from familiar programming languages. Program *loops*, *functions*, and *procedures* are used in VHDL behavioral descriptions for design simplicity and reuse/repetition of functional blocks. Since any description using these constructs can be rewritten equivalently without them, no additional behavioral faults are implied. Several design examples will be used to demonstrate the interaction between previously defined behavioral faults and these other programming constructs.

7.1 Loops

The VHDL subset, detailed in Appendix D [36], restricts the use of the *loop* statement to only the *for* iteration scheme. The bounds of the discrete range of the *loop* must be specified directly or indirectly as static values belonging to an integer type. Hence, the program *loop* can be expanded or "unrolled" to an equivalent form eliminating the *loop* construct.

7.1.1 A Simple Example

Example SHIFT4u in Figure 7-1 demonstrates the use of program *loops* to perform shifting operations. With control signal OP = "01", a right shift of the unsigned signal A is performed by the I in 0 to 2 *loop*. Similarly, with OP = "10", a left shift is accomplished via the I in 3 downto 1 *loop*. For this example, other values for the control signal *OP* pass signal A unchanged.

```
entity shift4u is
    port(
        OP: in std_logic_vector(1 downto 0);
        A: in std_logic_vector(3 downto 0);
        D: out std_logic_vector(3 downto 0)
    );
end shift4u;
```

Figure 7-1 Behavioral description for example SHIFT4u.

```
architecture behave of shift4u is
begin
  process(OP,A)
    variable TMP: std_logic_vector(3 downto 0);
  begin
    case OP is
      when "01" =>
        for I in 0 to 2 loop
          TMP(I) := A(I+1);
        end loop;
        TMP(3) := `0';
      when "10" =>
        for I in 3 downto 1 loop
          TMP(I) := A(I-1);
        end loop;
        TMP(0) := `0';
      when others =>
        TMP := A;
    end case;
    D \leq TMP;
  end process;
end behave;
```

Figure 7-1 Behavioral description for example SHIFT4u.

Due to the directly specified discrete range in each *loop*, they can be readily expanded to a sequence of statements eliminating the *loop* constructs. An expanded version of the *case* statement for example SHIFT4u is shown in Figure 7-2.

```
case OP is
when "01" =>
TMP(0) := A(1);
TMP(1) := A(2);
TMP(2) := A(3);
TMP(3) := '0';
when "10" =>
TMP(3) := A(2);
TMP(2) := A(1);
TMP(1) := A(0);
TMP(0) := '0';
when others =>
TMP := A;
end case;
```



The only behavioral faults affecting the expanded example SHIFT4u are the control faults for the *case* statement. The behavioral faults and resulting test vectors are shown in Table 7-1. Combining don't cares produces the behavioral test vectors in Figure 7-3.

Behavioral Fault	Corrupted Clause	Test Vectors (OP A)	
WHEN-00-CORRUPT (by WHEN-01)(OR)	TMP(0) := $A(0)$ OR $A(1)$ TMP(1) := $A(1)$ OR $A(2)$ TMP(2) := $A(2)$ OR $A(3)$ TMP(3) := $A(3)$ OR '0'	00 1010 00 x10x	
WHEN-00-CORRUPT (by WHEN-01)(AND)	$\begin{array}{llllllllllllllllllllllllllllllllllll$	00 0101 00 101X	
WHEN-00-CORRUPT (by WHEN-10)(OR)	TMP(0) := A(0) OR '0' TMP(1) := A(1) OR A(0) TMP(2) := A(2) OR A(1) TMP(3) := A(3) OR A(2)	00 0101 00 x01x	
WHEN-00-CORRUPT (by WHEN-10)(AND)	TMP(0) := $A(0)$ AND '0' TMP(1) := $A(1)$ AND $A(0)$ TMP(2) := $A(2)$ AND $A(1)$ TMP(3) := $A(3)$ AND $A(2)$	00 1010 00 X101	
WHEN-01-CORRUPT (by WHEN-00)(OR)	$\begin{array}{rll} \text{TMP}(0) & := & \text{A}(1) & \text{OR } \textbf{A}(0) \\ \text{TMP}(1) & := & \text{A}(2) & \text{OR } \textbf{A}(1) \\ \text{TMP}(2) & := & \text{A}(3) & \text{OR } \textbf{A}(2) \\ \text{TMP}(3) & := & `0' & \text{OR } \textbf{A}(3) \end{array}$	01 0101 01 101X	
WHEN-01-CORRUPT (by WHEN-00)(AND)	$\begin{array}{llllllllllllllllllllllllllllllllllll$	01 1010 01 x10x	
WHEN-01-CORRUPT (by WHEN-11)(OR)	$\begin{array}{rll} \text{TMP}(0) & := & \text{A}(1) & \text{OR } \textbf{A}(0) \\ \text{TMP}(1) & := & \text{A}(2) & \text{OR } \textbf{A}(1) \\ \text{TMP}(2) & := & \text{A}(3) & \text{OR } \textbf{A}(2) \\ \text{TMP}(3) & := & `0' & \text{OR } \textbf{A}(3) \end{array}$	01 0101 01 101X	
WHEN-01-CORRUPT (by WHEN-11)(AND)	TMP(0) := A(1) AND A(0) TMP(1) := A(2) AND A(1) TMP(2) := A(3) AND A(2) TMP(3) := `0' AND A(3)	01 1010 01 x10x	

Table 7-1	Behavioral	faults f	for exam	ple	SHIF	T4u.
-----------	-------------------	----------	----------	-----	------	------

Behavioral Fault	Corrupted Clause	Test Vectors (OP A)	
WHEN-10-CORRUPT (by WHEN-00)(OR)	TMP(0) := `0'OR A(0) $TMP(1)$:= A(0)OR A(1) $TMP(2)$:= A(1)OR A(2) $TMP(3)$:= A(2)OR A(3)	10 1010 10 X101	
WHEN-10-CORRUPT (by WHEN-00)(AND)	TMP(0) := `0'AND A(0) $TMP(1)$:= A(0)AND A(1) $TMP(2)$:= A(1)AND A(2) $TMP(3)$:= A(2)AND A(3)	10 0101 10 x01x	
WHEN-10-CORRUPT (by WHEN-11)(OR)	TMP(0) := `0'OR A(0) $TMP(1)$:= A(0)OR A(1) $TMP(2)$:= A(1)OR A(2) $TMP(3)$:= A(2)OR A(3)	10 1010 10 X101	
WHEN-10-CORRUPT (by WHEN-11)(AND)	TMP(0) := `0'AND A(0) $TMP(1)$:= A(0)AND A(1) $TMP(2)$:= A(1)AND A(2) $TMP(3)$:= A(2)AND A(3)	10 0101 10 X01X	
WHEN-11-CORRUPT (by WHEN-01)(OR)	TMP(0) := A(0) OR A(1) TMP(1) := A(1) OR A(2) TMP(2) := A(2) OR A(3) TMP(3) := A(3) OR '0'	11 1010 11 X10X	
WHEN-11-CORRUPT (by WHEN-01)(AND)	$\begin{array}{llllllllllllllllllllllllllllllllllll$	11 0101 11 101X	
WHEN-11-CORRUPT (by WHEN-10)(OR)	TMP(0) := $A(0)$ OR '0'TMP(1) := $A(1)$ OR $A(0)$ TMP(2) := $A(2)$ OR $A(1)$ TMP(3) := $A(3)$ OR $A(2)$	11 0101 11 X01X	
WHEN-11-CORRUPT (by WHEN-10)(AND)	TMP(0) := A(0) AND '0' TMP(1) := A(1) AND A(0) TMP(2) := A(2) AND A(1) TMP(3) := A(3) AND A(2)	11 1010 11 X101	

.

%0I	? A	D	:	time	∋;
00	0101	0101	:	500	ns;
00	1010	1010	:	500	ns;
01	0101	0010	:	500	ns;
01	1010	0101	:	500	ns;
10	0101	1010	:	500	ns;
10	1010	0100	:	500	ns;
11	0101	0101	:	500	ns;
11	1010	1010	:	500	ns;

Figure 7-3 WAVES test vectors for example SHIFT4u.

Example SHIFT4u from Figure 7-1 was synthesized to the gate level Structure1 shown in Figure 7-4. Fault simulations were performed using the behavioral test vectors from Figure 7-3. The resulting SSL gate level fault coverage of 90/90 = 100% is shown in Figure 7-5. An alternate synthesis tool and target architecture was next used to produce Structure2 for example SHIFT4u. Fault simulations using the behavioral test vectors from Figure 7-3 resulted in a *SSL* gate level fault coverage of 112/112 = 100%.



Figure 7-4 Synthesized Structure1 for example SHIFT4u.



Figure 7-5 Fault coverage for Structure1 of example SHIFT4u.

7.1.2 Comparison with Previous Fault Models

The only previous behavioral fault model to address the *loop* construct was that proposed by Riesgo and Uceda [60]. As part of faults on *expressions*, the *for_in_loop* construct was faulted by the *index* controlling the *loop* changing its range from the *minimum* to the *maximum*+1 and from the *minimum*-1 to the *maximum*. As can be seen from example SHIFT4u, such faults would produce signals that do not even exist in the synthesized hardware, A(-1) and A(3). Thus, their proposed fault model is more a software mutation, than hardware oriented as they claim.

The restrictions placed on the *loop* construct by the VHDL synthesis subset imply that all such *loops* can be readily expanded. Since this expansion or "unrolling" eliminates the *loops* from the behavioral description, no additional behavioral faults are introduced. The next section will show that the behavior of *functions* and *procedures* is much the same.

7.2 Functions and Procedures

Much like program *loops*, *functions* and *procedures* are used mainly as a convenience for ease of programming. In general, any VHDL code written with *functions* and *procedures* can be mapped to the same hardware as equivalent code without *functions* or *procedures* [10]. Since the VHDL behavioral descriptions can be expanded to eliminate these

134

programming constructs, no additional behavioral faults are introduced by the use of *functions* or *procedures*.

7.2.1 Example ADD4fn

Example ADD4fn in Figure 7-6 describes a 4-bit *ripple carry* adder using *functions* to perform the *sum* and *carry* operations. The functions FA_S and FA_C are shown in Figure 7-7.

```
process(A,B,CIN)
  variable CARRY: std_logic_vector(4 downto 0);
  variable SUM : std_logic_vector(3 downto 0);
begin
  CARRY(0) := CIN;
  for I in 0 to 3 loop
    SUM(I) := FA_S(A(I), B(I), CARRY(I));
    CARRY(I+1) := FA_C(A(I), B(I), CARRY(I));
  end loop;
  S <= SUM;
  COUT <= CARRY(4);
end process;</pre>
```

Figure 7-6 Behavioral description for example ADD4fn.

```
function FA_S (AIN, BIN, CIN: std_logic) return
  std_logic is
begin
  return AIN xor BIN xor CIN;
end FA_S;
function FA_C (AIN, BIN, CIN: std_logic) return
  std_logic is
begin
  return (AIN and BIN) or (AIN and CIN) or
    (BIN and CIN);
end FA_C;
```

Figure 7-7 Functions for example ADD4fn.

The *loop* and *function* programming constructs can be eliminated from the behavioral description by expanding the *loop* and replacing the *function* call with its returned *expression*. The results of this expansion and substitution are shown in Figure 7-8. Since the VHDL behavioral descriptions in Figure 7-6 and Figure 7-8 synthesize to the same gate

level structures, no additional behavioral faults have been introduced by the use of *functions*.

```
begin
  CARRY(0) := CIN;
   SUM(0) := A(0) \text{ xor } B(0) \text{ xor } CARRY(0);
  CARRY(1) := (A(0) \text{ and } B(0)) \text{ or } (A(0) \text{ and } CARRY(0)) \text{ or }
      (B(0) and CARRY(0));
  SUM(1) := A(1) \text{ xor } B(1) \text{ xor } CARRY(1);
  CARRY(2) := (A(1) \text{ and } B(1)) \text{ or } (A(1) \text{ and } CARRY(1)) \text{ or }
      (B(1) and CARRY(1));
   SUM(2) := A(2) \text{ xor } B(2) \text{ xor } CARRY(2);
  CARRY(3) := (A(2) \text{ and } B(2)) \text{ or } (A(2) \text{ and } CARRY(2)) \text{ or }
      (B(2) \text{ and } CARRY(2));
  SUM(3) := A(3) \text{ xor } B(3) \text{ xor } CARRY(3);
  CARRY(4) := (A(3) \text{ and } B(3)) \text{ or } (A(3) \text{ and } CARRY(3)) \text{ or }
    (B(3) and CARRY(3));
   S \leq SUM;
   COUT <= CARRY(4);
end process;
```

Figure 7-8 Expanded behavioral description for example ADD4fn.

7.2.2 Example ADD4pr

The 4-bit *ripple carry* adder can be equivalently written using a *procedure* as shown in Figure 7-9. The *procedure* FA is shown in Figure 7-10.

```
process(A,B,CIN)
variable CARRY: std_logic_vector(4 downto 0);
variable SUM : std_logic_vector(3 downto 0);
begin
CARRY(0) := CIN;
for I in 0 to 3 loop
FA(A(I), B(I), CARRY(I), SUM(I), CARRY(I+1));
end loop;
S <= SUM;
COUT <= CARRY(4);
end process;</pre>
```

Figure 7-9 Behavioral description for example ADD4pr.

136

```
procedure FA (AIN, BIN, CIN: in std_logic;
                              SOUT, COUT: out std_logic) is
begin
   SOUT := AIN xor BIN xor CIN;
   COUT := (AIN and BIN) or (AIN and CIN) or
      (BIN and CIN);
end FA;
```

Figure 7-10 Procedure FA for example ADD4pr.

Since the VHDL behavioral description ADD4pr in Figure 7-9 synthesizes to the same gate level structures as the examples in Figure 7-6 and Figure 7-8, no additional behavioral faults have been introduced by the use of *procedures*.

7.3 Conclusions

VHDL includes other constructs drawn from familiar programming languages. Program *loops*, *functions*, and *procedures* are used in VHDL behavioral descriptions for design simplicity and reuse/repetition of functional blocks. Since any description using these constructs can be rewritten equivalently without them, no additional behavioral faults are implied. Several design examples were used to demonstrate the interaction between previously defined behavioral faults and these other programming constructs.

Chapter 8 Comprehensive Examples

Two comprehensive examples have been chosen to demonstrate the gate level fault coverage of the new behavioral fault models. The first is an arithmetic logic unit (ALU) which performs selected functions on data inputs. The second example is a single error correcting circuit used in fault tolerant applications. Other obvious combinational logic examples such as a multiplexer or a magnitude comparator do not need to be investigated here due to their detailed analysis as part of the development of the fault models for the *if* statement and *relational operators*.

Application of the behavioral fault models to each of the comprehensive examples results in a set of test vectors necessary to detect the behavioral faults. These test vector sets are then applied to synthesized gate level implementations of the behavioral descriptions. Resulting gate level fault coverage is evaluated to determine the effectiveness of the behavioral fault models.

8.1 Arithmetic Logic Unit

The ALU design for this example was created using the LogicLib generator from the Mentor Graphics design tools. The type was selected as an ALU2901 which performs eight arithmetic and logic functions. Data widths of 4- and 8-bits will be evaluated.

8.1.1 Example ALU4wc

The generator parameters and the resulting entity description for a 4-bit ALU with both *carry-in* and *carry-out* are shown in Figure 8-1. The architecture description for

Written by LL_to_VHDL at Mon Jun 8 12:23:29 1998
Parameterized Generator Specification to VHDL Code
LogicLib generator called: ARITHMETIC
Passed Parameters are:
type = ALU2901
$W = 4$
carryin = YES
carryout = YES

Figure 8-1 Entity description for example ALU4wc.

```
library IEEE, ARITHMETIC;
use IEEE.std_logic_1164.all;
use ARITHMETIC.std_logic_arith.all;
-- alu4wc Entity Description
entity alu4wc is
    port(
        OP: in std_logic_vector(2 downto 0);
        A: in std_logic_vector(3 downto 0);
        B: in std_logic_vector(3 downto 0);
        CIN: in std_logic;
        COUT: out std_logic;
        D: out std_logic_vector(3 downto 0)
        );
end alu4wc;
```

Figure 8-1 Entity description for example ALU4wc.

example ALU4wc is shown in Figure 8-2. In the first section, variables are declared and initialized consistent with arithmetic operations involving a *carry-in* and a *carry-out*. Next, two *case* statements determine the appropriate operation to be performed. Lastly, the outputs are assigned based on whether the operation performed was *arithmetic* or *logic*.

```
architecture behave of alu4wc is
begin
  ARITHMETIC_Process: process(A,B,CIN,OP)
    variable operand1: std_logic_vector(4 downto 0);
    variable operand2: std_logic_vector(4 downto 0);
    variable a_ext: std_logic_vector(4 downto 0);
    variable b_ext: std_logic_vector(4 downto 0);
    variable not_a_ext: std_logic_vector(4 downto 0);
    variable not b ext: std_logic_vector(4 downto 0);
    variable carry_ext: std_logic_vector(1 downto 0);
    variable logic_out: std_logic_vector(3 downto 0);
    variable arith_out: std_logic_vector(4 downto 0);
  begin
      -- zero extend inputs to include carry bit
      a_ext := `0' & A;
      b_ext := `0' & B;
      not_a_ext := `0' & not A;
      not_b_ext := `0' & not B;
      carry_ext := '0' & CIN;
```

Figure 8-2 Architecture description for example ALU4wc.

-- ALU2901

```
______
-- Logical Functions --
_____
case OP is
 when "011" =>
   logic_out := A or B;
 when "100" =>
   logic_out := A and B;
 when "101" =>
   logic_out := (not A) and B;
 when "110" =>
   logic_out := A xor B;
 when "111" =>
   logic_out := not (A xor B);
 when others =>
   logic_out := (OTHERS => `X');
end case;
_____
-- Arithmetic Functions --
______
case OP is
 -- Arithmetic operations
 when "000" =>
   operand1 := a_ext;
   operand2 := b_ext;
 when "001" =>
   operand1 := not_a_ext;
   operand2 := b_ext;
 when "010" =>
   operand1 := a_ext;
   operand2 := not_b_ext;
 when others =>
   operand1 := (OTHERS => `X');
   operand2 := (OTHERS => `X');
end case;
arith_out := operand1 + operand2 + carry_ext;
```

Figure 8-2 Architecture description for example ALU4wc.

```
-- assign output
if (OP(2) = `1' or (OP(1) = `1' and OP(0) = `1'))
then
   D <= logic_out;
   COUT <= `X';
else
   D <= arith_out(3 downto 0);
   COUT <= arith_out(4);
end if;
end process ARITHMETIC_Process;
end behave;
```

Figure 8-2 Architecture description for example ALU4wc.

8.1.1.1 Faults on Logical Operators

Within the Logical Functions case statement, the variable $logic_out$ is determined by combining the signals A, B using the logical operators AND, OR, and XOR. For the first logical expression, when OP = 011:

logic_out := A OR B;

From the behavioral fault models for *logical operators*, the three test vectors (AB) necessary for an *OR* operator are *00*, *01*, and *10*. Since all signals are in fact four bits wide, these tests expand to produce the behavioral test vectors shown in Table 8-1.

Expression	OP	А	В	logic_out
A OR B	011	0000	0000	0000
		0000	1111	1111
		1111	0000	1111

 Table 8-1 Behavioral test vectors for OR operator.

The behavioral test vectors for the remaining Logical Functions are determined in a similar manner. The *logical operator AND* also requires three tests: 01, 10 and 11. Next, the XOR operator requires a complete set of four test vectors: 00, 01, 10, and 11. Like the OR operation, these vectors also expand to four bits for each operator. The resulting behavioral test vectors are shown in Table 8-2.

Expression	OP	А	В	logic_out
		0000	1111	0000
A AND B	100	1111	0000	0000
		1111	1111	1111
		1111	1111	0000
(not A) AND B	101	0000	0000	0000
		0000	1111	1111
A XOR B	110	0000	0000	0000
		0000	1111	1111
		1111	0000	1111
		1111	1111	0000
		0000	0000	1111
	111	0000	1111	0000
$\operatorname{Hot}(\mathbf{A} \wedge \mathbf{O} \times \mathbf{D})$	111	1111	0000	0000
		1111	1111	1111

 Table 8-2 Behavioral test vectors for remaining Logical Functions.

8.1.1.2 Faults on Arithmetic Operators

The ADD with carry operation performed in the Arithmetic Functions section determines the variable *arith_out* as follows:

```
arith_out := operand1 + operand2 + carry_ext;
```

Assuming a simple *ripple carry* implementation, the behavioral test vectors for the 4-bit addition come directly from Table 5-6 and are shown here in Table 8-3.

Due to the *case* statement for **Arithmetic Functions**, there are three different ways to form *operand1* and *operand2*. The resulting possible test vectors are shown in Table 8-4. Only one input combination is required for each **Test #**, though using all possibilities would not be incorrect, just redundant. As with previous examples, the *control faults* can provide the necessary insight for selecting a reduced set of behavioral test vectors.

Phase	Test #	operand1	operand2	carry_ext	arith_out
	1	0000	0000	0	0 0000
I 2 3 4	0000	1111	0	0 1111	
	1111	0000	0	0 1111	
	1111	1111	1	1 1111	
п	5	0000	1111	1	1 0000
6	1111	0000	1	1 0000	
III 7 8	7	0101	0101	0	0 1010
	8	1010	1010	· 1	1 0101

Table 8-3 Behavioral tests for 4-bit ADD with carry.

OP	Phase	Test #	А	В	CIN	arith_out
		1	0000	0000	0	0 0000
	т	2	0000	1111	0	0 1111
	1	3	1111	0000	0	0 1111
000		4	1111	1111	1	1 1111
000	тт	5	0000	1111	1	1 0000
	11	6	1111	0000	1	1 0000
	III	7	0101	0101	0	0 1010
		8	1010	1010	1	1 0101
001	I	1	1111	0000	0	0 0000
		2	1111	1111	0	0 1111
		3	0000	0000	0	0 1111
		4	0000	1111	1	1 1111
	тт	5	1111	1111	1	1 0000
	11	6	0000	0000	1	1 0000
	тт	7	1010	0101	0	0 1010
	111	8	0101	1010	1	1 0101

 Table 8-4 Possible test vectors for Arithmetic Functions.

.

OP	Phase	Test #	А	В	CIN	arith_out
		1	0000	1111	0	0 0000
	т	2	0000	0000	0	0 1111
	010	3	1111	1111	0	0 1111
010 П П		4	1111	0000	1	1 1111
	TT	5	0000	0000	1	1 0000
	11	6	1111	1111	1	1 0000
	ш	7	0101	1010	0	0 1010
		8	1010	0101	1	1 0101

 Table 8-4 Possible test vectors for Arithmetic Functions.

8.1.1.3 Control Faults

The control fault model states that each *clause* of an *if* or *case* statement is corrupted by other *clauses* that are logically adjacent. For the **Logical Functions** *case* statement, the first clause, *WHEN-011*, can be corrupted by either *WHEN-001* or *WHEN-010*. Both of these cases fall under the *others clause* and, hence, cause no corruption due to the don't care values. The third possible corruption is caused by the *WHEN-111 clause*. For example, the control fault *WHEN-011 CORRUPT* (by *WHEN-111)*(*OR*) produces the corrupted clause shown below.

logic_out := (A OR B) OR (not (A XOR B));

Test vector generation rules specify that (A OR B) be set to 0, while (not (A XOR B)) is set to 1. Checking the previously determined test vectors for logical operators from Table 8-1 and Table 8-2, the test vector with A = 0000 and B = 0000 meets these requirements and, hence, covers this control fault.

The control faults for all the **Logical Functions** are shown in Table 8-5. The faults are grouped according to the *corrupted clauses*. Each *corrupting clause* includes both an *OR-fault* and an *AND-fault*. The test generation rules provide requirements that produce the appropriate test vector.

Corrupted Clause	Corrupting Clause	Fault	Test Requirement	Test Vector (OP A B)
	001	NONE		
	010	NONE		
011	111	OR	A OR B = 0 not (A XOR B) = 1	011 0000 0000
	111	AND	A OR B = 1 not (A XOR B) = 0	011 0000 1111
	000	NONE		
	101	OR	A AND $B = 0$ (not A) AND $B = 1$	100 0000 1111
100	101	AND	A AND $B = 1$ (not A) AND $B = 0$	100 1111 1111
	110	OR	A AND B = 0 $A XOR B = 1$	100 0000 1111
		AND	A AND B = 1 $A XOR B = 0$	100 1111 1111
101	001	NONE		
	100	OR	(not A) AND B = 0 A AND B = 1	101 1111 1111
		AND	(not A) AND $B = 1$ A AND $B = 0$	101 0000 1111
	111	OR	(not A) AND B = 0 not $(A XOR B) = 1$	101 1111 1111
		AND	(not A) AND B = 1 not $(A XOR B) = 0$	101 0000 1111
	010	NONE		
110	100	OR	A XOR B = 0 $A AND B = 1$	110 1111 1111
	100	AND	$\begin{array}{l} A \text{ XOR } B = 1 \\ A \text{ AND } B = 0 \end{array}$	110 0000 1111

.

 Table 8-5 Control faults for Logical Functions.

.

.

145

Corrupted Clause	Corrupting Clause	Fault	Test Requirement	Test Vector (OP A B)
110	111	OR	A XOR B = 0 not (A XOR B) = 1	110 0000 0000
	111	AND	A XOR B = 1 not (A XOR B) = 0	110 1111 0000
111	011	OR	not (A XOR B) = 0 A OR B = 1	111 0000 1111
		AND	not (A XOR B) = 1 A OR B = 0	111 0000 0000
	101	OR	not (A XOR B) = 0 (not A) AND B = 1	111 0000 1111
		AND	not (A XOR B) = 1 (not A) AND B = 0	111 1111 1111
	110	OR	not (A XOR B) = 0 A XOR B = 1	111 1111 0000
		AND	not (A XOR B) = 1 A XOR B = 0	111 0000 0000

$1 a \nu c \nu$	Table 8-5	Control	faults for	r Logical	Functions
---	-----------	---------	------------	-----------	-----------

The control faults for the **Arithmetic Functions** are formed in the same manner. The resulting faults and their test vectors are shown in Table 8-6.

Corrupted Clause	Corrupting Clause	Fault	Test Requirement	Test Vector (OP A B)
000	001	OR	$a_ext = 0$ not_a_ext = 1	000 0000 xxxx
		AND	$a_ext = 1$ not_a_ext = 0	000 1111 XXXX
	010	OR	b_ext = 0 not_b_ext = 1	000 XXXX 0000
		AND	b_ext = 0 not_b_ext = 1	000 XXXX 1111
	100	NONE		

 Table 8-6 Control faults for Arithmetic Functions.

146

.

•

Corrupted Clause	Corrupting Clause	Fault	Test Requirement	Test Vector (OP A B)
001	000	OR	$not_a_ext = 0$ $a_ext = 1$	001 1111 XXXX
		AND	$not_a_ext = 1$ $a_ext = 0$	001 0000 XXXX
	011	NONE		
	101	NONE		
010	000	OR	$not_b_ext = 0$ $b_ext = 1$	010 XXXX 1111
		AND	$not_b_ext = 1$ b_ext = 0	010 XXXX 0000
	011	NONE		
	110	NONE		

Table 8-6 Control faults for Arithmetic Functions.

The final control faults affect the *if* statement that assigns the outputs. The *then* and *else* clauses determine whether the function performed was *logic* or *arithmetic*, respectively. The logical adjacencies among the clauses are illustrated in Figure 8-3; *logic* functions corresponding to the *then* clause are shaded. For example, the *logic* function for OP = 011 can be corrupted by the *arithmetic* functions with OP = 001 and OP = 010.



Figure 8-3 Logical adjacencies among clauses.

THEN-CORRUPT faults cause *logic* functions to be corrupted by *arithmetic* functions. As an example, the control fault 100-CORRUPT (by 000)(OR) produces the following corrupted assignment statements:

D <= logic_out OR arith_out(3 downto 0); COUT <= `X' OR arith_out(4);</pre>

Candidate test vectors for OP = 100 with corresponding values for $logic_out$ are found in Table 8-2. Corrupting values of $arith_out(3 \ downto \ 0)$ for OP = 000 can be found in Table 8-4. Comparing these values shows that an appropriate test vector is formed by A = 0000 and B = 1111 with CIN = 0. Note that a *logic* operation would normally leave CIN as a don't care, however, corruption by an *arithmetic* operation requires specification of this value. *THEN-CORRUPT* faults and their resulting test vectors are shown in Table 8-7.

Corrupted Clause	Corrupting Clause	Fault _.	Test Requirement	Test Vector (OP A B CIN)
	001	OR	logic_out = 0 arith_out = 1	011 0000 0000 0
011	001	AND	logic_out = 1 arith_out = 0	011 1111 0000 0
	010	OR	logic_out = 0 arith_out = 1	011 0000 0000 0
	010	AND	logic_out = 1 arith_out = 0	011 0000 1111 0
100	000	OR	$logic_out = 0$ arith_out = 1	100 0000 1111 0
		AND	logic_out = 1 arith_out = 0	100 0101 0101 0 100 1010 1010 1
101	001	OR	logic_out = 0 arith_out = 1	101 1111 1111 0
		AND	logic_out = 1 arith_out = 0	101 1010 0101 0 101 0101 1010 1
110	010	OR	logic_out = 0 arith_out = 1	110 0000 0000 0
	010	AND	logic_out = 1 arith_out = 0	110 0000 1111 0

Table 8-7 THEN-CORRUPT control faults.

ELSE-CORRUPT faults are formed in a similar manner resulting in corruption of an *arithmetic* function by a *logic* function. ELSE-CORRUPT faults are shown in Table 8-8.

Corrupted Clause	Corrupting Clause	Fault	Test Requirement	Test Vector (OP A B CIN)
	100	OR	arith_out = 0 logic_out = 1	000 0101 0101 0 000 1010 1010 1
000	100	AND	arith_out = 1 logic_out = 0	000 0101 0101 0 000 1010 1010 1
001	011	OR	arith_out = 0 logic_out = 1	001 1111 0000 0
	011	AND	arith_out = 1 logic_out = 0	001 0000 0000 0
	101	OR	arith_out = 0 logic_out = 1	001 1010 0101 0 001 0101 1010 1
		AND	arith_out = 1 logic_out = 0	001 1010 0101 0 001 0101 1010 1
010	011	OR	arith_out = 0 logic_out = 1	010 1111 1111 1
		AND	arith_out = 1 logic_out = 0	010 0000 0000 0
	110	OR	arith_out = 0 logic_out = 1	010 0000 1111 [.] 0
	110	AND	arith_out = 1 logic_out = 0	010 0000 0000 0

 Table 8-8 ELSE-CORRUPT control faults.

8.1.1.4 Final Behavioral Test Vector Set

Combining the control faults for Logical Functions from Table 8-5 and the *THEN*-*CORRUPT* control faults from Table 8-7 with the tests for Logical Functions from Table 8-1 and Table 8-2 will produce a final set of behavioral test vectors for OP = 011 through OP = 111. Further optimization is possible by noting that the two test vectors required for the control fault 100-CORRUPT (by 000)(AND) provide coverage for the A = 1, B = 1fault to the AND operator for OP = 100. A similar optimization applies to the control fault 101-CORRUPT (by 001)(AND). The resulting behavioral test vectors and covered control faults are shown in Table 8-9.

Expression	Test Vector (OP A B CIN)	Control Faults	
	011 0000 0000 0	011 by 001 (OR), 011 by 010 (OR), 011 by 111 (OR)	
A OR B	011 0000 1111 0	011 by 010 (AND), 011 by 111 (AND)	
	011 1111 0000 0	011 by 001 (AND)	
•	100 0000 1111 0	100 by 000 (OR), 100 by 101 (OR), 100 by 110 (OR)	
A AND B	100 1111 0000 X		
	100 0101 0101 0 100 1010 1010 1	100 by 000 (AND), 100 by 101 (AND), 100 by 110 (AND)	
	101 1111 1111 0	101 by 001 (OR), 101 by 100 (OR), 101 by 111 (OR)	
(not A) AND B	101 0000 0000 X		
	101 1010 0101 0 101 0101 1010 1	101 by 001 (AND), 101 by 100 (AND), 101 by 111 (AND)	
	110 0000 0000 0	110 by 010 (OR), 110 by 111 (OR)	
A YOD D	110 0000 1111 0	110 by 010 (AND), 110 by 100 (AND)	
A AOR B	110 1111 0000 X	110 by 111 (AND)	
	110 1111 1111 X	110 by 100 (OR)	
	111 0000 0000 X	111 by 011 (AND), 111 by 110 (AND)	
not (A XOR B)	111 0000 1111 X	111 by 011 (OR), 111 by 101 (OR)	
	111 1111 0000 X	111 by 110 (OR)	
	111 1111 1111 X	111 by 101 (AND)	

 Table 8-9 Final behavioral test vectors for Logical Functions.

Likewise, the control faults for Arithmetic Functions from Table 8-6 and the *ELSE*-CORRUPT control faults from Table 8-8 are combined with the possible test vectors for Arithmetic Functions from Table 8-4. The only behavioral tests, for the 4-bit ADD with carry, not specified by control faults are Test 5 and Test 6. Test vectors to cover this Phase were selected from OP = 010 simply to balance the number of test vectors in each group. The resulting behavioral test vectors and covered control faults are shown in Table 8-10.

150

Phase	Test #	Test Vector (OP A B CIN)	Control Faults		
	1	000 0000 0000 0	000 by 001 (OR), 000 by 010 (OR)		
		001 1111 0000 0	001 by 000 (OR), 001 by 011 (OR)		
Ŧ		010 0000 1111 0	010 by 000 (OR), 010 by 011 (OR), 010 by 110 (OR)		
ł	2	010 0000 0000 0	010 by 000 (AND), 010 by 011 (AND), 010 by 110 (AND)		
	3	001 0000 0000 0	001 by 000 (AND), 001 by 011 (AND)		
	4	000 1111 1111 1	000 by 001 (AND), 000 by 010 (AND)		
т	5	010 0000 0000 1			
11	6	010 1111 1111 1			
	7	000 0101 0101 0	000 by 100 (OR) 000 by 100 (AND)		
ш	8	000 1010 1010 1	000 by 100 (OK), 000 by 100 (AND)		
	7	001 1010 0101 0	0.01 by 101 (OR) 0.00 by 101 (AND)		
	8	001 0101 1010 1	001 by 101 (OK), 000 by 101 (AND)		

 Table 8-10 Final behavioral test vectors for Arithmetic Functions.

Hence, a final set of 31 test vectors has been formed by application of the new behavioral fault models. The behavioral test vectors and resulting outputs are presented in WAVES format in Figure 8-4.

% OP A	B CIN	COU	ГD:	time	∋;	
000 000	0 0000) 0 (0000):5	500	ns;
000 111	.1 1111	L 1 1	L 1111	. : 5	500	ns;
000 010	1 0101	L 0 () 1010) : 5	500	ns;
000 101	.0 1010) 1 1	L 0101	. : 5	500	ns;
8						
001 111	.1 0000) 0 (0000):5	500	ns;
001 000	0 0000) 0 () 1111	. : 5	500	ns;
001 101	0 0101	L 0 () 1010):5	500	ns;
001 010	1 1010) 1 1	L 0101	. : 5	500	ns;
8						

Figure 8-4 WAVES test vectors for example ALU4wc.

Figure 8-4 WAVES test vectors for example ALU4wc.

8.1.2 Evaluation of the Behavioral Test Vectors

Example ALU4wc was first synthesized to gate level Structure1 using AutoLogic II. The resulting optimized implementation contains 59 gates shown in Figure 8-5. According to MIL-STD 883D, Structure1 of example ALU4wc contains 284 unique *SSL* gate level faults. Fault simulations using the behavioral test vectors from Figure 8-4 resulted in complete gate level fault coverage shown in Figure 8-6.

An alternate target technology was next used to synthesize gate level Structure2. The resulting optimized circuit contains 78 gates and 312 unique *SSL* faults. Fault simulations using the behavioral test vectors from Figure 8-4 again resulted in complete gate level fault coverage.



Figure 8-5 Synthesized Structure1 for example ALU4wc.



Figure 8-6 Fault coverage for Structure1 of example ALU4wc.

Lastly, an alternate synthesis tool, Leonardo, was used to map the VHDL behavioral description to a Xilinx FPGA architecture. Fault simulations using the behavioral test vectors achieved a *SSL* gate level fault coverage of 398/398 = 100%. The behavioral fault models have been applied to multiple implementations using various target architectures and synthesis tools. The range of examples demonstrates the flexibility of the approach and provides experimental validation of the effectiveness of the new fault models.

153

8.1.3 Expansion of the Data Path

The data path for the arithmetic logic unit was next expanded to eight bits wide to create example ALU8wc. The only difference in the resulting behavioral description is the width of the corresponding variables and signals. Since the width of the control signal *OP* remained constant, no new control faults were introduced.

The behavioral test vectors for example ALU8wc, therefore, follow directly from those derived for example ALU4wc. The only change is the expansion of the data signals A, B, and D to eight bits wide. The resulting test vectors are presented in Figure 8-7. Note that the WAVES file still contains only 31 test vectors.

```
% OP A B CIN COUT D : time;
000 00000000 00000000 0 0 00000000 : 500 ns;
000 11111111 1111111 1 1 11111111 : 500 ns;
000 01010101 01010101 0 0 10101010 :
                                      500 ns;
000 10101010 10101010 1 1 01010101 : 500 ns;
웅
001 11111111 00000000 0 0 00000000 : 500 ns;
001 0000000 0000000 0 0 11111111 :
                                      500 ns;
001 10101010 01010101 0 0 10101010 : 500 ns;
001 01010101 10101010 1 1 01010101 :
                                      500 ns;
ષ્ટ
010 00000000 11111111 0 0 00000000 : 500 ns;
010 0000000 0000000 0 0 11111111 :
                                      500 ns;
010 0000000 0000000 1 1 00000000 :
                                      500 ns;
010 11111111 11111111 1 1 00000000 : 500 ns;
g
011 00000000 00000000 0 X 00000000 : 500 ns;
011 00000000 11111111 0 X 11111111 : 500 ns;
011 11111111 00000000 0 X 11111111 : 500 ns;
z
100 00000000 11111111 0 X 00000000 : 500 ns;
100 11111111 00000000 X X 00000000 : 500 ns;
100 01010101 01010101 0 X 01010101 : 500 ns;
100 10101010 10101010 1 X 10101010 : 500 ns;
웅
101 11111111 11111111 0 X 00000000 : 500 ns;
101 00000000 00000000 X X 00000000 : 500 ns;
101 10101010 01010101 0 X 01010101 : 500 ns;
101 01010101 10101010 1 X 10101010 : 500 ns;
```

Figure 8-7 WAVES test vectors for example ALU8wc.

110	00000000	00000000	0	Х	00000000	:	500	ns;
110	00000000	11111111	0	Х	11111111	:	500	ns;
110	11111111	00000000	Х	Х	11111111	:	500	ns;
110	11111111	11111111	Х	Х	00000000	:	500	ns;
8								
111	00000000	00000000	Х	х	11111111	:	500	ns;
111	00000000	11111111	Х	х	00000000	:	500	ns;
111	11111111	00000000	Х	х	00000000	:	500	ns;
111	11111111	11111111	Х	х	11111111	:	500	ns;

Figure 8-7 WAVES test vectors for example ALU8wc.

Example ALU8wc was next synthesized using AutoLogic II to produce Structure1 containing 130 gates. Fault simulations using the behavioral test vectors from Figure 8-7 resulted in a SSL gate level fault coverage of 542/542 = 100%. Lastly, Leonardo was used to synthesize example ALU8wc to gate level Structure2. Fault simulations using the behavioral test vectors again achieved complete gate level fault coverage.

8.1.4 Summary

The new behavioral fault models, developed in this dissertation, have been applied to the comprehensive examples ALU4wc and ALU8wc. From these faults, behavioral test vectors have been derived for the 4-bit case, then readily expanded for the 8-bit example. The resulting complete *SSL* gate level fault coverage for multiple implementations is summarized in Table 8-11. Again, the range of examples demonstrates the flexibility of the approach and provides experimental validation of the effectiveness of the new fault models.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
ALU4wc	Structure1	284	31	100%
	Structure2	312	31	100%
	Structure3	398	31	100%
A T T TO	Structure1	546	31	100%
ALU8wc	Structure2	758	31	100%

 Table 8-11
 ALU fault experiment results.

8.2 Error Correcting Circuit

The second comprehensive example is a combinational circuit capable of correcting single-bit errors in data words. The error correction capability is achieved by the use of multiple overlapping parity bits forming a Hamming code. If x is the number of information bits, then the number of parity bits, p, is equal to the smallest integer value of p that satisfies $2^{p} \ge x + p + 1$ [41]. Hence, four data bits (X) would require three parity bits (P) to create example HAMMING4. Likewise, eight data bits require four parity bits for example HAMMING8.

8.2.1 Example HAMMING4

The entity description for example HAMMING4 is shown in Figure 8-8. This module assumes the existence of another circuit which generates the parity bits (P) from the data bits (X). The data/parity combination X,P is then subject to corruption prior to processing by example HAMMING4. The error correcting circuit takes the input data and parity bits and performs single bit error correction to produce the output data bits (D).

```
entity hamming4 is
   port(
        X: in std_logic_vector(1 to 4);
        P: in std_logic_vector(1 to 3);
        D: out std_logic_vector(1 to 4)
     );
end hamming4;
```

Figure 8-8 Entity description for example HAMMING4.

The architecture description for example HAMMING4, shown in Figure 8-9, contains two parts. In the first section, the data bits and parity bits are combined using XOR trees to generate check bits forming a *syndrome* (S). Detection of a single-bit error produces a 1 on one or more bits of the *syndrome*. Next, the overall value of the *syndrome* bits determines which, if any, data bit needs corrected.

8.2.1.1 Faults on XOR-only Expressions

Behavioral faults on the *expressions* for the *syndrome* (S) are all on *XOR operators*. Hence, optimized test vectors can be generated based on the modified Bossen algorithm developed in Section 6.1.2.2.

```
architecture behave of hamming4 is
begin
process(X,P)
variable S : std_logic_vector(1 to 3);
begin
    S(1) := X(1) XOR X(2) XOR X(4) XOR P(1);
    S(2) := X(1) XOR X(2) XOR X(4) XOR P(2);
    S(3) := X(2) XOR X(3) XOR X(4) XOR P(2);
    S(3) := X(2) XOR X(3) XOR X(4) XOR P(3);
    D(1) <= X(1) XOR (S(1) AND S(2) AND not S(3));
    D(2) <= X(2) XOR (S(1) AND not S(2) AND S(3));
    D(3) <= X(3) XOR (not S(1) AND S(2) AND S(3));
    D(4) <= X(4) XOR (S(1) AND S(2) AND S(3));
    end process;
end behave;
```

Figure 8-9 Architecture description for example HAMMING4.

A generic 4-input XOR-only expression is shown below:

Z <= A XOR B XOR C XOR D

Applying the modified Bossen algorithm results in the labelling shown in Figure 8-10 and Figure 8-11. Note that this example represents a special case where it is possible to find identical test sequences for structures Cascade1 and Cascade2. The resulting optimized test vectors for a generic 4-input XOR-only expression are shown in Table 8-12.



Figure 8-10 Structure Cascade1 for 4-input XOR-only expression.



Figure 8-11 Structure Cascade2 for 4-input XOR-only expression.
Signal	Label	Sequence	Test Vector (ABCD)	Z
Α	S	0011	0000	0
В	R	0110	0111	1
С	R	0110	1110	1
D	Т	0101	1001	0

Table 8-12 Optimized test vectors for 4-input XOR-only expression.

Mapping the generalized case onto the first XOR-only expression relates A to X(1), B to X(2), C to X(4), D to P(1), and Z to S(1). The resulting behavioral test vectors are shown in Table 8-13. Mapping the generalized case onto expressions S(2) and S(3) produces the test vectors shown in Table 8-14 and Table 8-15, respectively. Don't care values can be eliminated for the three XOR-only expressions, resulting in a reduced set of behavioral test vectors shown in Table 8-16.

$\begin{array}{c} ABCD\\ (X_1X_2X_4P_1) \end{array}$	Z (S ₁)	$\begin{array}{c} X\\ (X_1X_2X_3X_4)\end{array}$	$\begin{array}{c} P\\ (P_1P_2P_3)\end{array}$	S (S ₁ S ₂ S ₃)
0000	0	00x0	0XX	0XX
0111	1	01X1	1XX	1XX
1110	1	11X1	0XX	1XX
1001	0	10X0	1XX	0XX

Table 8-13 Optimized test vectors for expression S(1).

$\begin{array}{c} \text{ABCD} \\ (X_1 X_3 X_4 P_2) \end{array}$	Z (S ₂)	$\begin{array}{c} X \\ (X_1 X_2 X_3 X_4) \end{array}$	$\begin{array}{c} P\\ (P_1P_2P_3)\end{array}$	S (S ₁ S ₂ S ₃)
0000	0	0x00	XOX	x0x
0111	1	0X11	X1X	X1X
1110	1	1X11	x0x	X1X
1001	0	1X00	X1X	XOX

Table 8-14 Optimized test vectors for expression S(2).

$\begin{bmatrix} ABCD \\ (X_2X_3X_4P_3) \end{bmatrix}$	Z (S ₃)	$\begin{array}{c} X\\ (X_1X_2X_3X_4)\end{array}$	$\begin{array}{c} P\\ (P_1P_2P_3)\end{array}$	S (S ₁ S ₂ S ₃)
0000	0	X000	XX0	XX0
0111	1	X011	XX1	XX1
1110	1	X111	XX0	XX1
1001	0	X100	XX1	XX0

 Table 8-15 Optimized test vectors for expression S(3).

X ₁ X ₂ X ₃ X ₄	$P_1P_2P_3$	S ₁ S ₂ S ₃
0000	000	000
0011	101	001
0100	101	000
0111	110	111
1000	110	000
1111	000	111

Table 8-16 Reduced test vector set for XOR-only expressions.

8.2.1.2 Faults on Other Logical Expressions

Behavioral faults for the remaining expressions are all on logical operators. Test vectors can be generated using the parse tree method developed in Section 6.1.1.2. A parse tree for expression D(1) is shown in Figure 8-12. The binary nodes (1,2,3) are formed by the logical operators.



Figure 8-12 Parse tree for expression D(1).

Application of the new behavioral fault models implies four faults for the XOR operator and three faults for each of the AND operators. The resulting behavioral test vectors are presented in Table 8-17. Eliminating don't care values results in the reduced set of behavioral test vectors for expression D(1) shown in Table 8-18. Application of similar parse trees to the remaining logical expressions produces the reduced test vectors in Table 8-19 through Table 8-21.

Node	Behavioral Test	Test Requirements	Test Vector $(X_1 S_1 S_2 S_3)$
	00	$X_1 = 0$, $(S_1 \text{ AND } S_2 \text{ AND not } S_3) = 0$	0 0XX
1	01	$X_1 = 0$, $(S_1 \text{ AND } S_2 \text{ AND not } S_3) = 1$	0 110
	10	$X_1 = 1$, (S ₁ AND S ₂ AND not S ₃) = 0	1 0XX
	11	$X_1 = 1$, (S ₁ AND S ₂ AND not S ₃) = 1	1 110
	01	$S_1 = 0$, $(S_2 \text{ AND not } S_3) = 1$, $X_1 = 0$	0 010
2	10	$S_1 = 1$, (S_2 AND not S_3) = 0, $X_1 = 0$	0 10X
	11	$S_1 = 1$, (S_2 AND not S_3) = 1, $X_1 = 0$	0 110
	01	$S_2 = 0$, not $S_3 = 1$, $S_1 = 1$, $X_1 = 0$	0 100
3	10	$S_2 = 1$, not $S_3 = 0$, $S_1 = 1$, $X_1 = 0$	0 110
	11	$S_2 = 1$, not $S_3 = 1$, $S_1 = 1$, $X_1 = 0$	0 111

Table 8-17 Behavioral test vectors for expression D(1).

:

X ₁ X ₂ X ₃ X ₄	\$1\$253	Functional Tests (Node [Test])
0XXX	010	1 [00], 2 [01]
0XXX	100	2 [10], 3 [01]
0XXX	110	1 [01], 2 [11], 3 [10]
0XXX	111	3 [11]
1XXX	0XX	1[10]
1XXX	110	1 [11]

 Table 8-18 Reduced test vectors for expression D(1).

X ₁ X ₂ X ₃ X ₄	S ₁ S ₂ S ₃	Functional Tests (Node [Test])
XOXX	001	1 [00], 2 [01]
XOXX	100	3 [10]
X0XX	101	1 [01], 2 [11], 3 [11]
X0XX	111	2 [10], 3 [01]
X1XX	0XX	1[10]
X1XX	101	1[11]

.

•

Table 8-19	Reduced	test	vectors for	expression	D	(2).
-------------------	---------	------	-------------	------------	---	----	----

X ₁ X ₂ X ₃ X ₄	S ₁ S ₂ S ₃	Functional Tests (Node [Test])
XX0X	001	2 [10], 3 [01]
xx0x	010	3 [10]
XXOX	011	1 [01], 2 [11], 3 [11]
XX0X	111	1 [00], 2 [01]
XX1X	011	1 [11]
XX1X	1XX	1 [10]

Table 8-20 Reduced test vectors for expression D(3).

X ₁ X ₂ X ₃ X ₄	S ₁ S ₂ S ₃	Functional Tests (Node [Test])
XXX0	011	1 [00], 2 [01]
XXX0	101	2 [10], 3 [01]
XXX0	110	3 [10]
XXX0	111	1 [01], 2 [11], 3 [11]
XXX1	0XX	1 [10]
XXX1	111	1 [11]

Table 8-21 Reduced test vectors for expression D(4).

Eliminating don't care values by combining test vectors for logical expressions D(1) through D(4) produces the reduced set of behavioral test vectors shown in Table 8-22.

$X_1X_2X_3X_4$	$S_1S_2S_3$
1001	001
0101	010
1100	011
1110	011
001X	100
X010	101
X110	101
0X10	110
1X10	110
0000	111
0001	111

Table 8-22 Test vectors for logical expressions D(1) through D(4).

8.2.1.3 Final Behavioral Test Vector Set

The reduced set of behavioral test vectors for the XOR-only expressions, Table 8-16, can now be combined with the behavioral test vectors for the other *logical expressions*, Table 8-22. Unspecified don't care values are arbitrarily set to 0. The resulting test vectors were sorted and are presented in WAVES format in Figure 8-13.

	% X	Ρ	D	:	time	e;
·	0000	000	0000	:	500	ns;
	0000	111	0001	:	500	ns;
	0001	000	0000	:	500	ns;
	0010	101	1010	:	500	ns;
	0010	110	0110	:	500	ns;
	0010	111	0010	:	500	ns;
	0011	101	0011	:	500	ns;
	0100	101	0100	:	500	ns;
	0101	000	0101	:	500	ns;
	0110	011	0010	:	500	ns;
	0111	110	0110	:	500	ns;

Figure 8-13 WAVES test vectors for example HAMMING4.

1000	110	1000	:	500	ns;
1001	000	1001	:	500	ns;
1010	011	0010	:	500	ns;
1100	000	1110	:	500	ns;
1110	011	1100	:	500	ns;
1111	000	1110	:	500	ns;

Figure 8-13 WAVES test vectors for example HAMMING4.

8.2.2 Evaluation of the Behavioral Test Vectors

Example HAMMING4 was first synthesized to gate level Structure1 using AutoLogic II. The resulting optimized implementation contains 21 gates shown in Figure 8-14. According to MIL-STD 883D, Structure1 of example HAMMING4 contains 114 unique *SSL* gate level faults. Fault simulations using the behavioral test vectors from Figure 8-13 resulted in complete gate level fault coverage.



Figure 8-14 Synthesized Structure1 for example HAMMING4.

An alternate target technology was next used to synthesize gate level Structure2. The resulting optimized circuit contains 19 gates and 116 unique *SSL* faults. Fault simulations using the behavioral test vectors from Figure 8-13 again resulted in complete gate level fault coverage. Lastly, Leonardo was used to map the VHDL behavioral description to a Xilinx FPGA architecture. Fault simulations using the behavioral test vectors achieved a *SSL* gate level coverage of 226/226 = 100%.

8.2.3 Expansion of the Data Path

The data path for the error correcting circuit was next expanded to eight bits wide to create example HAMMING8. As previously stated, four parity bits are now required to provide single-bit error correction capability. The entity description is shown in Figure 8-15. The architecture description for example HAMMING8 is shown in Figure 8-16. Note the additional *syndrome expression* due to the 4th parity bit.

```
entity hamming8 is
   port(
      X: in std_logic_vector(1 to 8);
      P: in std_logic_vector(1 to 4);
      D: out std_logic_vector(1 to 8)
   );
end hamming8;
```

Figure 8-15 Entity description for example HAMMING8.

```
architecture behave of hamming8 is
begin
  process(X,P)
     variable S : std_logic_vector(1 to 4);
  begin
     S(1) := X(1) \text{ XOR } X(2) \text{ XOR } X(4) \text{ XOR } X(5) \text{ XOR } X(7)
                 XOR P(1);
     S(2) := X(1) \text{ XOR } X(3) \text{ XOR } X(4) \text{ XOR } X(6) \text{ XOR } X(7)
                 XOR P(2);
     S(3) := X(2) \text{ XOR } X(3) \text{ XOR } X(4) \text{ XOR } X(8) \text{ XOR } P(3);
     S(4) := X(5) XOR X(6) XOR X(7) XOR X(8) XOR P(4);
     D(1) \leq X(1) \text{ XOR } (S(1) \text{ AND } S(2) \text{ AND not } S(3)
                 AND not S(4));
     D(2) \leq X(2) \text{ XOR } (S(1) \text{ AND not } S(2) \text{ AND } S(3)
                 AND not S(4));
     D(3) \leq X(3) \text{ XOR (not } S(1) \text{ AND } S(2) \text{ AND } S(3)
                 AND not S(4);
     D(4) \leq X(4) \text{ XOR } (S(1) \text{ AND } S(2) \text{ AND } S(3)
                 AND not S(4);
     D(5) \leq X(5) XOR (S(1) AND not S(2) AND not S(3)
                 AND S(4);
     D(6) \leq X(6) XOR (not S(1) AND S(2) AND not S(3)
                 AND S(4));
     D(7) \leq X(7) XOR (S(1) AND S(2) AND not S(3)
                 AND S(4);
     D(8) \leq X(8) XOR (not S(1) AND not S(2) AND S(3)
                 AND S(4);
  end process;
end behave;
```

Figure 8-16 Architecture description for example HAMMING8.

None of the test vectors for example HAMMING4 can be readily expanded for use with example HAMMING8. However, deriving the behavioral test vectors follows the same process that was used in Section 8.2.1.1 and Section 8.2.1.2. Generalized Bossen

test vectors can be found for the XOR-only expressions and parse trees can be used for the remaining *logical expressions*. The resulting 10 + 23 = 33 test vectors are shown in WAVES format in Figure 8-17.

% X	P	D	:	time	е;	
00000000	0000	00000000	:	500	ns;	
01101111	0000	01111111	:	500	ns;	
01110010	0010	01110000	:	500	ns;	
10011101	1100	10011101	:	500	ns;	
11101110	1101	01101110	:	500	ns;	
11110010	1110	11110000	:	500	ns;	
00100101	0011	00100100	:	500	ns;	
00110110	0011	00110111	:	500	ns;	
01011011	0000	01011010	:	500	ns;	
01011011	0011	01011011	:	500	ns;	
8						
11010010	0100	11010010	:	500	ns;	
10011010	0100	10011010	:	500	ns;	
11011010	1111	11011011	:	500	ns;	
11011011	1100	11011010	:	500	ns;	
01011010	0100	01011010	:	500	ns;	
11011000	0100	11011100	:	500	ns;	
11011100	0001	11011000	:	500	ns;	
11001010	0100	11101010	:	500	ns;	
11101010	0010	11001010	:	500	ns;	
11011010	1011	11011010	:	500	ns;	
00100101	1000	00100101	:	500	ns;	
00100101	1001	00101101	:	500	ns;	· · · ·
00101101	0000	00100101	:	500	ns;	
00100101	1010	01100101	:	500	ns;	
01100101	0000	00100101	:	500	ns;	
00100100	1000	00100100	:	500	ns;	
00100101	1100	10100101	:	500	ns;	
10100101	0000	00100101	:	500	ns;	
00100001	1000	00100011	:	500	ns;	
00100011	0101	00100001	:	500	ns;	
00000101	1000	00010101	:	500	ns;	
00010101	0110	00000101	:	500	ns;	
00100101	1111	00100101	:	500	ns;	

Figure 8-17 WAVES test vectors for example HAMMING8.

Example HAMMING8 was next synthesized using AutoLogic II to produce Structure1 containing 36 gates. Fault simulations using the behavioral test vectors from Figure 8-17

resulted in a SSL gate level fault coverage of 226/226 = 100%. Lastly, Leonardo was used to synthesize example HAMMING8 to gate level Structure2. Fault simulations using the behavioral test vectors again achieved complete gate level fault coverage.

8.2.4 Summary

The new behavioral fault models, developed in this dissertation, have been applied to the comprehensive examples HAMMING4 and HAMMING8. From these faults, behavioral test vectors were derived for both the 4-bit and 8-bit cases. The resulting complete *SSL* gate level fault coverage for multiple implementations is summarized in Table 8-23.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
HAMMING4	Structure1	114	17	100%
	Structure2	116	17	100%
	Structure3	226	17	100%
HAMMING8	Structure1	226	33	100%
	Structure2	446	33	100%

 Table 8-23 HAMMING fault experiments.

8.3 Conclusions

Two comprehensive examples were chosen to demonstrate the gate level fault coverage of the new behavioral fault models. The ALU involved the interaction of control faults with both *arithmetic* and *logical operator* faults. The single error correcting circuit, HAMMING, used both XOR-only and mixed *logical operator expressions*. Application of the new fault models to the comprehensive examples resulted in sets of test vectors necessary to detect the behavioral faults. These test vectors were then applied to synthesized gate level implementations of the behavioral descriptions. The resulting complete *SSL* gate level fault coverage provides experimental validation of the effectiveness of the behavioral fault models.

Chapter 9 Conclusions and Future Work

This chapter summarizes the research contributions of this dissertation and outlines directions for future work. Some concluding remarks are also provided.

9.1 Research Contributions

The main contributions of this dissertation include improved behavioral fault models as well as the techniques for generalizing the effects of low level faults and abstracting them into the behavioral domain. The new fault models are more closely linked to underlying hardware faults than those developed by previous research. Test vectors based on these new behavioral fault models achieve complete *SSL* gate level fault coverage over a broad range of implementations.

9.1.1 Generalized Functional Faults

A functional analysis technique has been developed for generalizing the effects of industry standard *single-stuck-line (SSL)* faults on gate level circuits. The key is determining sets of functional faults which are not tied to a specific realization. What is desired is a general set of faults which provide coverage for functional faults from multiple implementations.

For regular structures, such as cellular logic arrays and parity trees, complete functional testing is achieved by exhaustive testing of each functional building block. For other functions, faults are generalized from *sum-of-products (SOP)* and *product-of-sums* (*POS*) implementations to obtain a set of functional faults not tied a specific realization. As was the case with *physically-induced* faults [29], a unique fault produced by a particular realization can be readily added to the set of functional faults.

This dissertation has used the SSL fault model as the basis for its higher level fault models. The generalization and abstraction techniques developed here are not dependent on this choice of a low level fault model. Other low level fault models which complement or improve on the SSL fault model can also be readily applied.

9.1.2 New Behavioral Fault Models

New behavioral fault models have been developed, which are more closely linked to underlying hardware faults than previous fault models. The effects of the generalized sets of functional faults are abstracted into the behavioral domain by establishing a relationship between the higher level language construct and the lower level faults it should encompass.

The fault modeling technique used throughout this dissertation is that of external corruption of the original VHDL constructs, rather than replacement/mutation of operators. Where possible, specific faults have been defined, such as the *Clause-CORRUPT* control faults. When a direct mapping of functional faults cannot be made to produce a simple behavioral fault model, an error vector approach has been applied. The functional test vectors are mapped into error vectors which then corrupt the results of the VHDL operation for the appropriate input combinations.

While the new fault models are definitely more complex than previous ones, this is because they more accurately reflect the underlying complexity of the hardware faults which they attempt to model. The increased complexity of the fault models eliminates the need to supplement behavioral test vector sets via heuristics in order to improve gate level fault coverage.

9.1.3 Gate Level Fault Coverage of Behavioral Test Vectors

Application of the behavioral fault models to examples throughout this dissertation resulted in sets of test vectors necessary to detect the behavioral faults. Fault experiments were then performed using the behavioral test vectors and synthesized gate level implementations. Multiple synthesis tools and target architectures were employed to create a broad range of realizations of the behavioral descriptions. Resulting gate level fault coverage was evaluated to illustrate the effectiveness of the behavioral fault models and is summarized in Appendix B.

Two comprehensive examples were chosen to demonstrate the gate level fault coverage of the new behavioral fault models. The ALU involved the interaction of control faults with both *arithmetic* and *logical operator* faults. The single error correcting circuit, HAMMING, used both XOR-only and mixed *logical operator expressions*. Application of

- 168

the new fault models to the comprehensive examples resulted in sets of test vectors necessary to detect the behavioral faults. These test vectors were then applied to synthesized gate level implementations of the behavioral descriptions. The resulting complete *SSL* gate level fault coverage provides experimental validation of the effectiveness of the behavioral fault models.

9.1.4 Behavioral Test Generation

The base fault model for *arithmetic operators* is derived from the *ripple carry* connection of 2- and 3-input functional building blocks. Behavioral test generation rules, presented in this dissertation, demonstrate that only eight test vectors are required for complete gate level fault coverage, regardless of the size of the *operands*. For *logical operators*, behavioral test generation rules were developed for the special case of XORonly *expressions*. A generalized Bossen algorithm is presented that allows for optimization of test sequences while allowing for multiple possible realizations.

Most fault-oriented techniques use some form of a three step approach to the test generation process. First, a fault must be *activated* at the desired location in the circuit model. Then, the effect of the fault must be *propagated* to a point where it can be observed and, hence, detected. Finally, the inputs of the model must be determined to *justify* the desired signal values throughout the circuit. Variations of these techniques attempt to utilize the information available in higher level models to more efficiently accomplish the computationally intensive tasks of fault *propagation* and *justification*.

The behavioral fault models developed in this dissertation only affect the *activation* step of the test generation process. Hence, the high level algorithms developed to handle the *propagation* and *justification* steps still remain valid. Integration of the new behavioral fault models with a behavioral test generation algorithm such as the B-algorithm [21][22] can be of mutual benefit. Such advanced test generation algorithms already address problems such as reconvergent fanout, while use of more complex fault models can eliminate the need to supplement test vector sets via heuristics.

9.1.5 Behavioral Fault Simulation

The new behavioral fault models developed in this dissertation can now be integrated with fault injection techniques such as those developed by DeLong et al. [23][24] to allow fault simulation at higher levels of abstraction. Because the fault models were developed using an external corruption or error vector approach, modifications to the original VHDL behavioral description need only be made once. Individual faults in the compiled VHDL fault simulation model can then be activated by external manipulation of the corrupting signals/error vectors.

The ability to fault simulate VHDL behavioral models rather than more detailed gate level ones will allow better management of ever increasing design complexity. Working with behavioral fault models will also allow fault simulation to be performed earlier in the design scheme, without details of the gate level implementation. In fact, depending on the source of the component, a gate level description may never be available. Thus, these new behavioral fault models facilitate better integration of fault simulation into the overall design process.

9.2 Future Work

The models and techniques presented in this dissertation represent another important step in the development of a design methodology for performing fault simulation throughout the design process. The following sections present a brief description of directions for future research.

9.2.1 Expansion of Behavioral Fault Models

The fault models developed in this dissertation address combinational logic circuits based on the *IEEE Draft Standard for VHDL Register Transfer Level Synthesis* [36]. These behavioral models need to be expanded to include sequential components. The draft standard includes a set of representative design examples whose intent is to specify certain prevalent modeling styles resulting in basic hardware elements like flip-flops, latches, etc. The specification of processes and resulting inferenced logic are detailed in Appendix D.

This dissertation has used the SSL fault model as the basis for its higher level fault models. The generalization and abstraction techniques developed here are not dependent on this choice of a low level fault model. Other low level fault models which complement or improve on the SSL fault model can also be readily applied. Additional fault effects may be abstracted into the behavioral domain, thus improving the overall behavioral fault

models. Fault coverage metrics would have to be adjusted accordingly to effectively represent the low level fault coverage of the improved behavioral fault models.

9.2.2 Tool Development

As previously stated, the behavioral fault models developed in this dissertation can now be integrated into higher level test generation algorithms. The resulting behavioral test generation tool would allow designers to develop test vector sets based on VHDL behavioral descriptions. These behavioral test vector sets could then be used to fault simulate a component at the gate level or even used to test components for which a gate level description is not available.

The new behavioral fault models developed in this dissertation can also be combined with fault injection techniques to allow fault simulation at higher levels of abstraction. A VHDL behavioral fault simulation tool would allow fault experiments to be performed earlier in the design scheme, without details of the gate level implementation. Both behavioral test generation and behavioral fault simulation tools will aid in complexity management and better integrate fault simulation into the overall design process.

9.2.3 Higher Levels of Abstraction

The VHDL descriptions and subsequent fault models in this dissertation cross from the structural into the behavioral domain and move up the design hierarchy from the gate to the register level as defined in Table 1-1. A logical extension to this work is the continuation to higher levels of abstraction such as the chip or system level. The further migration of fault models would clearly support the ultimate goal of developing a design methodology for performing fault simulation throughout the design process.

The design tools used in this dissertation involved synthesis of behavioral data flow descriptions into structural gate level circuits. Moving higher up the design hierarchy next involves algorithmic synthesis tools which translate chip level algorithms into data flow descriptions. Understanding this synthesis process is key to further abstraction of the behavioral fault models.

Ultimately the level of abstraction is reached where the VHDL description in that of a combined hardware/software system. Fault models need to be developed at this system

level so fault simulation and test generation can be integrated into the hardware/software codesign process.

9.3 Concluding Remarks

To cope with the ever increasing complexity of digital circuits, engineers can now work at higher levels of abstraction by taking advantage of computer aided design packages and hardware description languages. Sophisticated synthesis tools provide a design environment which allows the use of higher level VHDL behavioral models. The details of the gate level implementation are safely hidden, shielding the designer from additional complexity. The fault models and abstraction techniques developed in this dissertation represent another important step in integrating fault simulation and testing into such a VHDL synthesis environment.

Expansion of these new behavioral fault models and development of associated computer-aided tools, will allow better management of design complexity. Fault simulation and testing of digital circuits can be moved away from the traditional gate level to join other design aspects at higher levels of abstraction. The end result will be a design methodology which includes performing fault simulation throughout the entire design process.

References

- Abraham, J.A. and V.K. Agarwal, "Test Generation for Digital Systems," Fault-Tolerant Computing: Theory and Techniques, D. K. Pradhan, ed., Prentice-Hall, Englewood Cliffs, NJ, 1985.
- [2] Abraham, J.A. and K. Fuchs, "Fault and Error Models for VLSI," Proceedings of the IEEE, Vol. 74, No. 5, May 1986, pp. 639-654.
- [3] Abramovici, M., M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, Woodland Hills, CA, 1990.
- [4] Al Hayek, G. and C. Robach, "On the Adequacy of Deriving Hardware Test Data from the Behavioral Specification," *Proceedings EUROMICRO 96, 22nd Euromicro Conference*, September 1996, pp. 337-342.
- [5] Al Hayek, G. and C. Robach, A Mutation-based Test for VHDL Descriptions, technical report, LSR-IMAG, 1995.
- [6] Armstrong, J.R., "Chip Level Modeling of LSI Devices," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-3, No. 4, October 1984, pp. 288-297.
- [7] Armstrong, J.R., "Chip Level Modeling with HDL's," *IEEE Design & Test of Computers*, Vol. 5, No. 1, February 1988, pp. 8-18.
- [8] Armstrong, J.R., Chip Level Modeling with VHDL, Prentice Hall, Englewood Cliffs, NJ, August 1989.
- [9] Armstrong, J.R., F.S. Lam, and P.C. Ward, "Test Generation and Fault Simulation for Behavioral Models," *Performance and Fault Modeling with VHDL*, J.M. Schoen, ed., Prentice Hall, Englewood Cliffs, NJ, 1992, pp. 240-303.
- [10] Armstrong, J.R. and F.G. Gray, Structured Logic Design with VHDL, Prentice Hall, Englewood Cliffs, NJ, 1993.
- [11] Ashenden, P.J., *The VHDL Cookbook*, technical report, Department of Computer Science, University of Adelaide, South Australia, 1990.

- [12] Banerjee, P., A Model for Simulating Physical Failures in MOS VLSI Circuits, Technical Report CSG-13, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1985.
- Barclay, D.S. and J.R. Armstrong, "A Heuristic Chip-level Test Generation Algorithm," *Proceedings 23rd Design Automation Conference*, June 1986, pp. 257-262.
- [14] Barclay, D.S., An Automatic Test Generation Method for Chip-Level Circuit Descriptions, master's thesis, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, January 1987.
- [15] Baweja, G., Gate Level Coverage of A Behavioral Test Generator, master's thesis, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, March 1993.
- [16] Bossen, D.C., D.L. Ostapko, and A.M. Patel, "Optimum Test Patterns for Parity Networks," *Proceedings AFIPS Fall Joint Computer Conference*, Vol. 37, November 1970, pp. 33-38.
- [17] Brahme, D. and J.A. Abraham, "Functional Testing of Microprocessors," IEEE Transactions on Computers, Vol C-33, June 1984, pp. 475-485.
- [18] Chakraborty, T. and S. Ghosh, "On Behavior Fault Modeling for Combinational Digital Designs," *Proceedings International Test Conference*, September 1988, pp. 593-600.
- [19] Chao, C.H. and F.G. Gray, "Micro-Operation Perturbations in Chip Level Fault Modeling," Proceedings 25th Design Automation Conference, 1988, pp. 579-582.
- [20] Chen, C.-I.H. and S. Perumal, "Analysis of the Gap between Behavioral and Gate-Level Fault Simulation," *Proceedings Sixth Annual IEEE International ASIC Conference and Exhibit*, September 1993, pp. 144-147.
- [21] Cho, C.H., A Formal Model for Behavioral Test Generation, doctoral dissertation, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, February 1994.

- [22] Cho, C.H. and J.R. Armstrong, "B-algorithm: A Behavioral Test Generation Algorithm," *Proceedings International Test Conference*, 1994, pp. 968-979.
- [23] DeLong, T.A., B.W. Johnson, and J.A. Profeta, III, "A Fault Injection Technique for VHDL Behavioral-Level Models," *IEEE Design & Test of Computers*, Vol. 13, No. 4, Winter 1996, pp. 24-33.
- [24] DeLong, T.A., D.T. Smith, B.W. Johnson, and J.P. Hanna, "Simulator Independent Fault Simulation Using WAVES," *Proceedings Fall '96 VIUF Conference*, October 1996, pp. 129-138.
- [25] ESIP, Level-0 VHDL Synthesis Syntax and Semantics, technical report, Microelectronics Group, University of Cantabria, December 1995.
- [26] Ghosh, S. "Behavior-Level Fault Simulation," *IEEE Design & Test of Computers*, June 1988, pp. 31-42.
- [27] Ghosh, S. and T.J. Chakraborty, "On Behavior Fault Modeling for Digital Designs," Journal of Electronic Testing, Vol. 2, Kluwer Academic Publishers, 1991.
- [28] Gupta, A.K. and J.R. Armstrong, "Functional Fault Modeling and Simulation for VLSI Devices," *Proceedings 22nd Design Automation Conference*, 1985, pp. 720-726.
- [29] Hansen, M.C. and J.P. Hayes, "High-Level Test Generation using Physically-Induced Faults," *Proceedings VLSI Test Symposium*, May 1995, pp. 20-28.
- [30] Hayes, J.P., "On the Realization of Boolean Functions Requiring a Minimal or Near-minimal Number of Tests," *IEEE Transactions on Computers*, Vol. C-20, December 1971, pp. 1506-1513.
- [31] Hayes, J.P., "Modeling Faults in Digital Logic Circuits," *Rational Fault Analysis*,
 R. Saeks and S. R. Liberty, eds., Marcel Dekker, New York, NY, 1977, pp. 78-95.
- [32] Hayes, J.P., "A Fault Simulation Methodology for VLSI," *Proceedings 19th Design Automation Conference*, June 1982, pp. 393-399.

- [33] Hayes, J.P., "Fault Modeling for Digital MOS Integrated Circuits, "IEEE Transactions on Computer-Aided Design, Vol. CAD-3, No. 3, July 1984, pp. 200-207.
- [34] Hayes, J.P., "Fault Modeling," *IEEE Design & Test*, Vol. 2, No. 2, April 1985, pp. 37-44.
- [35] Herzog, J.H., Design and Organization of Computing Structures, Franklin, Beedle & Associates, Wilsonville, OR, 1996.
- [36] IEEE P1076.6/D1.12, Draft Standard for VHDL Register Transfer Level Synthesis, VHDL Synthesis Interoperability Working Group, IEEE, Piscataway, NJ, March 1998.
- [37] IEEE Std 1029.1-1991, IEEE Standard for Waveform and Vector Exchange (WAVES), IEEE, New York, NY, 1991.
- [38] IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual, IEEE, New York, NY, 1988.
- [39] Jenn, E., J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool," *Proceedings 24th International Symposium* on Fault-Tolerant Computing, June 1994, pp. 66-75.
- [40] Johnson, B.W., D.T. Smith, and T.A. DeLong, A Survey of Fault Simulation, Fault Grading, and Test Pattern Generation Techniques with an Emphasis on the Feasibility of VHDL Based Fault Simulation, University of Virginia, Center for Semicustom Integrated Systems, Department of Electrical Engineering, Charlottesville, VA, Report to Rome Laboratory/PKRZ, May 1996.
- [41] Johnson, E.L. and M.A. Karim, Digital Design, A Pragmatic Approach, PWS Engineering, Boston, MA, 1987.
- [42] Johnson, W., "Behavioral-Level Test Development," Proceedings of the 16th Annual Design Automation Conference, 1979, pp. 171-179.
- [43] Kalia, A., Cadence VHDL Synthesizable Subset, technical report, VHDL Synthesis Interoperability Working Group, August 1996.

- [44] Kautz, W.H., "Testing for Faults in Cellular Logic Arrays," *Proceedings 8th Annual Symposium Switching, Automata Theory*, 1967, pp. 161-174.
- [45] Kohavi, Z., *Switching and Finite Automata Theory*, McGraw-Hill, New York, NY, 1978.
- [46] Lam, F.S., Test Generation for Behavioral Models with Reconvergent Fanouts and Feedback, master's thesis, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, September 1989.
- [47] Leonardo Synthesis and Technology Guide, Release 4.0, Exemplar Logic, Inc., Alameda, CA, 1996.
- [48] Levendel, Y.H. and P.R. Menon, "Test Generation Algorithms for Computer Hardware Description Languages," *IEEE Transactions on Computers*, Vol. C-31, pp. 577-589, July 1982.
- [49] Lin, T., Functional Test Generation of Digital LSI/VLSI Systems Using Machine Symbolic Execution Technique, doctoral dissertation, SUNY-Binghamton, Computer Science Department, Binghamton, NY, 1985.
- [50] Lin, T. and S.Y.H. Su, "The S-Algorithm: A Promising Solution for Systematic Functional Test Generation," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-4, July 1985, pp. 250-263.
- [51] McCluskey, E.J., Logic Design Principles with Emphasis on Testable Semi-custom Circuits, Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [52] MIL-STD-883D, Test Methods and Procedures for Microelectronics, Method 5012.1, November 1991.
- [53] Mourad, S. and E.J. McCluskey, "Testability of Parity Checkers," *IEEE Transactions on Industrial Electronics*, Vol. 36, No. 2, May 1989, pp. 254-262.
- [54] Noh, T.H., C.-I.H. Chen, and S.M. Chung, "Behavioral Fault Simulation and ATPG System for VHDL," Proceedings Seventh Annual IEEE International ASIC Conference and Exhibit, 1994, pp. 412-416.

- [55] Norrod, F.E., The E-Algorithm, an Automatic Test Generation Algorithm for Hardware Description Languages, master's thesis, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, February 1988.
- [56] Norrod, F.E., "An Automatic Test Generation Algorithm for Hardware Description Languages," *Proceedings 26th ACM/IEEE Design Automation Conference*, June 1989, pp. 429-434.
- [57] O'Neill, M.D., An Improved Chip-Level Test Generation Algorithm, master's thesis, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, January 1988.
- [58] O'Neill, M.D., D.D. Jani, C.H. Cho, and J.R. Armstrong, "BTG: A Behavioral Test Generator," Proceedings of the IFIP WG 10.2 Ninth International Symposium on Computer Hardware Description Languages and their Applications, June 1989, pp. 347-361.
- [59] Pla, V., J.-F. Santucci, and N. Giambiasi, "On the Modeling and Testing of VHDL Behavioral Descriptions of Sequential Circuits," *Proceedings European Design Automation Conference*, September 1993, pp. 440-445.
- [60] Riesgo, T. and J. Uceda, "A Fault Model for VHDL Descriptions at the Register Transfer Level," *European Design Automation Conference*, September 1996, pp. 462-467.
- [61] Santucci, J.F., A.L. Courbis, and N. Giambiasi, "Behavioral Testing of Digital Circuits," *Journal of Microelectronic Systems Integration*, Vol. 1, No. 1, 1993, pp. 55-77.
- [62] Shen, J.P., W. Maly, and F.J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design & Test*, Vol. 2, December 1985, pp. 13-26.
- [63] Sridhar, T. and J.P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors," *IEEE Transactions on Computers*, Vol. C-30, No. 8, August 1981, pp. 563-571.

- [64] Standard for Synthesizing from VHDL Language at the Register Transfer Level, Synopsys, Mountain View, CA, October 1996.
- [65] Thatte, S.M. and J.A. Abraham, "A Methodology for Functional Level Testing of Microprocessors," Proceedings 8th International Conference on Fault-Tolerant Computing, June 1978, pp. 90-95.
- [66] Thatte, S.M. and J.A. Abraham, "Test Generation for Microprocessors," *IEEE Transactions on Computers*, Vol. C-29, No. 6, June 1980, pp. 429-441.
- [67] The TTL Data Book, 2nd Ed., Texas Instruments, Inc., Dallas, TX, 1976.
- [68] VHDL Style Guide for AutoLogic II, Mentor Graphics Corporation, Wilsonville, OR, 1995.
- [69] Ward, P.C. and J.R. Armstrong, "Behavioral Fault Simulation in VHDL," Proceedings 27th Design Automation Conference, June 1990, pp. 587-593.
- [70] Yount, C.R., The Automatic Generation of Instruction-Level Error Manifestations of Hardware Faults: A New Fault-Injection Model, doctoral dissertation, Carnegie Mellon University, Pittsburgh, PA, May 1993.
- [71] Yount, C.R. and D.P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors," *IEEE Transactions on Computers*, Vol. 45, No. 8, August 1996, pp. 881-891.

Appendix A Additional Examples

In order to more fully examine the effectiveness of the new behavioral fault models, additional examples are provided here. Examples have been chosen to represent a broad range of design possibilities. Multiple synthesis options are employed to ensure the examples are as general as possible.

A.1 Array Indexing

As shown in Appendix E, indexing an array such as a bit_vector also implies a multiplexer architecture. Consider the VHDL behavioral description for example ARRAY4 shown in Figure A-1.

```
entity ARRAY4 is
  port(Y: in BIT_VECTOR(3 downto 0);
        I: in INTEGER range 3 downto 0;
        Z: out BIT);
end ARRAY4;
architecture BEHAVE of ARRAY4 is
begin
  process(Y,I)
      begin
        Z <= Y(I);
      end process;
end BEHAVE;
```

Figure A-1 Behavioral description for example ARRAY4.

The assignment statement $Z \leq Y(I)$ can be equivalently written as a *case* statement, as shown in Figure A-2.

```
case I is
  when 0 => Z <= Y(0);
  when 1 => Z <= Y(1);
  when 2 => Z <= Y(2);
  when 3 => Z <= Y(3);
end case;</pre>
```

Figure A-2 Equivalent case statement for example ARRAY4.

This selection activity implies the applicability of the control fault model developed in Chapter 3. According to the model, each *selection* (clause or index value) can be affected by two different types of faults, *Clause-CORRUPT (OR)* and *Clause-CORRUPT (AND)*. According to a binary encoding for *I*, specified by the synthesis subset, Figure A-3 shows the resulting logical adjacencies for this example.





Thus, applying the control fault model to example ARRAY4 in Figure A-1, results in the behavioral faults shown in Table A-1.

Behavioral Fault	Corrupted Selection	Test Vector (I Y)
Y(0)-CORRUPT (by Y(1))(OR)	Z <= Y(0) OR Y(1)	0 XX10
Y(0)-CORRUPT (by Y(1))(AND)	Z <= Y(0) AND Y(1)	0 XX01
Y(0)-CORRUPT (by Y(2))(OR)	Z <= Y(0) OR Y(2)	0 X1X0
Y(0)-CORRUPT (by Y(2))(AND)	Z <= Y(0) AND Y(2)	0 X0X1
Y(1)-CORRUPT (by Y(0))(OR)	Z <= Y(1) OR Y(0)	1 XX01
Y(1)-CORRUPT (by Y(0))(AND)	Z <= Y(1) AND Y(0)	1 XX10
Y(1)-CORRUPT (by Y(3))(OR)	Z <= Y(1) OR Y(3)	1 1X0X
Y(1)-CORRUPT (by Y(3))(AND)	Z <= Y(1) AND Y(3)	1 0X1X
Y(2)-CORRUPT (by Y(0))(OR)	Z <= Y(2) OR Y(0)	2 X0X1
Y(2)-CORRUPT (by Y(0))(AND)	Z <= Y(2) AND Y(0)	2 X1X0
Y(2)-CORRUPT (by Y(3))(OR)	Z <= Y(2) OR Y(3)	2 10XX
Y(2)-CORRUPT (by Y(3))(AND)	Z <= Y(2) AND Y(3)	2 01XX

 Table A-1 Behavioral faults for example ARRAY4.

Behavioral Fault	Corrupted Selection	Test Vector (I Y)
Y(3)-CORRUPT (by Y(1))(OR)	Z <= Y(3) OR Y(1)	3 0X1X
Y(3)-CORRUPT (by Y(1))(AND)	Z <= Y(3) AND Y(1)	3 1X0X
Y(3)-CORRUPT (by Y(2))(OR)	Z <= Y(3) OR Y(2)	3 01XX
Y(3)-CORRUPT (by Y(2))(AND)	Z <= Y(3) AND Y(2)	3 10XX

 Table A-1 Behavioral faults for example ARRAY4.

Combining don't care values produces the behavioral test vectors shown in WAVES format in Figure A-4.

I	Ż	:	time;	
00	0	:	500 ns;	
00	1	:	500 ns;	
01	0	:	500 ns;	
01	1	:	500 ns;	
10	0	:	500 ns;	
10	1	:	500 ns;	
11	0	:	500 ns;	
11	1	:	500 ns;	
	I 00 01 01 10 10 11 11	I Z 00 0 01 1 01 0 01 1 10 0 10 1 11 0 11 1	I Z : 00 0 : 00 1 : 01 0 : 01 1 : 10 0 : 10 1 : 11 0 :	I Z : time; 00 0 : 500 ns; 00 1 : 500 ns; 01 0 : 500 ns; 01 1 : 500 ns; 10 0 : 500 ns; 10 1 : 500 ns; 11 0 : 500 ns; 11 1 : 500 ns;

Figure A-4 WAVES test vectors for example ARRAY4.

Example ARRAY4 was synthesized to the gate level circuit shown in Figure A-5. Fault simulations were performed using the behavioral test vectors derived from the control fault model resulting in a *SSL* gate level fault coverage of 44/44 = 100%.



Figure A-5 Synthesized circuit for example ARRAY4.

A.2 Generalization of the Control Fault Model

Previous examples have used explicit values of control signals to determine selection in *if-then-else* and *case* statements. When working in the behavioral domain, the specific combination of control inputs may not matter, hence, an enumerated type may be used. The assignment of control signal values to the elements of the enumerated type can be left to the design tools and, thus, delayed to later in the design cycle. The new control fault model can be easily generalized to allow for use of enumerated types.

Example CASE2, in Figure A-6, uses an enumerated type to control the selection of assignments for a *case* statement.

case SEL is	
when J =>	
Z <= Y0;	
when K =>	
Z <= Y1;	
when L =>	
Z <= Y2;	
when M =>	
Z <= Y3;	
end case;	

Figure A-6 Behavioral description for example CASE2.

Due to the independence of the input signals in this example, the synthesis tool produces the standard implementation of the *case* statement as a 4-to-1 multiplexer. Each of the four inputs (Y3, Y2, Y1, Y0) can be assigned to any of the four multiplexer inputs (A, B, C, D), depending on the designation of control bits for the enumerated type SEL. There are, therefore, 4! = 24 possible permutations of the eventual synthesized circuit.

The control fault model can still be applied to the generalized *case* statement in example CASE2, without consideration of the eventual designation of the control bits for the enumerated type SEL. According to the fault model, each clause of a *case* statement is affected by two different types of faults, *Clause-CORRUPT* (*OR*) and *Clause-CORRUPT* (*AND*). The only difference with this example is the determination of logical adjacencies between clauses.

Since no assignment of control values has yet been made for the enumerated type SEL, actual determination of adjacencies among clauses cannot be made. It must, therefore, be

assumed that each clause has the possibility of being adjacent to, and hence corrupted by, any other clause. This assumption implies that each of the four clauses in this example is affected by three WHEN-CORRUPT (OR) faults and three WHEN-CORRUPT (AND) faults. For example, the (OR) corruption of the when K clause results in three behavioral faults: WHEN-K-CORRUPT (by J)(OR), WHEN-K-CORRUPT (by L)(OR), and WHEN-K-CORRUPT (by M)(OR). Application of the control fault model to the generalized case statement from example CASE2, therefore, results in a total of 24 behavioral faults.

Though the application of the control fault model to the generalized *case* statement has resulted in the definition of eight additional behavioral faults, recall also the concept of *compatible fault sets*. The group of faults that corrupt a single channel can form a *compat-ible fault set* and their test vectors can, therefore, be combined. For example, the three *WHEN-CORRUPT (OR)* faults that affect clause *K* lead to the derivation of the following test vectors (*SEL Y3 Y2 Y1 Y0*): KXX01, KX10X, and K1X0X. Though in the actual circuit, only two of the other clauses will be adjacent to clause *K*, the three test vectors are still compatible, forming the test vector K1101. Comparison of this test vector with the corresponding one derived for the $Z \le Y1$ clause for example IF2, (*SEL Y3 Y2 Y1 Y0*) 011X01, shows that the only difference is the elimination of an additional don't care. Alternate assignment of control signals for *K* simply alter the adjacencies between clauses and result in the elimination of a different don't care. A list of *WHEN-CORRUPT (OR)* behavioral faults and their corresponding test vectors is given in Table A-2. A similar list can be easily derived for the 12 *WHEN-CORRUPT (AND)* faults.

Behavioral Fault	Test Vector (SEL Y3 Y2 Y1 Y0)
WHEN-J-CORRUPT (by K)(OR)	
WHEN-J-CORRUPT (by L)(OR)	J 1110 .
WHEN-J-CORRUPT (by M)(OR)	
WHEN-K-CORRUPT (by J)(OR)	
WHEN-K-CORRUPT (by L)(OR)	к 1101
WHEN-K-CORRUPT (by M)(OR)	

Table A-2 Behavioral faults an	l corresponding test vectors	for example CASE2.
--------------------------------	------------------------------	--------------------

Behavioral Fault	Test Vector (SEL Y3 Y2 Y1 Y0)
WHEN-L-CORRUPT (by J)(OR)	
WHEN-L-CORRUPT (by K)(OR)	L 1011
WHEN-L-CORRUPT (by M)(OR)	
WHEN-M-CORRUPT (by J)(OR)	
WHEN-M-CORRUPT (by K)(OR)	M 0111
WHEN-M-CORRUPT (by L)(OR)	

Table A-2 Behavioral faults and corresponding test vectors for example CASE2.

The control fault model has now been generalized to handle enumerated types and the delay of assignment of control signal values. The only change to the model was the inclusion of additional *Clause-CORRUPT* faults due to the assumption that, in the general case, each clause may be adjacent to any other clause. The additional behavioral faults did not, necessarily, result in any additional test vectors due to the concept of *compatible fault sets*.

A.3 Signed Comparison

Example GREATER3 is presented here to demonstrate the application of the behavioral fault models for relational operators to signed as well as unsigned comparisons. As can be seen in Figure A-7, example GREATER3 compares two integers with ranges of -4 to +3. These control signals will be synthesized as 3-bit 2's complement numbers.

```
entity GREATER3 is
  port(A,B: in INTEGER range -4 to +3; GT: out BIT);
end GREATER3;
architecture BEHAVE of GREATER3 is
begin
  process(A,B)
  begin
    if A > B then
      GT <= `1';
    else
      GT <= `0';
    end if;
    end process;
end BEHAVE;
```

Applying the behavioral faults for unsigned *relational operators* from Chapter 4 gives the fault classifications for a signed *GT* function shown in Figure A-8.



Figure A-8 Fault classes for 3-bit signed GT function.

Encoding the test vectors as 3-bit 2's complement numbers results in the WAVES file shown in Figure A-9.

```
% A B GT : time;
% Class I
100 100 0 : 500 ns;
101 101 0 : 500 ns;
110 110 0 : 500 ns;
111 111 0 : 500 ns;
000 000 0 : 500 ns;
001 001 0 : 500 ns;
010 010 0 : 500 ns;
011 011 0 : 500 ns;
111 000 0 : 500 ns;
001 010 0 : 500 ns;
```

% C]	ass	IJ	I		
101	100	1	:	500	ns;
110	101	1	:	500	ns;
111	110	1	:	500	ns;
000	111	1	:	500	ns;
001	000	1	:	500	ns;
010	001	1	:	500	ns;
011	010	1	:	500	ns;

Figure A-9 WAVES test vectors for example GREATER3.

Example GREATER3 was synthesized and optimized to gate level Structure1, shown in Figure A-10, using AutoLogic II. In order to add even more diversity to the problem, an alternate design library from that used for example COMPARE was chosen.



Figure A-10 Synthesized Structure1 for example GREATER3.

According to MIL-STD 883D, Structure1 contains 50 unique SSL gate level faults. Fault simulations using the behavioral test vectors from Figure A-9 resulted in complete gate level fault coverage.

Next, an alternate synthesis tool, Leonardo, was used to map the VHDL behavioral description to a Xilinx FPGA architecture. Fault simulations using the behavioral test vectors achieved a *SSL* gate level fault coverage of 48/48 = 100%.

Lastly, the signed comparison was implemented using *arithmetic operations*. Evaluation of the greater than function can also be performed by subtracting the two operands and examining the sign of the result. For example GREATER3, the most significant bit of the operation B - A forms the output GT. The resulting synthesized and optimized circuit for Structure3 is shown in Figure A-11.

Comparison of the test vectors from Figure A-9 with the truth table for a full-subtractor from Table 5-15 indicates that the behavioral test vectors will provide complete func-



Figure A-11 Synthesized Structure3 for example GREATER3.

tional testing for the subtractor modules used to synthesize Structure3. Fault simulations confirm the complete gate level fault coverage of 41/41 = 100%.

A.4 Unsigned Threshold

Example GE23u is an unsigned threshold comparison using a greater than or equal to (GE) operator. As shown in Figure A-12, A is an integer with range 0 to 31, which will be synthesized to a 5-bit unsigned number.

```
entity GE23u is
   port(A: in INTEGER range 0 to 31;
      GE: out BIT);
end GE23u;
architecture BEHAVE of GE23u is
begin
   process(A)
      begin
      if A >= 23 then
        GE <= `1';
      else
        GE <= `0';
      end if;
   end process;
end BEHAVE;</pre>
```

Figure A-12 Behavioral description for example GE23u.

The *GE* function places the threshold between 22 and 23. According to the behavioral fault model for *threshold operators*, developed in Chapter 4, the threshold lies seven spaces from the center of the range of values, implying step sizes of one, two, and four. Test vectors to the left of the threshold are 22 - 1 = 21, 21 - 2 = 19, and 19 - 4 = 15. Test vectors to the right are 23 + 1 = 24, 24 + 2 = 26, and 26 + 4 = 30. The behavioral test vectors are shown graphically in Figure A-13 and as a WAVES file in Figure A-14.

									-							1
A >= 23 15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	

Figure A-13 Behavioral test vectors for example GE23u.

8 A (ΞE	:	time	∋;
01111	0	:	500	ns;
10001	0	:	.500	ns;
10101	0	:	500	ns;
10110	0	:	500	ns;
10111	1	:	500	ns;
11000	1	:	500	ns;
11010	1	:	500	ns;
11110	1	:	500	ns;

Figure A-14 WAVES test vectors for example GE23u.

Example GE23u was synthesized to the gate level circuit shown in Figure A-15. Fault simulations using the behavioral test vectors from Figure A-14 resulted in a *SSL* gate level fault coverage of 24/24 = 100%.



Figure A-15 Synthesized circuit for example GE23u.

A.5 Adder/Subtractor

Figure A-16 gives the VHDL behavioral description for a 4-bit adder/subtractor circuit, ADDSUB4. The two inputs (A,B) are combined to produce a 4-bit output (D). The operation to be performed is selected by the control signal (OP): OP = '0' selects addition, while OP = '1' selects subtraction.

```
if OP = `0' then
   D <= A + B;
else
   D <= A - B;
end if;</pre>
```

Figure A-16 Behavioral description for example ADDSUB4.

A.5.1 Faults on Arithmetic Operators

Assuming a simple *ripple carry* implementation, the behavioral test vectors for the 4bit addition and 4-bit subtraction can be determined directly from the fault models. The resulting WAVES file is presented in Figure A-17.

8	OP A	В	D	:	time	∋;
ષ્ઠ	ADD					
0	0000	0000	0000	:	500	ns;
0	0000	1111	1111	:	500	ns;
0	1111	0000	1111	:	500	ns;
0	1111	1111	1110	:	500	ns;
0	0001	1111	0000	:	500	ns;
0	1111	0001	0000	:	500	ns;
0	0101	0101	1010	:	500	ns;
0	1010	1010	0100	:	500	ns;
୫	SUB					
1	0000	0000	0000	:	500	ns;
1	0000	1111	0001	:	500	ns;
1	1111	0000	1111	:	500	ns;
1	1111	1111	0000	:	500	ns;
1	0000	0001	1111	:	500	ns;
1	1110	1111	1111	:	500	ns;
1	0101	1010	1011	:	500	ns;
1	1010	0101	0101	:	500	ns;

Figure A-17 WAVES test vectors for example ADDSUB4.

A.5.2 Control Faults

The control fault model specifies that each clause of an *if-then-else* statement can be affected by two different types of faults, *Clause-CORRUPT* (*OR*) and *Clause-CORRUPT* (*AND*). The behavioral test vectors for the *arithmetic operators* must be evaluated to ensure that adequate coverage is provided for these control faults.

For example, the first control fault, *THEN-CORRUPT (OR)*, results in the corrupted version of the *then* clause:

D <= (A + B) OR (A - B)

To test for this fault, the uncorrupted version of the clause (A + B) needs to be set to 0, while the corrupting clause (A - B) is set to 1. The Phase II test vector A = 1111, B = 0001results in A + B = 0000, while A - B = 1110. Since no combination of A, B will produce

- 190

complementary values for D_0 , this test vector should provide sufficient coverage for this control fault. Other control faults and their covering test vectors are summarized in Table A-3. Note that no additional test vectors are required to provide coverage for all control faults.

Control Fault	OP	А	В	A + B	A - B
THEN-CORRUPT (OR)	0	1111	0001	0000	1110
THEN-CORRUPT (AND)	0	0000	1111	1111	0001
ELSE-CORRUPT (OR)	1	1111	1111	1110	0000
ELSE-CORRUPT (AND)	1	0000	0001	0001	1111

Table A-3 Coverage for control faults.

A.5.3 Evaluation of the Behavioral Test Vectors

Example ADDSUB4 was synthesized to the gate level circuit in Figure A-18. Fault simulation using the behavioral test vectors from Figure A-17 results in complete gate level fault coverage of 124/124 = 100%.



Figure A-18 Synthesized circuit for example ADDSUB4.

A.5.4 CLA Implementation

If the target technology includes *CLA* circuits, additional test vectors are required to cover the behavioral faults. The vectors for the addition operation are those derived in Chapter 5 and summarized in Table 5-10. Behavioral test vectors for the subtraction operation can be derived using 2's complement arithmetic.

The highest order carry in a 4-bit subtractor implemented with a CLA adder is C_2 . From Table 5-7, the terms for C_2 , including a *carry-in*, are $P_2P_1P_0C_{-1}$, $P_2P_1G_0$, P_2G_1 , and G_2 . Behavioral test vectors for *missing carry* and *extra carry* faults can be derived for the subtraction operation by converting the corresponding addition to subtraction.

According to the new behavioral fault model, test vectors are necessary for *missing* carry faults for all but the lowest and highest order terms. *Missing carry* behavioral faults are presented in Table A-4. The left hand side of the table gives the test vector for addition, while the right hand side shows the conversion to subtraction.

			Add	ition		Subtraction			
Stage	Term	А	В	CIN	S	М	S	D	
0	$P_2P_1G_0$	0001	1111	1	0001	0001	0000	0001	
1	P_2G_1	0010	1111	1	0010	0010	0000	0010	

Table A-4 Missing carry faults for subtraction.

Test vectors for *extra carry* faults are required for all but the two most significant stages. Behavioral test vectors for the *extra carry* faults are presented in Table A-5. The *CLA* test vectors for example ADDSUB4 are, therefore, presented in Figure A-19.

			Add	ition	Subtraction				
Stage	Term	Α	В	CIN	S	М	S	D	
		0000	1110	1	1111	0000	0001	1111	
-1	P ₂ P ₁ P ₀ C ₋₁	0000	1101	1	1,110	0000	0010	1110	
		0000	1011	1	1100	0000	0100	1100	
0	P ₂ P ₁ G ₀	0001	1101	1	1111	0001	0010	1111	
		0001	1011	1	1101	0001	0100	1101	

Table A-5 Extra carry faults for subtraction.

A CLA implementation of example ADDSUB4 is presented in Figure A-20 [41]. Fault simulations were performed using the *ripple carry* test vectors from Figure A-17, plus the CLA test vectors from Figure A-19. The *ripple carry* vectors alone produced a SSL gate level fault coverage of 203/209 = 97.13%. The remaining faults were then detected by the CLA test vectors, resulting in complete gate level fault coverage.

ક	OP A	В	D	:	time	∋;
8	ADD (CLA				
0	0001	0011	0100	:	500	ns;
0	0001	0101	0110	:	500	ns;
0	0010	0110	1000	:	500	ns;
ક	SUB (CLA				
1	0001	0000	0001	:	500	ns;
1	0010	0000	0010	:	500	ns;
1	0000	0001	1111	:	500	ns;
1	0000	0010	1110	:	500	ns;
1	0000	0100	1100	:	500	ns;
1	0001	0010	1111	:	500	ns;
1	0001	0100	1101	:	500	ns;

Figure A-19 CLA test vectors for example ADDSUB4.



Figure A-20 CLA implementation of example ADDSUB4.

A.6 Arithmetic with Constants

The behavioral test vector patterns developed in Chapter 5 will now be applied to two larger examples to demonstrate the effect of synthesis optimizations on gate level fault coverage. First, example PLUS25 adds the *constant 25* to an 8-bit number. Next, example MINUS25 combines the *subtraction operator* with the same *constant*. Alternately, this example can be viewed as *addition* using the *constant -25*.

. 193
A.6.1 Example PLUS25

Example PLUS25 implements the arithmetic operation $Z \le Y + 25$. Signals Y and Z are declared with ranges from -128 to +127, hence they will be synthesized as 8-bit 2's complement numbers. The binary representation for +25 is 00011001, thus the Phase II test patterns are formed starting with the vector 11100110. The resulting behavioral test vectors are shown in Figure A-21.

% Y Z	: time;				
00000000	00011001	:	500	ns;	
11111111	00011000	:	500	ns;	
ક					
11100110	11111111	:	500	ns;	
11001101	11100110	:	500	ns;	
10011011	10110100	:	500	ns;	
00110111	01010000	:	500	ns;	
01101110	10000111	:	500	ns;	
11011100	11110101	:	500	ns;	
10111001	11010010	:	500	ns;	
01110011	10001100	:	500	ns;	

Figure A-21 Behavioral test vectors for example PLUS25.

Example PLUS25 was synthesized and optimized to the gate level Structure1 shown in Figure A-22. Note that the optimization process has altered the original *ripple carry* structure of the 8-bit adder.



Figure A-22 Synthesized Structure1 for example PLUS25.

Fault simulations using the behavioral test vectors from Figure A-21 resulted in a SSL gate level fault coverage of 114/115 = 99.13%. The functional testing approach is still

able to achieve near complete fault coverage. An alternate synthesis tool and target architecture was next used to produce Structure2 for example PLUS25. Fault simulations using the behavioral test vectors from Figure A-21 resulted in a *SSL* gate level fault coverage of 190/190 = 100%.

A.6.2 Example MINUS25

Example MINUS25 performs the arithmetic operation $Z \le Y - 25$. Again, the binary representation for +25 is 00011001, resulting in the behavioral test vectors and associated functional tests shown in Table A-6.

			Test #						
Phase	Y	Z	ST ₇	ST ₆	ST ₅	ST ₄	ST ₃	ST ₂	ST ₁
т	00000000	11100111	0	0	0	0	0	0	0
	11111111	11100110	3	3	3	3	3	3	3
	00011001	00000000	1	1	1	3	3	1	1
	00110010	00011001	1	1	2	2	1	1	2
	01100100	01001011	1	3	2	0	1	2	0
	11001000	10101111	3	2	Ö	0	2	0	0
11	10010001	01111000	2	0	0	2	1	1	1
	00100011	00001010	1	1	2	0	1	1	3
	01000110	00101101	1	2	0	0	1	3	2
	10001100	01110011	2.	0	0	1	3	2	0

 Table A-6 Functional tests for example MINUS25.

Example MINUS25 was synthesized and optimized using the same process as Structure1 for example PLUS25. Fault simulations using the behavioral test vectors from Table A-6 again resulted in a SSL gate level fault coverage of 114/115 = 99.13%.

A.7 XOR4

Using the behavioral fault models developed in Chapter 6, test vectors can be developed for example XOR4 shown below.

Z <= A XOR B XOR C XOR D

195

First, the parse tree method from Section 6.1.1.2 will identify the test vectors necessary for exhaustive testing of each *XOR operator*. Next, the modified Bossen algorithm from Section 6.1.2.2 will produce an optimized set of test vectors for the given *expression*.

A.7.1 Parse Tree Test Vectors

A parse tree for example XOR4 is shown in Figure A-23. The nodes (1,2,3) are formed by the XOR operators, while the leaves of the tree are the signals A, B, C, and D.



Figure A-23 Parse tree for example XOR4.

According to the generalized set of functional faults from Section 6.1.2.1, an exhaustive set of four tests is necessary for each *XOR operator*. Applying these tests to the parse tree from Figure A-23 produces the behavioral test vectors shown in Table A-7.

Node	Behavioral Test	Test Requirements	Test Vector (ABCD)
	00	A = 0, B XOR C XOR D = 0	0000
1	01	A = 0, B XOR C XOR D = 1	0100
	10	A = 1, B XOR C XOR D = 0	1000
	11	A = 1, B XOR C XOR D = 1	1100
	00	B = 0, C XOR D = 0, A = 0	0000
	01	B = 0, C XOR D = 1, A = 0	0010
2	10	B = 1, C XOR D = 0, A = 0	0100
	11	B = 1, C XOR D = 1, A = 0	0110

Table A-7 Behavioral test vectors for example XOR4.

Node	Behavioral Test	Test Requirements	Test Vector (ABCD)
	00	C = 0, D = 0, B = 0, A = 0	0000
2	01	C = 0, D = 1, B = 0, A = 0	0001
3	10	C = 1, D = 0, B = 0, A = 0	0010
	11	C = 1, D = 1, B = 0, A = 0	0011

 Table A-7 Behavioral test vectors for example XOR4.

Eliminating redundant test vectors produces the WAVES file shown in Figure A-24.

% ABC	CD	Z	: t:	ime ;
0000	0	:	500	ns;
0001	1	:	500	ns;
0010	1	:	500	ns;
0011	0	:	500	ns;
0100	1	:	500	ns;
0110	0	:	500	ns;
1000	1	:	500	ns;
1100	0	:	500	ns;

Figure A-24 WAVES test vectors for example XOR4.

A.7.2 Evaluation of Behavioral Test Vectors

Example XOR4 was synthesized into multiple gate level realizations to evaluate the behavioral test vectors. Varying structures, synthesis tools, and design libraries were employed to produce a broad range of implementations.

Structure1 of example XOR4 is shown in Figure A-25. Fault simulations using the behavioral test vectors from Figure A-24 result in a *SSL* fault coverage of 24/24 = 100%.



Figure A-25 Structure1 for example XOR4.

Structure 2 for example XOR4 is shown in Figure A-26. Fault simulations with the behavioral test vectors also produce a *SSL* gate level fault coverage of 24/24 = 100%.



Figure A-26 Structure2 for example XOR4.

An alternate synthesis tool and target architecture were used to produce Structure3 for example XOR4, shown in Figure A-27. Fault simulations using the same behavioral test vectors from Figure A-24 result in a *SSL* gate level fault coverage of 54/54 = 100%.



Figure A-27 Structure3 for example XOR4.

Lastly, Structure4 for example XOR4 is presented in Figure A-28. Again, fault simulations using the behavioral test vectors result in complete gate level fault coverage.



Figure A-28 Structure4 for example XOR4.

A.7.3 Optimized Test Vectors

Applying the modified Bossen algorithm to example XOR4 results in the labelling shown in Figure A-29 and Figure A-30. Note, the 4-bit example represents a special case where it is possible to find identical test sequences for structures Cascade1 and Cascade2.



Figure A-29 Structure Cascade1 for example XOR4.



Figure A-30 Structure Cascade2 for example XOR4.

The resulting optimized test vectors are shown in Table A-8.

Signal	Label	Sequence
Α	S	0011
В	R	0110
С	R	0110
D	Т	0101

Test Vector (ABCD)	Z
0000	0
0111	1
1110	1
1001	0

Table A-8 Optimized test vectors for example XOR4.

A.7.4 Evaluation of Optimized Test Vectors

Fault simulations were conducted using the optimized behavioral test vectors from Table A-8 and Structures1-4 for example XOR4. For all four implementations of example XOR4, complete gate level fault coverage was achieved using either test vector set.

Appendix B Fault Experiment Results

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
CASE1	Structure1	34	8	100%
CASEI	Structure2	34	8	100%
	Structure1	44	8	100%
AKKA14	Structure2	44	8	100%
SUIETA.,	Structure1	90	8	100%
ЭПІГ I 4 0	Structure2	112	8	100%

Table B-1Control fault experiments.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
I ESSO	Structure1	29	8	100%
LESS2	Structure2	30	8	100%
	Structure1	31	8	100%
EQUALS	Structure2	31	8	100%
	Structure1	50	18	100%
GREATER3	Structure2	48	18	100%
	Structure3	41	18	100%
1 25	Structure1	18	6	100%
LES	Structure2	20	6	100%
CE92m	Structure 1	24	8	100%
GE23u	Structure2	22	8	100%
LT12u	Structure	14	4	100%
GT3n	Structure	17	5	100%

 Table B-2 Relational operator fault experiments.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
	Otan aturn 1	74	12	98.65%
	Structure1	74	12	98.65%
COMPARE	Star store 2	70	12	100%
	Structurez	12	12	100%
	0	07	22	97.94%
	Structure1	97	22	97.94%
COMPARE3	Street 2	02	22	100%
	Structure2	92	22	100%
	Stracture	150	12	100%
COMPARE4	Structure	150	12	100%
	Structure	178	22	100%
COMPARE 34	Structure	178	22	100%

.

Table B-2 Relational operator fault experiments.

Example	Example Implementation		Behavioral Test Vectors	Fault Coverage
	Structure1	102	8	100%
	Streetward 2	120	8	97.69%
ADD4	Structure2	150	11	100%
	Structure3	142	8	100%
	Structure 1	138	8	100%
		100	8	88.30%
ADD4wc	Structure2	100	20	100%
	Structure3	194	8	100%

Table B-3 Arit	thmetic operator	fault	experiments.
----------------	------------------	-------	--------------

201

.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage	
	Structure1	234	8	100%	
			8	93.55%	
ADD8	Structure2	310	33	93.55% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100%	
			22	100%	
	Structure1	112	8	100%	
SUB4		157	8	94.90%	
	Structure2	157	157 15		
	Structure1	124	16	100%	
ADDSUB4	Structure2	209	26	100%	
	Structure1	50	6	100%	
INC4	Structure2	80	6	100%	
INC8	Structure	114	10	100%	
DEC4	Structure	51	6	100%	
ADDINC4	Structure	108	14	100%	
PLUS3	Structure	68	7	100%	
MINUS5	Structure	85	8	100%	
DI 11025	Structure1	115	10	99.13%	
PLUS25	Structure2	190	10	100%	
MINUS25	Structure · .	115	10	99.13%	

•

۰.

 Table B-3 Arithmetic operator fault experiments.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
SOP1	Structure1	18	5	100%
	Structure2	18	. 5	100%
SOP4	Structure 1	72	5	100%
	Structure2	72	5	100%

Table B-4 Other operator fault experiments.

202

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage	
DOG1	Structure1	18	5	100%	
POSI	Structure2	18	5	100%	
	Structure1	30	- 8	100%	
GT	Structure2	35	8	100%	
	Structure3	29	8	100%	
	Star stress 1	24	8	100%	
	Structure1	24	4	100%	
	Stars at small	24 8 100 24 4 100 54 8 100			
XOD (Structure2	24	4	100%	
XOR4	Cture attaine 2	54	100%		
	Structures	54	100%		
	Ci	4.4	8	100%	
	Structure4	44	100%		
	Structure 1	30	7	100%	
VODS	Structure2	30	7	100%	
XOK5	Structure3	58	7	100%	
	Structure4	58	7	100%	
	Structure1	44	5	100%	
AB54	Structure2	44	5	100%	
4 D C 9	Structure1	132	9	100%	
AB20	Structure2	126	9	100%	
	Structure1	49	5	100%	
NEU4	Structure2	50	5	100%	
	Structure1	114	9	100%	
NEG8	Structure2	204	9	99.02%	

.

.

 Table B-4 Other operator fault experiments.

.

Example	Implementation	SSL Faults	Behavioral Test Vectors	Fault Coverage
	Structure1	284	31	100%
ALU4wc	Structure2	312	31	100%
	Structure3	398	31	100%
ALU8wc	Structure1	546	31	100%
	Structure2	758	31	100%
HAMMING4	Structure1	114	17	100%
	Structure2	116	17	100%
	Structure3	226	17	100%
HAMMING8	Structure1	226	33	100%
	Structure2	446	33	100%

 Table B-5 Comprehensive fault experiments.

•

,

Appendix C VHDL Behavioral Descriptions

VHDL can be used to model the function performed by a module at a level of abstraction above the gate level. Such a description is called a functional or behavioral description. The VHDL Cookbook [11] and the IEEE Standard VHDL Language Reference Manual [38] serve as two key resources for designing hardware using behavioral descriptions. VHDL allows description of behavior in the form of a sequence of familiar programming language constructs. Behavioral descriptions use variables and signals along with their corresponding assignment statements to model the desired functionality of the ultimate hardware. Expressions perform arithmetic or logical computations by applying an operator to one or more operands. Constructs used to control the selection and sequencing of instructions include *if*, case, and loop statements. Concurrency of execution in hardware is modeled using a process statement. Like other programming languages, VHDL provides subprogram facilities in the form of procedures and functions.

C.1 Variables and Signals

An object is a named item in a VHDL description which has a value of a specified type. A *variable* is an object whose value may be changed at any time during the simulation of the circuit. It is local to a process or subprogram and has a single current value. A *signal* is an object with a value that is changed only at scheduled times. Signals represent electrical quantities that can be used to transmit information and are normally used to connect submodules in a design. Each object has a corresponding *assignment statement*.

As in other programming languages, a *variable* is given a new value using an assignment statement. In the simplest case, the target of the assignment is an object name and the value of the expression is given to the named object. *Variable* assignments occur immediately when the assignment statement is executed and are local to a *process* or subprogram. A *signal* assignment schedules a transaction to a signal. The target must represent a *signal* or be an aggregate of *signals*. Scheduled transactions are executed as simulation time progresses. *Signals* are global in a *process* or subprogram and are the only means of communication between *processes*.

205

C.2 Expressions

An *expression* is a formula that defines the computation of a value. The type of an *expression* depends only upon the types of its *operands* and on the *operators* applied. The predefined VHDL *operators* are listed in Table C-1.

Туре	Operators					
Logical	AND	OR	NAND	NOR	XOR	
Relational	=	/=	<	<=	>	>=
Adding	+	-	&			
Unary (sign)	+	-				
Multiplying	*	/	MOD	REM		
Miscellaneous	**	ABS	NOT			

Table C-1 Predefined VHDL operators.

Relational *operators* must have both *operands* of the same type and yield Boolean results. The equality *operators* (= and /=) can have *operands* of any type. The remaining *operators* must have *operands* which are scalar types or one-dimensional arrays of discrete types.

The sign *operators* (+ and -) and the addition (+) and subtraction (-) operators have their usual meaning on numeric *operands*. The concatenation *operator* (&) operates on one-dimensional arrays to form a new array with the contents of the right *operand* following the contents of the left *operand*. It can also concatenate a single new element to an array or two individual elements to form an array.

The multiplication (*) and division (/) operators work on integer and floating point types. The modulus (MOD) and remainder (REM) operators only work on integer types. The absolute value (ABS) operator works on any numeric type. Finally, the exponentiation (**) operator can have an integer or floating point left operand, but must have an integer right operand.

C.3 If Statement

The *if* statement allows selection of statements to execute depending on one or more conditions. The syntax is:

```
if_statement ::=
    if condition then
        sequence_of_statements
    { elsif condition then
        sequence_of_statements }
    [ else
        sequence_of_statements ]
    end if;
```

The conditions are *expressions* resulting in Boolean values. The conditions are evaluated successively until one is found that yields the value *true*. In that case the corresponding statement list is executed. Otherwise, if the *else* clause is present, its statement list is executed.

C.4 Case Statement

The *case* statement allows selection of statements to execute depending on the value of a selection *expression*. The syntax is:

```
case_statement ::=
    case expression is
        case_statement_alternative
        { case_statement_alternative }
    end case;

case_statement_alternative ::=
    when choices =>
        sequence_of_statements
choices ::= choice { | choice }

choice ::=
    simple_expression
    | discrete_range
    | element_simple_name
    | others
```

The selection *expression* must result in either a discrete type, or a one-dimensional array of characters. The alternative whose *choice* list includes the value of the *expression* is selected and the statement list executed. Note that all the *choices* must be distinct, that is, no value may be duplicated. Furthermore, all values must be represented in the *choice*

lists, or the special choice others must be included as the last alternative. If no choice list includes the value of the expression, the others alternative is selected.

Some examples of case statements [11]:

```
case opcode is
  when X"00" => perform_add;
  when X"01" => perform_subtract;
  when others => signal_illegal_opcode;
end case;
case element_color is
  when red =>
    statements for red;
  when green | blue =>
    statements for green or blue
  when orange to turquoise =>
    statements for these colors;
end case;
```

C.5 Loop Statements

VHDL has a basic *loop* statement, which can be augmented to form the usual *while* and *for* loops seen in other programming languages. The *while* iteration scheme allows a test condition to be evaluated before each iteration. The iteration only proceeds if the test evaluates to *true*. If the test is *false*, the *loop* statement terminates. An example [11]:

```
while index < length loop
    index := index + 1;
end loop;
```

The *for* iteration scheme allows a specified number of iterations. The *loop* parameter specification declares an object which takes on successive values from the given range for each iteration of the *loop*. Within the statements enclosed in the *loop*, the object is treated as a constant, and so may not be assigned to. An example [11]:

```
for item in 1 to last_item loop
   table(item) := 0;
end loop;
```

There are two additional statements which can be used inside a *loop* to modify the basic pattern of iteration. The *next* statement terminates execution of the current iteration

and starts the subsequent iteration. The *exit* statement terminates execution of the current iteration and terminates the *loop*. The syntax of these statements is:

next_statement ::= next [loop_label] [when condition];

exit_statement ::= exit [loop_label] [when condition];
If the loop label is omitted, the statement applies to the inner-most enclosing loop, otherwise it applies to the named loop. If the when clause is present but the condition is false,
the iteration continues normally.

C.6 Process Statement

The primary unit of behavioral description in VHDL is the *process*. A *process* is a sequential body of code which can be activated in response to changes in state specified by a sensitivity list or a *wait* statement. When more than one *process* is activated at the same time, they execute concurrently.

An example of a process statement with a sensitivity list [11]:

```
process (reset, clock)
  variable state : bit := false;
begin
  if reset then
    state := false;
  elsif clock = true then
    state := not state;
  end if;
  q <= state after prop_delay;
  -- implicit wait on reset, clock
end process;</pre>
```

During the initialization phase of simulation, the *process* is activated and assigns the initial value of state to the signal q. It then suspends at the implicit *wait* statement indicated in the comment. When either reset or clock change value, the *process* is resumed and execution repeats from the beginning.

Processes, like all other concurrent statements, read and write *signals* and interface port values to communicate with the rest of the architecture. They are unique in that they behave like concurrent statements to the rest of the design, but they are internally sequential. In addition, only *processes* define *variables* to hold intermediate values in a sequence of computations.

C.7 Procedures and Functions

Subprograms, like *processes*, use sequential statements to define algorithms that compute values. Unlike *processes*, however, they cannot directly read or write *signals* from the rest of the architecture. All communication is performed through the subprogram's interface; each subprogram call has its own set of interface *signals*.

The two types of subprograms are *functions* and *procedures*. A *function* returns a single value directly. A *procedure* returns zero or more values through its interface. Subprograms may perform repeated calculations, often in different parts of an architecture. The syntax is:

```
subprogram_specification ::=
    procedure designator [(parameter_list)] |
    function designator [(parameter_list)] return type
```

By using these programming language constructs, designers can use VHDL to develop behavioral level models that can be simulated to verify their correct functioning, prior to generating hardware.

Appendix D VHDL Synthesis

Automated design tools are available which allow the designer to synthesize and optimize circuit descriptions expressed through hardware description languages such as VHDL. In order to ensure proper and consistent synthesis of VHDL language constructs to hardware, such tools include guidelines which describe both the syntax and semantics of the subset of the entire VHDL language which they support. The VHDL Synthesis Interoperability Working Group (SIWG) was established to develop a public domain VHDL Synthesis Interoperability Standard which, if used by designers, will allow VHDL synthesis models to be portable across synthesis vendors that support this standard. Their synthesis domain is register transfer level (RTL) logic synthesis which corresponds to the functional/behavioral descriptions modeled in this and previous research.

A draft IEEE standard [36] has been developed which encompasses recommendations drawn from several key sources, including Cadence [43], Mentor Graphics [68], and Synopsys [64]. The foundation for the synthesis standard was laid by the European VHDL Synthesis Working Group's *Level-0 VHDL Synthesis Syntax and Semantics* [25]. Beyond the baseline Level-0, the draft proposes standardization Level 1 containing constructs that are currently supported by many synthesis tools. The standard also attempts to describe VHDL constructs as:

- 1) Supported: RTL synthesis will map the construct to hardware.
- 2) Ignored: RTL synthesis will ignore the construct. Encountering the construct will not cause synthesis to fail, but synthesis results may not match simulation results.
- 3) Not supported: RTL synthesis does not support the construct.

A synthesis tool is defined as any system, process, or tool that interprets register transfer level VHDL source code as a description of an electronic circuit in accordance with the terms of this standard and derives a gate level netlist description of that circuit.

D.1 Level-0

The Level-0 VHDL Synthesis Syntax and Semantics [25] represents a first step towards a standard for VHDL allowing its use in other hardware related tasks like formal verification, fault simulation, test generation, etc. which currently can not be performed in a standard way. The intention behind the definition of Level-0 was to overcome numerous problems currently associated with high level synthesis. The Level-0 syntax and semantics constitute a standard subset of VHDL for synthesis applications which will allow description portability between tools as well as design reusability.

Level-0 represents the minimum syntactical and semantical requirements to any synthesis tool in the market. Of the 217 syntax elements described in Appendix A of the VHDL Language Reference Manual [38]:

- 106 syntax elements are fully supported.

- 53 syntax elements are supported with restrictions.

- 56 syntax elements are not allowed.

- 2 syntax elements are ignored.

Despite the current limitations imposed by the state of the art in commercial synthesis technology, the Level-0 synthesis syntax and semantics allows the description of digital systems as an interconnection of combinational and sequential blocks and, therefore, allows the description of any digital system at the RT level (i.e. algorithmic finite state machines). As a consequence, any synchronous VHDL description in any proprietary VHDL subset can be translated to the Level-0 synthesis syntax and semantics maintaining all the relevant information about the functionality of the design. This portable description will be accepted by any other synthesis tool giving functional equivalent results.

In addition to the support of specific language constructs, Level-0 also defines certain synthesis semantics and usage guidelines. Probably the most important of these design guidelines are those for processes, which serve as the basis for behavioral descriptions. Processes must have a set of special characteristics, which can be summarized in four different kinds of processes shown in Figure D-1.

Other restrictions imposed by Level-0 include the limited set of operators allowed. The operators: *abs*, "**", "/", *mod*, and *rem* are not supported. The multiplying operator "*" is supported, only if both operands are constants or the second operand is a power of two. Additionally, floating point operands are not allowed and the only type of loops allowed are *for* loops.

- Processes which contain a sensitivity list including all the signals which are read into the process and in which all signals and variables are assigned in all the conditional branches. This kind of process models pure combinational logic.
- 2) Processes which contain a sensitivity list including all the signals that are read into the process and whose variables are assigned in all the conditional branches of the process. This kind of process can model a mixture of pure combinational logic and asynchronous latches. Latches are inferred when signals are not assigned in a conditional branch.
- 3) Processes which have, as their first statement, a wait statement in the form:

wait until clock = value and clock'event;

This kind of process models a Moore synchronous sequential machine.

4) Processes which have a sensitivity list including the *clock* signal and optionally an asynchronous *reset* signal and an *if* statement controlled by the event and edge of the clock signal. Thus, this kind of process has the following syntax:

```
process (clk_name, reset_name)
  process_declarative_part
begin
  if (reset_name = value) then
      { signal_assignment_statement }
    elsif (clk_name = value and clk_name'event) then
      { signal_assignment_statement }
    end if;
end process;
```

Figure D-1 Process types for Level-0.

D.2 Mentor Graphics

The VHDL Style Guide for AutoLogic II [68] describes how to write synthesizable VHDL for the Mentor Graphics AutoLogic II synthesis environment. The guidelines encompass language restrictions, style issues, modeling methods, and design methods. Section 1 of the manual summarizes the subset of VHDL language supported for synthesis. All of the language elements previously discussed in Appendix C of this dissertation are supported by AutoLogic II. In addition to the predefined VHDL operators listed in

Table C-1, shift operators from VHDL 1993 are also supported. These include: shift left/right arithmetic (sla/sra), shift left/right logical (sll/srl), and rotate left/right (rol/ror).

AutoLogic II also supports the Mentor Graphics' std_logic_arith package, defined in the Mentor Graphics' ARITHMETIC library. This package includes functions that allow signed and unsigned math to be performed using the STD_LOGIC and STD_LOGIC_VECTOR types. The functions provide for binary and unary arithmetic operations beyond the numeric operands predefined by VHDL. The package also defines the SIGNED and UNSIGNED types that are used directly with arithmetic and relational shifts, comparison, logical, and type conversion functions. Relational operators are supplemented by the comparison functions: equal (eq), not equal (ne), less than (lt), greater than (gt), greater than or equal to (ge), and less than or equal to (le).

The remainder of the manual presents the VHDL constructs and modeling styles necessary to synthesize varying types of hardware. A generous set of examples provides the designer with a virtual cookbook on how to design specific circuits. Section 2 presents constructs and style issues describing hardware functionality implemented with a combinatorial architecture. Topics include signal assignments and sequential statements with special emphasis given to the control constructs *if* and *case*. Section 3 discusses how AutoLogic II recognizes clocked sections of descriptions and defines edge sensitive and level sensitive latch inferencing. The guidelines for combinational and sequential processes agree with those discussed in Figure D-1 as part of the Level-0 synthesis semantics.

Section 4 of the manual introduces the use of predefined operators and functions, data types, and procedures used in describing hardware with VHDL. Descriptions and examples give details for use of the predefined modeling environment, as well as specifics on the types of hardware to be synthesized. Finally, Section 5 presents the guidelines for developing and synthesizing synchronous state machine descriptions. Numerous examples describe multiple modeling styles for both Mealy and Moore class state machines.

D.3 Synopsys

Input to the VHDL Synthesis Interoperability Working Group was provided by Synopsys in the form of a *Standard for Synthesizing from VHDL Language at the Register Transfer Level* [64], which now takes the form of an unapproved IEEE Standards Draft. This standard describes the use of a synthesis tool to translate high-level VHDL descriptions to gate-level netlists. The purpose of the standard is to define how a synthesis tool shall behave when synthesizing from VHDL at the register transfer level. Clauses 3 through 7 describe the language constructs, data types, expressions, sequential statements, and concurrent statements used for writing VHDL design descriptions. The standard is really just a language reference manual, with examples, for the subset of VHDL that is applicable to synthesis.

Many VHDL constructs, although useful for simulation and other stages in the design process, are not relevant to synthesis. Because these constructs cannot be synthesized, they are not supported by the synthesis tool. Clause 8 provides a list of all VHDL constructs with the level of support for each. A construct may be fully supported, ignored, or unsupported. Ignored means that the construct is allowed in the VHDL source, but is ignored by the synthesis tool. Unsupported means that the construct is not allowed in the VHDL source and that the synthesis tool flags it as an error.

Some of the key design restrictions imposed by the synthesis tool are summarized in Figure D-2. Though these restrictions give very specific guidance to the designer regard-

- Integer types are automatically converted to bit vectors whose width is as small as possible to accommodate all possible values of the type's range, either in unsigned binary for nonnegative ranges or in 2's complement form for ranges that include negative numbers.
- 2) Floating-point types, such as *REAL*, are unsupported.
- The arithmetic operators "+" and "-" are predefined for all integer operands.
 For adders more than four bits wide, a synthetic library component is used.
- 4) Multiplying operators ("*", "/", mod, and rem) are predefined for all integer types with the following restrictions:

a) "*" (Integer multiplication) - No restrictions. A multiplication operator is implemented as a synthetic library cell.

b) "/" (Integer division) - The right operand shall be a power of 2.

Neither operand shall be negative. Implemented as a bit shift.

Figure D-2 Design restrictions for Synopsys

- 5) Some forms of the *if* statement may be used to test for signal edges and, therefore, imply synchronous logic. This usage causes the synthesis tool to infer registers or latches.
- 6) Synthesized hardware is sensitive to all signals read by a process. To guarantee that a VHDL simulator distinguishes the same results as the synthesized hardware, a process sensitivity list shall contain all signals whose changes require resimulation of that process.

Figure D-2 Design restrictions for Synopsys

ing use of the language, the details of the implementation by the synthesis tool are hidden. This lack of synthesis information makes the Synopsys tool less useful than Mentor Graphics' tool for exploring the relationship of language constructs with resulting hardware for the development of higher level fault models.

D.4 IEEE Draft Standard

The resulting *IEEE Draft Standard for VHDL Register Transfer Level Synthesis* [36] builds on Level-0 and incorporates constructs common to synthesis tools by Mentor Graphics and Synopsys. The draft standard represents a subset of VHDL with corresponding design guidelines meant to ensure consistent synthesis of gate level netlists from behavioral descriptions.

As with the other synthesis environments discussed in this dissertation, a syntax subset of VHDL is defined. The key language constructs supported for behavioral modeling are listed below:

- 1) *if* statement, *case* statement, *loop* statement.
- 2) procedure, function.
- 3) constant, variable, signal.
- 4) all predefined VHDL operators shown in Table C-1.

Design restrictions are consistent with those discussed for Synopsys in Figure D-2. The only iteration scheme supported for the *loop* statement is *for*.

The draft standard also contains a set of representative design examples whose intent is to specify certain prevalent modeling styles resulting in basic hardware elements like flip-flops, latches, etc. The specification of processes and resulting inferenced logic are consistent with the description of combinational and sequential logic for Level-0 shown in Figure D-1.

Use of this subset is meant enhance the portability of VHDL designs across synthesis tools conforming to the standard. It should also minimize the potential of functional simulation mismatches between models before they are synthesized and models after they are synthesized. It, therefore, should also serve as the basis for defining higher level fault models which have a closer relationship to resulting synthesized hardware.

Appendix E

Hardware Implementation of VHDL Constructs

Several VHDL language constructs lend themselves directly to hardware implementation with common functional modules such as multiplexers. By examining these language to hardware relationships, this thesis intends to build the foundation on which higher level fault models can be defined, that are more closely related to their underlying gate level counterparts. The necessary insights will be drawn from two resources which directly discuss the relationship between certain VHDL constructs and the ultimate hardware.

One discussion of hardware implementation of VHDL constructs comes from *Structured Logic Design with VHDL* by Armstrong and Gray [10]. In a section titled "Automated Synthesis of VHDL Constructs," they demonstrate the relationship between multiplexers and language constructs that involve selection, like *if* and *case*. Another insight into the relationship between VHDL language constructs and hardware comes from the *VHDL Style Guide for AutoLogic II* by Mentor Graphics [68]. Again, the link is established between the control constructs *if* and *case* and the multiplexer functional building block.

E.1 Structured Logic Design

As part of their discussion of algorithmic synthesis, Armstrong and Gray present the concept of automatic translation of a representative sample of VHDL constructs into hard-ware. They concentrate on translations that are application independent, rather than ones from specified programming styles into restricted sets of hardware.

The first discussion involves constructs that involve selection of a specific element from a specified set. The *case* statement implies selection of one case from a specified set of cases. The *if...then...else* statement implies selection of the highest priority condition that is *true* from a prioritized list of conditions. Also, one element of a vector may be selected by specifying an index value. All of these statements involve selection and, therefore, exhibit the functionality of a multiplexer. Figure E-1 shows several examples of VHDL constructs that can be mapped to multiplexer elements [10].

```
package TYPES is
   attribute ENCODING:
                         STRING;
   type ENUM is (A, B, C, D);
  attribute ENCODING of ENUM: type is "00 01 10 11"
end types;
use work.TYPES.all;
entity MUX is
  port (X, Y: in BIT;
        VECT: in BIT_VECTOR(3 downto 0);
        CHOICE: in ENUM;
        INDEX: in INGETER range 3 downto 0;
        Z1, Z2, Z3: out BIT);
end MUX;
architecture MUX_CONSTRUCTS of MUX is
begin
 MUX1: process (CHOICE, X, Y)
        begin
           case CHOICE is
              when A \Rightarrow Z1 <= X;
              when B \Rightarrow Z1 \leq Y;
             when C \Rightarrow Z1 \ll not X;
              when D \Rightarrow Z1 \ll not Y;
           end case;
        end process MUX1;
 MUX2: process (X, Y, VECT)
        begin
           if X = 1' then
              Z2 \ll VECT(3);
           elsif Y = 1' then
              Z2 <= VECT(2);
           else
              Z_2 \ll VECT(1) and VECT(0);
           end if;
        end process MUX2;
 MUX3: process (VECT, INDEX)
        begin
           Z3 <= VECT(INDEX);</pre>
        end process MUX3;
end MUX_CONSTRUCTS;
```

Figure E-1 VHDL constructs that map to multiplexer elements.

Process MUX1 in Figure E-1 is an example of a *case* statement that can be implemented with a multiplexer element. The *case* statement selects a function of inputs X and Y to assign to output Z1 based on the value of enumerative data type CHOICE. In package TYPES an attribute ENCODING of type STRING is declared which directs the automated design tool to assign binary codes to elements of the type ENUM. With this information, the tool translates the process MUX1 into the hardware circuit shown in Figure E-2. For example, when CHOICE=(00), ZI=X as implied by the VHDL source code.



Figure E-2 Hardware implementation for case statement.

The VHDL construct *if...then...else* also involves selection among several alternative actions. Therefore, multiplexer elements can be used to implement this construct. Process MUX2 in Figure E-1 shows an example that involves inputs X, Y, and VECT. By scanning the *if...then...else* clause, the automatic design tool can produce the truth table shown in Table E-1. Notice that the first *if* clause that is *true* selects the action to be performed. It is possible that more than one *if* clause is *true*. For example, if X=Y=1, then two of the *if* clauses are *true*. However, in this case, Z2 is assigned the value VECT(3) because the clause (*if* X='1') takes precedence over the clause (*if* Y='1').

X	Y	Z2
0	0	VECT(1) and VECT(0)
0	1	VECT(2)
1	0	VECT(3)
1	1	VECT(3)

Table E-1 Truth table for process MUX2.

Table E-1 directly implies the multiplexer implementation in Figure E-3, where signals X and Y are connected to the address inputs of the multiplexer and the data inputs for each XY combination are specified by the table entries.



Figure E-3 Hardware implementation for *if* statement.

Finally, if *VECT* is a vector and *INDEX* is an integer, then an assignment of the form Z3 <= VECT(INDEX)

is also a selection activity. In this case, one of the elements of *VECT*, as specified by *INDEX*, is being assigned to Z3. This type of statement also maps directly to a multiplexer device. Process MUX3 in Figure E-1 shows an example of this type of selection activity. In the absence of an attribute specifying a coding other than binary for *INDEX*, the example leads directly to the circuit in Figure E-4.



Figure E-4 Hardware implementation for vector indexing.

Next, program *loops* are discussed and illustrated through a classic ripple carry adder circuit. Figure E-5 shows the VHDL code for a 4-bit adder circuit implemented as a connection of full adders (*FA*). It is assumed that the *process* in architecture LOOP_ADDER

is embedded in a larger system that is not shown. The *loop* architecture maps directly to the iterative combinational logic network shown in Figure E-6.

```
entity ADD4 is
  port (A,B: in BIT_VECTOR(3 downto 0); CIN: in BIT;
         S: out BIT_VECTOR(3 downto 0); COUT: out BIT);
end ADD4;
architecture LOOP_ADDER of ADD4 is
begin
  process (A, B, CIN)
     variable CARRY: BIT_VECTOR(4 downto 0) := "00000";
     variable SUM: BIT_VECTOR(3 downto 0);
  begin
     CARRY(0) := CIN;
      for I in 0 to 3 loop
        SUM(I) := A(I) \text{ xor } B(I) \text{ xor } CARRY(I);
        CARRY(I+1) := (A(I) \text{ and } B(I)) \text{ or } (A(I) \text{ and } B(I))
           CARRY(I)) or (B(I) and CARRY(I));
      end loop;
     S \leq SUM;
     COUT <= CARRY(4);
   end process;
end LOOP ADDER;
```





Figure E-6 Hardware implementation of ripple carry adder.

Lastly, Armstrong and Gray also illustrate possible mappings of *functions* and *procedures* to hardware. Figure E-7 shows VHDL code for architecture FUNCTION_ADDER of entity ADD4. In the example, the logic equations for a full adder (FA) are implemented by *function* declarations. There are separate declarations for each output of the FA: sum (FA_S) and carry (FA_C).

- 223

```
architecture FUNCTION_ADDER of ADD4 is
  function FA_S (AIN, BIN, CIN: BIT) return BIT is
  begin
     return AIN xor BIN xor CIN;
  end FA_S;
  function FA_C (AIN, BIN, CIN: BIT) return BIT is
  begin
     return (AIN and BIN) or (AIN and CIN) or
            (BIN and CIN);
  end FA_C;
begin
  process (A, B, CIN)
     variable CARRY: BIT_VECTOR(4 downto 0) := "00000";
     variable SUM: BIT_VECTOR(3 downto 0) := "0000";
  begin
     CARRY(0) := CIN;
     for I in 0 to 3 loop
        SUM(I) := FA_S(A(I), B(I), CARRY(I));
       CARRY(I+1) := FA_C(A(I), B(I), CARRY(I));
     end loop;
     S \leq SUM;
     COUT <= CARRY(4);
  end process;
end FUNCTION_ADDER;
```

Figure E-7 Using functions to represent combinational logic.

Inside the program loop, the assignments to SUM(I) and to CARRY(I+1) are replaced by function calls. Since only the notation has changed, not the basic operation of the algorithm, it is clear that the architecture FUNCTION_ADDER can be mapped to the same hardware as architecture LOOP_ADDER. The general conclusion is that *functions* should be mapped to combinational logic circuits.

Similarly, *procedures* are used mainly as a convenience for ease of programming and could be used to replace the full adder in the previous example. In general, any VHDL code that uses procedures can be mapped to the same hardware as equivalent code without procedures. The main difference between procedures and functions is that functions always map to combinational logic, whereas procedures may map to sequential logic.

E.2 Mentor Graphics

The VHDL Style Guide for AutoLogic II [68] not only describes how to write synthesizable VHDL, but also gives details on how specific language constructs are implemented in the ultimate hardware. Examples are provided for concurrent *signal* assignment statements, *if* statements, *case* statements, and *variable* index assignments.

A concurrent *signal* assignment always results in combinational logic. This *signal* assignment specifies that a target *signal* is to receive some waveform. The waveform can either be a static value or some defined behavior. An example of a static assignment is:

D <= A;

This statement implies that signal D gets signal A and is therefore hard-wired to A. The waveform assigned to the target signal can also define some behavior. This behavior may be a simple logical expression or a complex expression that includes a function or procedure call. An example of a simple logical expression is:

 $D \leq A AND B;$

Assuming D, A, and B are all bits and ports, this description synthesizes into an AND gate.

The simple *if* statement does not contain enough information to synthesize a combinatorial network. For example, what happens if the condition evaluates to a boolean *false* value as described in Figure E-8. The output *signal D* receives input *signal B* when A is a '1'. This implies that if *signal A* is not a '1' then *signal D* must retain its old value. The description is then synthesized to a level-sensitive or transparent latch enabled by *signal A* whose output is *signal D*.

```
architecture RTL of IF_TEST is
begin
    process (A,B)
    begin
    if (A = `1') then
        D <= B;
    end if;
    end process;
end RTL;</pre>
```

Figure E-8 Code example for simple *if* statement.

In order to synthesize a combinatorial network using an *if* statement, the *if* statement must explicitly define the behavior for all possible evaluations of the condition. In Figure

E-8 this means that the *if* statement must define what *signal* D is to receive when *signal* A is not '1'. Figure E-9 shows how the addition of an *else* clause completes the definition of the behavior of an *if* statement. In this description a simple 2-to-1 multiplexer is modeled using an *if* statement with an *else* clause. The synthesized and optimized result is shown in Figure E-10.

```
entity MUX21 is
    port (A, B, C : in std_logic;
        D : out std_logic);
end MUX21;
architecture RTL of MUX21 is
begin
    process (A, B, C)
    begin
        if (A = `1') then
        D <= B;
        else
        D <= C;
        end if;
        end process;
end RTL;
```

Figure E-9 Code example for *if-else* statement.



Figure E-10 Synthesized hardware for *if-else* statement.

The *case* statement controls the execution of one or more sequential statements based on the value of an *expression*. VHDL requires all possible values for a selector must be described in a *case* statement. This can be done by having as many *when* clauses as selector *choices* or by use of a *when others* clause. The *case* statement implies a multiplexing architecture. For example, consider the description in Figure E-11. The selector for the *case* statement is *SEL*. Since *SEL* is two bits wide, there are 2^2 possible selector values for synthesis purposes. The *when others* clause accounts for other values not specified with a

- 225

when clause. The gate level design is implemented by a 4-by-1 multiplexer architecture as shown in Figure E-12.

```
entity MUX4 is
  port (DATA_IN : in std_logic_vector(3 downto 0);
        SEL : in std_logic_vector(1 downto 0);
        DATA_OUT : out std_logic);
end MUX4;
architecture RTL of MUX4 is
begin
  process (SEL, DATA_IN)
  begin
     case SEL is
       when "00" => DATA_OUT <= DATA_IN(0);
       when "01" => DATA_OUT <= DATA_IN(1);
       when "10" => DATA_OUT <= DATA_IN(2);
       when others => DATA_OUT <= DATA_IN(3);
     end case;
  end process;
end RTL;
```







AutoLogic II also supports indexed assignments using *variables*. In the following example *INDEX* is used as a *variable* index to the vector *DATA_IN*. An alternate architecture RTL2 can be used to describe the behavior of the entity MUX4 in Figure E-11. A function, *to_integer*, is assumed available to convert a 2-bit bit_vector to its integer representation.

```
architecture RTL2 of MUX4 is
begin
    process (SEL, DATA_IN)
    variable INDEX : integer;
    begin
        INDEX := to_integer(SEL);
        DATA_OUT <= DATA_IN(INDEX);
    end process;
end RTL2;
```

Figure E-13 Code example for variable index assignment.

The signal assignment statement

DATA_OUT <= DATA_IN(INDEX);</pre>

is equivalent to the *case* statement in Figure E-11 and results in the same multiplexer architecture in Figure E-12.

Appendix F VHDL Source Code

This appendix contains the VHDL source code for the examples used throughout this dissertation. The examples are grouped according to the tables of fault experiment results found in Appendix B. The behavioral description is first included as **example.vhd**, followed by the behavioral test vectors in WAVES format, **example_vectors.txt**. Lastly, one or more structural descriptions are outlined for gate level implementations. The VHDL structural descriptions are numbered for multiple realizations as **example_structure1.vhd** and **example_structure2.vhd**.

		F.1.4 casel_structure2.vhdVHDLobject: Architecture 'structure2' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/casel/casel.2level_s')ofentity*casel'(componentinterface*/home/rjh6v/csis/mentor/B4/eddm/work/casel/casel:casel)VHDLobject: Architecture 'structure2' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/casel/casel.casel)VHDLobject: Architecture 'structure2' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/casel.casel)VHDLobject: Architecture 'structure2' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/casel.casel)Senerated on: rifeb 13 16:47:05 1998Senerated by: rjh6vSource from: /home/rjh6v/csis/mentor/B4/eddm/work/casel/casel_slevel_sProgram: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997	<pre>//csis/mentor/B4/eddm/work/ //csis/mentor/B4/eddm/work/ /csis/mentor/B4/eddm/work/ - 5130NL bECLANATIONS - 1997 - 10000ent inv component inv component inv component orb - 10000ent orb</pre>	
<pre>F.1 CASE1 F.1. case1.vhd entity CASE1 is port (72.71, Y0: in BIT, VECTOR(1 downto 0); port (72.71, WIT);</pre>	<pre>end CASE1; architecture BEHAVE of CASE1 is begin process(SEL,Y2,Y1,Y0) begin case SEL is when *00* => when *01* => z <= Y1; when o11* => z <= Y1; end case; end case; end BEHAVE;</pre>	F.1.2 casel_vectors.txt * SEL Y2 Y1 YO Z: time; * Clause-CORRUPT(AND) 00 001 1: 500 ns; 10 101 1: 500 ns; 11 10X 1: 500 ns; 11 10X 1: 500 ns; 00 1101 0: 500 ns; 11 10X 1: 500 n	<pre>F.1.3 casel_structurel.vhd - vHbL object: Entity 'casel' (component interface '/home/rjh6v/ casel:casel') - ofenerated by: rjh6v - ofenerated by: rjh6v/ - ofenerated by: rjh6v/ - source from: vHbLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST - DERRY STATEMENT - LIBEARY STATEMENT - LIBEARY view: - PACKAGE STATEMENT USE iee: - PACKAGE STATEMENT USE iee: - PACKAGE STATEMENT - LIBEARY is a std_logic_l164.ALL; - PACKAGE std_logic_</pre>	<pre>i : out std_logic); end case1;</pre>

.

•
ARRAY4			. 230
<pre>component and3 component inv component or4; INLINE CONFIGURATIONS begin CONCURRENT ISIGNAL ASSIGNMENTS CONCURRENT INSTANTIATIONS end structure1; .2.4 array4_structure2.vhd</pre>	<pre>- WHDLobject: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ ddm/work/array4.array4.") of entity "array4" (component interface "/home/rjh6v/csis/mentor/B4/ ddm/work/array4.array4") of entity "array4" (component interface "/home/rjh6v/csis/mentor/B4/ Generated by: rjh6v v1 29 15:13:44 1998 - Generated by: rjh6v v1 29 15:13:44 1998 - Generated by: rjh6v/csis/mentor/B4/eddm/work/array4/array4_xi72a_s - Frogram: WHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 - Program: WHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 - Frogram: WHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 - Frogram: Component scheme to the total feb 5 16:56:22 FST 1997 - Component and total total total feb 5 16:56:22 FST 1997 - Component inv component inv - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS</pre>	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure2;	
1 	VE of ARAY4 is VF constructions for a second	<pre>tructure1.vhd tity `array4' (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/ wed Jun 10 10:00:18 1998 Tib6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/part /home/rjh6v/csis/mentor/B4/eddm/work/array4/part /hDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 /hDLwrite v8.5_56:56:56:56:56:56:56:56:56:56:56:56:56:5</pre>	rchitecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ lows: 0fentity"array4"(componentinterface"/home/rjh6v/csis/mentor/B4/ med Jun 10 10:00:18 1998 Tjh6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v /home/rjh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s rlh6v/csis/mentor/B4/eddm/work/array4/array4_gen_low_s

•

•

<pre>Land the second of the se</pre>		F.3 SHIFT4u	PACKAGE STATEMENT USE ieee.std_logic_1164.ALL; entity shiftdu is
<pre>1522 The first fir</pre>	<pre>transmission of the second of the secon</pre>		PORT LIST
<pre>101 10. In the first intervent in the first intervent in the first intervent int</pre>	<pre>provide the state of the s</pre>	library IEEE, ARTHNETIC; use IEEE:std_logic_l164.all; use ARTHNETIC.std_logic_all;	Port in std_logic_vector(1 downto 0); a : in std_logic_vector(3 downto 0); a : ont std_logic_vector(3 downto 0);
<pre>131 131 at attached atta</pre>	<pre>1223</pre>	entity shift4u is	end shift4u;
<pre>1527 add http://add/add/add/add/add/add/add/add/add/a</pre>	<pre>district in the intervent of all the intervent</pre>	DUC: in std_logic_vector(1 downto 0); A: in std_logic_vector(3 downto 0); A: out std_logic_vector(3 downto 0);	VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ shift4u/shift4u_gen_low_s")ofentity"shift4u'(componentinterface"/home/rjh6v/csis/mentor/
<pre>protect rest of a fit for a fit</pre>	<pre>Mathemater a halter is a state is a sta</pre>); end shift4u;	B4/eddm/work/shift4u:shift4u*) Generated on: Mon Jun 8 16:17:34 1998
1523 and	<pre>manual is net relation in relation in</pre>	architecture behave of shift4u is	Generated by: rjnov. Source from: /home/rjh6v/csis/mentor/B4/eddm/work/shift4u/shift4u_gen_low_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
The first of a first of	The first is a first is	<pre>process(OP,A) variable TMP: std_logic_vector(3 downto 0);</pre>	library ieee; library work;
Test in 0 to 2 1000 Test in 0 to 2 1000 Test in 0 to 2 1000 Test in 1000 Test in 1000 Test in 1000 <td>The first of the constraints of</td> <td>begun case OP is when "01" =></td> <td>use ieee.std_logic_1164.ALL; architecture structurel of shift4u is</td>	The first of the constraints of	begun case OP is when "01" =>	use ieee.std_logic_1164.ALL; architecture structurel of shift4u is
1521 and 150; and 15	The state of	For I in 0 to 2 loop TMP(I) := $A(I+1)$;	TYPE DECLARATIONS SIGNAL DECLARATIONS COMPANIENT DECLARATIONS
The function of	Table 1 and a control 1 and a control 1 and a component intra	end loop; TMP(3) := '0';	- CUTFORENT_DELENTATIONS COMPONENT and2 Commonent and2
1521 Status	Image: and the state is a second st	for I in 3 downto 1 loop	component inv component inv component nand2
10.00 10.00 <td< td=""><td>0::::::::::::::::::::::::::::::::::::</td><td>TWP[1] := A(L-1); end loop; montrol := 10:</td><td>component nand3 component or2</td></td<>	0::::::::::::::::::::::::::::::::::::	TWP[1] := A(L-1); end loop; montrol := 10:	component nand3 component or2
end case: D <= TPP; D <= TPP; end created end behave: end behave: end behave: end behave: end behave: end behave: E.3. shiftdu_yrectors.tr v or ND :: Lime: 0.010 010 550 m; 0.010 010 01 550 m; 0.010 010 010 01 550 m; 0.010 010 010 01 00 m; 0.010 010 01 550 m; 0.010 010 010 00 m; 0.010 010 01 550 m; 0.010 010 01 550 m; 0.010 010 010 00 m; 0.010 010 01 00 m; 0.0	met cs::::::::::::::::::::::::::::::::::::	when others => TWP:==A;	component xor2 INLINE CONFIGURATIONS
D <= TPD; end braves; end bra	D <= The; D <= The; and process; and process; and process; and process; and process; and process; F.3.3 shiftdu_yettor: F.3.3 shiftdu_yettor: F.3.4 shiftdu_structure2.vhd F.3.4 shiftdu_structure2.vhd F.3.5 shiftdu_structure3.vhd F.3.5 shiftdu_structure3.vhd F	end case;	Degin CONCURRENT SIGNAL ASSIGNMENTS
end brocess; end brocess; end brocess; end brocess; end brocess; end brocess; end brocess; end brocess; F.J. Shift(u.rclmtcl.th) envino top:::::::::::::::::::::::::::::::::::	mod process; end brocess; f.3.3 shiftdu_vectorstri f.3.3 shiftdu_vectorstri f.3.3 shiftdu_vectorstri f.3.3 shiftdu_vectorstri f.3.3 shiftdu_vectorstri f.4.4 shiftdu_vectorstri f.4.4 shiftdu_vectorstri f.4.4 shiftdu_vectorstri f.4.4 shiftdu_vectorstri f.4.4 shiftdu_vectorstri f.4.4 shiftdu_vectorstri f.4.4 shiftdu_structure2.htd f.4.4 shiftdu_structure2.htd f.4	D <= TMP;	COMPONENT INSTANTIATIONS end structure1;
F.3.2 shiftdu_vectors.txt * 0.010.1500:00:00:00:00:00:00:00:00:00:00:00:00:	F.3.2 shiftd_vectors.it * OP X D: time: * OP X	end process; end behave;	F.3.4 shift4u_structure2.vhd
<pre># Op A D : time; 0 0101 0101 : 500 ms; 0 0100 010 0101 : 500 ms; 0 0100 010 010 010 010 010 010 010 010</pre>	0.0 A D: time: 0.0 A D: time: 0.0 A D: time: 0.0010 1001 500 ms; 0.001 0011 500 ms; 0.001 0011 500 ms; 0.0 1001 500 ms; 0.0 001 500 ms; 0.0 001 500 ms; 0.0 1001 500 ms; 0.0 001 500 ms; 0.0 001 500 ms; 0.0 1001 500 ms; 0.0 001 500 ms; 0.0 001 500 ms; 0.0 1001 500 ms; 0.0 001 500 ms; 0.0 001 500 ms; 10 1001 500 ms; 0.0 001 500 ms; 0.0 001 500 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms; 0.0 ms; 0.0 001 100 ms; 11 1010 1010 500 ms;	F.3.2 shift4u_vectors.txt	VHDL object: Architecture "structure" (schematic "/home/rih6v/csis/mentor/B4/eddm/work/
000000000000000000000000000000000000	100 1001 500 150	<pre>% OP A D : time; co 0101 0 101 0 0 0 0 0 0 0 0 0 0 0 0 0 0</pre>	shift4u/shift4u_xi72a_s*)ofentity*shift4u*(componentinterface*/home/rjh6v/csis/mentor/B4/ eddm/work/shift4u:shift4u*)
01 1010 1010 500 ms; 10 1010 1010 500 ms; 10 1010 1010 500 ms; 11 1010 1010 1010 100 100 100 ms; 11 1010 1010 100 100 100 100 100 ms; 11 1010 1010 1000 100 100 100 ms; 11 1010 1010 100 100 100 100 100 100 10	11010 1000 500 ms; 10010 1000 500 ms; 11010 1010 100 100 ms; 11010 1010 100 100 ms; 11010 1010 100 ms; 11010 100 ms; 11010 100 100 ms; 11010 100 100 ms; 11010 100 100 ms; 11010 100 100 100 ms; 11010 100 100 100 100 100 100 100 100 1	00 ULOL VIUL : 500 ms; 00 1101 1101 : 500 ms;	Generated on: Wed Jul 29 15:28:47 1998 Generated by: rjh6v
<pre>10 1010 0100 : 500 ms; 11 0101 0101 : 500 ms; 12 010 0101 : 500 ms; 13 0101 010 : 500 ms; 14 0101 0101 : 500 ms; 15 0 ms; 15 0</pre>	110 1010 1000 500 ms; 11 010 1010 100 ms; 11 010 1010 100 ms; 11 010 100 ms; 11 010 100 ms; 10 010 100 ms; <	01 0101 0101 : 500 ms; 01 1010 0101 : 500 ms; 10 0101 : 500 ms;	Source from: /home/rjh6v/csis/mentor/B4/eddm/work/shift4u/shift4u_xi72a_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
<pre>11 1010 1010 : 500 ms; 11 1010 1010 : 500 ms; 11 1010 1010 : 500 ms; 11 1010 1010 : 500 ms; 12 3 Shiftdu_structurel.vhd 14 structures of shiftdu is 15 3 Shiftdu_structures of shiftdu is 16 3 structures of shiftdu is 17 3 structures of shiftdu is 18 3 structures of shiftdu is 18 3 structures of shiftdu is 18 3 structures of shiftdu is 19 3 structures of shiftdu is 19 3 structures of shiftdu is 10 0 0 mon jun 8 16:17:34 1998 19 5 structures in 10 0 mon jun 8 16:17:34 1998 10 mon jun 8 16:17:34 1998 10 mon jun 8 16:17:34 1998 11 mon jun 8 16:17:34 1998 12 mon jun 8 16:17:34 1998 13 structures in 14 structures in 15 mon jun 8 16:17:34 1998 14 structures in 15 mon jun 8 16:17:34 1998 16 mon jun 8 16:17:34 1998 17 mon jun 8 16:17:34 1998 17 mon jun 9 16:17:34 1998 18 mon jun 9 16:17:34 1998 19 mon jun 9 16:17:34 1998 10 mon jun 9 mon jun 9 16:17:34 1998 10 mon jun 9 10:17:34 19</pre>	<pre>11 1010 1010 : 500 ms; F.3.3 shift/u_structure1.vhd F.3.3 shift/u_structure1.vhd</pre>	10 1010 1010 500 ms; 11 0101 0101 500 ms;	library ieee;
<pre> TYPE DECLARATIONS TYPE DECLARATIONS TYPE DECLARATIONS VHDLobject: Ently "shiftu" (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/ CORPONENT_DECLARATIONS CORPONENT_DECLARATIONS COMPONENT INECTAL DECLARATIONS Component and Component inv Component inv Component inv Component inv CONCREATIONS CO</pre>	- TYPE DECLARATIONS - VHDLobject: Entity 'shiftdu' (component interface 'home/rjh6v/csis/mentor/B4/eddm/work/ shiftdu:shiftdu') - CORPONENT_DECLARATIONS - CONFORMENT_DECLARATIONS - CONFORMENT_DECLARATIONS - COMPONENT and - COMPONENT and - COMPONENT and - COMPONENT and - CONFORMENT - LIBRARY STATEMENT - LIBRARY STATEMENT - LIBRARY STATEMENT - LIBRARY ieee: - LIBRARY ieee: - LIBRARY ieee: - LIBRARY ieee: - COMPONENT INSTANTIATIONS - COMPONENT INSTANTIATIONS	11 1010 1010 : 500 ms; R 3 3 chiftán cfrinchineil Vhd	uuse isee.std./ architecture structure2 of shift4u is
VHDLObject: Bntity "shift4u" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ COMPONENT_DECLARATIONS = Greated by: "Jh6v" (component and 2 = Greated by: "Jh6v" a 16:17:34 1998 = Generated by: "Jh7v" a 16:17:34 1998 = Generated	VHDLObject: Bntity "shift4u" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ COMPONENT_DECLARATIONS = Generated for: 0n Jun 8 16:17:34 1998 = Generated by: rjh6v = Generated by: rjh6v = Source from: /home/rjh6v/csis/mentor/B4/eddm/work/shift4u/part = Source from: /home/rjh6v/csis/mentor/B4/eddm/work/shift4u		TYPE DECLARATIONS SIGNAL DECLARATIONS
Generated on: Mon Jun 8 16:17:34 1998 Generated by: hinkv Supponent or2 Supponent or2 INLINE CONFIGURATIONS Frogram: VHDwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 ELIBRARY STATEMENT COMPONENT INSTANTIATIONS COMPONENT INSTANTIATIONS COMPON	Generated on: Mon Jun 8 16:17:34 1998 Generated by: rjh6v/csis/mentor/B4/eddm/work/shift4u/part Source from: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Frogram: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 EVORTRENT SIGNAL ASSIGNMENTS CONCREATE SIGNAL ASSIGNMENTS CONCREATE SIGNAL ASSIGNMENTS CONFORT SIGNAL ASSIGNMENTS CONFORT INSTANTIATIONS end structure2; NSTANTIATIONS EIBRARY ieee: 1500 CONFORT SIGNAL ASSIGNMENTS CONFORT SIGNAL ASSIGNAL ASSIGNAL ASSIGNAL ASSIGNAL ASSIGNAL ASSIGNAL ASSIGNAL ASSIGNAL A	VHDLobject: Entity"shift4u" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ shift4u:shift4u")	COMPONENT_DECLARATIONS component and2
Source from: /home/rjh6v/csis/mentor/B4/eddm/work/shift4u/part Frogram: VHDuwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 ELTBRARY STATEMENT LIBRARY STATEMENT LIBRARY ieee: LIBRARY ieee:	Source from: /home/rjh6v/csis/mentor/B4/edm/work/shift4u/part Program: VHDwrite V8.1_2.4 Wed Feb 5 16:56:22 PST 1997 CONCURRENT SIGNAL ASSIGNMENTS CONPONENT INSTANTIATIONS LIBRARY ieee; LIBRARY ieee; LIBRARY ieee;	Generated on: Mon Jun 8 16:17:34 1998 Generated bv: rih6v	component inv component or2
CONCURRENT SIGNAL ASSIGNMENTS LIBERRY STATEMENT COMPONENT INSTANTIATIONS end structure2; ISBRAY jeee;	CONCURRENT SIGNAL ASSIGNMENTS LIBRARY STATEMENT COMPONENT INSTANTIATIONS end structure2; LIBRARY ieee;	Source from: /home/rjh6v/csis/mentor/B4/eddm/work/shift&u/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	INLINE CONFIGURATIONS begin
LIBRARY ieee: end structure2; end structure2;	LIBRARY ieee: end structure2,		CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS N
		LIBRARY ieee;	end structure2;

- SIGNAL DECLARATIONS - CONDONANT DECLARATIONS - CONDONANT AND CANDONANT AND CONDONANT	component and component inv	downto 0); LT: out BIT); INLINE CONFIGURATIONS	e is	COMPONENT INSTANTIATIONS end structurel;	F.4.4 less2_structure2.vhd		<pre>less2/less2_ri72a_s")ofentity"less2"(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/ work/less2:less2")</pre>	Generated on: Mon Aug 3 15:29:10 1998 Generated by: rjhfv	SOURCE ITOM: / NOME//ZINS/CSIS/MEDIOY/DE//GOUN/WORK/LESS/LESS/LESS/ZIN/ZA_S Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	library icee; library work:	use ieee.std_logic_l164.ALL; architecture structure2 of less2 is	SIGNAL DECLARATIONS COMPONENT_DECLARATIONS		component or2 - INLINE CONFIGURATIONS	Degin - CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS	end structure2; 2* (component interface */home/rjh6v/csis/mentor/B4/eddm/work/	15:25:34 1998 5v/csis/mentor/B4/eddm/work/less2/part v8.5_2.4 Wed Feb 5 16:56:22 PST 1997			r(1 downto 0); pr(1 downto 0);	<pre>structure (schematic '/home/rjh6v/csis/mentor/B4/eddm/work/ istructure (schematic '/home/rjh6v/csis/mentor/B4/eddm/work/</pre>	3 15:25:34 1998	5v/csis/mentor/B4/eddm/work/less2/less2_gen_low_s v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	. 23	2
less2.vhd	ry lete; eee.std_logic_l164.all; v LESS2 is	(A,B: in BIT_VECTOR(1 downto 0); LT SS22;	ecture BEHAVE of LESS2 is	ess (A, B) gin	[f A < B then LT <= '1';	iise LT <= '0'; 	Ind if; process; process;	less2_vectors.txt	r: time;	: I : 500 ns;			III	: 500 ns; : 500 ns; : 500 ns;	less2_structure1.vhd	object: Entity 'less2' (component	ess'') tested on: Mon Aug 3 15:25:34 19 tated by: rjh6v tefrom: /home/rjh6v/csis/menton ram: VHDLwrite v8.5_2.4 Mo	ieee; e.std_logic_1164.ALL;	less2 is ORT LIST	1 : in std_logic_vector(1 downto 0) 0 : in std_logic_vector(1 downto 0) 1 : in std_logic); t : out std_logic); ss2;	object: Architecture "structure" (ess2 ren low s"hofentitv"less2"(com	ss2:1ess2") erated on: Mon Aug 3 15:25:34 19	<pre>rated by: rjh6v cce from: /home/rjh6v/csis/mento: ram: VHDLwrite v8.5_2.4 W</pre>	/ ieee; / work;	

<pre>Section 1 Section 1 S</pre>
EGUAT3 E. BOUL3 F.S. 1 equation E.S. 1 equation T.S. 2 equation E.S. 1 equation T.S. 2
EGONU3 Edit RUL13 Fait equation Edit equation Edit equation
EGUINT: 1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
Sci ROUA3 Sti RUUA4
<pre>S 2003 S 2</pre>
Science Science
Sta CUU33 R.1 Editalia R.1 Editalia R.1. Editalia R.2. Editalia R.2.<
Sta COUNTS
Sta COUNT Sta
Sta CUU13 Sta CUU13 Sta CUU14 Sta
Standard
<pre>FTATATATATATATATATATATATATATATATATATATA</pre>
St R0131 F3. R0131
St R0N1 F3. R0N1
S. EQUAJ: F3. EQUA: F3. EQUA: F3.1. EqUANCI F3.1. EqUAN
S. RUN1 F3. RUN1 F4. RUN1 F4. RUN1 F4. RUN1 F4. RUN1 F5. RUN1
S. ROM3 F3. Lange F3. Fande
St R013 St R013 <td< td=""></td<>
SA TOULI S. TOULI
F3. RUN1
5. EQUAL 7.3. EQUAL 7.3. Equals of the second of the seco
S. ROMJ S. S. ULI
F3. RUN3 F4. RUN3 F4. RUN3 F4. RUN3 F4. RUN3 F4. RUN3 F5. RUN3
F3 RUN1 F3. RUN1 F4. RUN1 F4. RUN1 F5. RUN1
F3 EQN13 F3. EQN14 F3. EQN14 F3. Equily the state of t
F3 EQUA13 F3. EQUA13 F3. EQUA14 F4. equal3 is in the second control is in the second con
F3 EQUA13 F3. EQUA13 F3.1 equal3 versions F3.2 equal3 versions F3.3 equal3 versions F3.4 equal3 versions F3.5 equal3 versions F
F3 ROU13 F3. ROU13 F3.1 emiliant F3.1 emiliant Table 1 Table 2
F5 ROURJ F5. ROULJ
Signature Signature Signature Signation Signature
<pre>F3 F0U13 F3. F0U13 F5. F0U13 F5</pre>
<pre>F3 F0U13 F3. F0U13 F3</pre>
F5 EQUAJ3 F5. EQUAJ3 F5. EQUAJ3 F5. Equal3.term There realized in the second domain of the second domain of the second sec
F5 ROURJ St. GOURJ St. GOURS St. GO
F5 EQUA13 F5. EQUA13 F5. EQUA13 F5. Equal3 team The first of constraints Th
F5 BOUA3 F5 BOUA3 F5. BOUA3
F5 EQUA3F5 EQUA3F5 EQUA3F5. EQUA3F5.1 equa3.vbcF5.1 equa3.vbcF5.2 equa3.vbcF5.2 equa3.vbcF5.3
F5 EQUAJf5. EQUAJf5. EQUALf5.1 equal3.thdf5.1 equal3.thdf5.1 equal3.thdf5.1 equal3.thdf1.1 errorsf1.1 errors
F5 EQUA3 F5. EQUA3 F5. EQUA3 F5.1 equal3.vid Threaty events
F5 FOUAl3 F5 FOUAl3 F5. FOUAl3 Librery vest, Librery vest, Li
F5 ROUA3 F5. BOUA13 F5.1 Granity intervieweit Librery intervieweit <
F.5 FOUN13 F.5. FOUN14 F.5.1 equal3.vhd Library iee:
F5 FOUL3 F5.1 COUL3 F5.2 COUL3
F.S. ROULJ F.S. ROULJ F.S. COULJ F.S. Could.
F.S. ROUL3 F.S. ROUL3 F.S. COUL3
F.S. BOUNL3 F.S. BOUNL3 F.S. Countaine F.S. Countaine Here: Here:
F5 F0UAl3 F5.1 COUL3 F5.1 Coul3/vid Interview
F.S. BOUAL3 F.S.1 emility.rd F.S.1 emility.rd Ihere: Ihere: Interview:
F.S. EQUAL3 F.S. EQUAL3 F.S. L. equal3.And Hibrary less: Hibrary less: <t< td=""></t<>
F.5 EQUAL3 F.5 EQUAL3 F.1 equal3.vid Harer view Harer view <
F5 EQUAL3 F5.1 equal3.vhd F5.2 equal2.vhd F5.3 equal3.vhd F5.4 equal3.vhd F5.5 equal2.vhd F5.6 equal3.vhd F5.7 equal3.vhd F5.2 equal3.vhd F5.2 equal3.vhd F5.2 equal3.vec/siz
F5 EQUAL3 F5.1 equal3.Md F5.1 equal3.Md There yees: There yees: <t< td=""></t<>
F5 EQUAL3 F5.1 equal3.vhd F.1.1 equal3.vhd F.1.1 equal3.vhd F.1.1 equal3.vhd F5.1 equal3.vhd F5.2 equal3.vhd F5.2 equal3.vhd F5.3 equal3.vhd F5.4 equal3.vhd F5.5 equal3.vhd F5.6 equal3.vhd
F.5 EQUAL3 F.5 EQUAL3 F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.2 equal3.vhd Ilbrary tees: perity point is perity point
<pre>F5 FQUAL3 F5.1 equal3.vhd F5.1 equal3.vhd F5.2 equal3.vhd</pre>
F5 EQUAL3 F5.1 equal3.vhd F5.1 equal3.vhd Interry iee:
F.5 EQUAL3 F.5.1 equal3.vhd F.5.1 equal3.vhd Threaty tiees: Threaty view;
F.S. EQUAL3 F.S. EQUAL3 F.S. Lequal3.vhd Threaty tees: Library tees:
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Library view:
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Ibravy tee;
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd F.S.1 equal3.vhd F.S.1 equal3.vhd Ibrary iee: Ibrary iee: <
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd F.S.1 equal3.vhd F.S.1 equal3.vhd Intervy ieee: Intervie Interview:
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Tibrary tee:
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd F.S.1 equal3.vhd Ihrary tee: Ihrary vectors Itrary vectors entity sould is entity sould is entity entity entity sould is entity entity entity entity in arr vectors for a 11; post: in arr vectors in
F.S. EQUAL3 F.S. EQUAL3 F.S.1 cqual3.vhd Ilibrary see: F.S.1 cqual3.vhd Ilibrary see: Ibrary see: Ilibrary see: instry see: Ilibrary see: out: Ilibrary see: instry second out: Ilibrary see: out: Ilibrary see: instry second out: Ilibrary second out: Ilibrary second out: instry second out: Ilibrary second out: <thilibrary out:<="" second="" th=""> Ilibrary se</thilibrary>
F.S. EQUAL3 F.S. EQUAL3 F.S.1 counal3.vhd Ilbrary leee; Ilbrary ieee; Ilbrary ieee; see: secilogic_lif4.AL; use ieee std_logic_lif4.AL; use ieee std_logic_lif4.AL; use ieee std_logic_lif4.AL; is std_logic_lif4.AL; use ieee std_logic_lif4.AL; is std_logic_lif4.AL; std_logic_lif4.AL; is std_logic_lif4.AL; std_logic_li
F.S. EQUAL3 F.S.1 equal3.vhd F.S.1 equal3.vhd Ihrary ieee; Introperson Intrope
<pre>F.S EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Ilbrary ieee; Ilbrary ieee; Ilbrary ieee; Ilbrary iee; Ilbrary i</pre>
F.5 EQUAL3 F.5.1 Equal3.vhd F.5.1 equal3.vhd Ihrary iee; Introver, Ise std.logic_I164.AL; uee iees etd.logic_I164.AL; ued kontextracture and3p Degin pegin if A = 0 then begin if A = 0 then begin if A = 0 then <tr< td=""></tr<>
F.5 EQUAL3 F.5 EQUAL3 F.5.1 equal3.vhd Ilbrary ieee; Intrary ieee; intrary ieee; Intrary iee;
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Ilbrary ieee; Ilbrary ieee; Ilbrary work; Ilbrary ieee; Ilbrary ieee; Ilbrary ieee; Ilbrary work; Ilbrary work; Ilbrary work; Ilbrary work;<
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Ilbrary ieee; Ilbrary iee;
<pre>F.5 EQUAL3 F.5 EQUAL3 F.5.1 equal3.vhd f.5.1 equal3.shd f.5.1 equal3.</pre>
F.5 EQUAL3 F.5 EQUAL3 F.5.1 equal3.vhd Ihrary work: Invary week: Ihrary work: invary week: Ihrary work: uberview: Ihrary work: invary week: Ihrary work: uberview: It a = b them: uberview: If a = b them: uberview: If a = b them: uberview: If a = uo: uberview: It a = uo: <tr< td=""></tr<>
F.5 EQUAL3 F.5. EQUAL3 F.5.1 equal3.vid F.5.1 equal3.vid Interv ieee: ilibrary work; use ieee: std.logic_lid4.ALL; use ieee: std.logic_lid4.ALL; use iere: s
F.5 EQUAL3 F.5. EQUAL3 F.5.1 equal3.vhd Ihrary vork; Ihrary vork; Ihrary vork; Intrary iee; Intrary ist;
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd F.S.1 equal3.vhd F.S.1 equal3.vhd Invary tees; intrary tees; use ieee std.logic_l164.alt; use iee std.logic_l164.alt; use
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd F.S.1 equal3.vhd Ibrrary work: library work: port(library work: library work:
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd intrary verk; intrary work; intrary work; int
F.5 EQUAL3 F.5 EQUAL3 F.5.1 equal3.vhd F.5.1 equal3.vhd F
F.5 EQUAL3 F.5 EQUAL3 F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd Ilibrary work; use ieee: std_logic_l164.ALL; use ieee: std_logic_l164.ALL; std in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3; architecture BEHAVE of EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is pot(A, B: in BTT_VECTOR (2 downto 0); EQ: out BTT); end EQUAL3 is process (A, B) if A = B then if A = B
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd library work; Ibrary veck; logic_l164.all; use isee stol_logic_l164.all; - TYPE DECLANTIONS entity Econal logic_l164.all; - TYPE DECLANTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - SIGMAL DECLANATIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - SIGMAL DECLANATIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(A, B: in BIT_VECTOR(2 downto 0); EQ: out BIT); - CONFORTERTIONS port(B, B: In BIT_VECTOR(2 downto 0); EQ: OUT BIT); - CONFORTERTIONS port(B, B: In BIT<
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd library uere; library were; use teee std_logic_l164.AL; use teee teee std_logic_L164.AL; use teee std_logic_L164.AL; u
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd library ieee; library ieee; std.logic_l164.ALt; use ieee std_logic_l164.ALt; use ieee ieee std_logic_l164.ALt; use ieee ieee std_logic_l164.ALt; use ieee iee iee iee use ieee iee iee std_logic_l164.ALt; use ieee iee iee use ieee iee iee use iee iee iee iee use iee iee iee iee use iee iee use iee iee use iee iee use
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Iihrary work; Iihrary work; Iihrary work; Iir vector(164.all;) Iir vector(164.all;) Iir vector(185 in BIT_Vector(2 downto 0); EQ: out BIT); Iir vector(164.all;) Iir vector(185); Iir NE CONFORMATIONS Iir vector(185); Iir vector(185); Iir vector(185); Iir vector(185)
F.S. EQUAL3 ETHER
F.S. EQUAL3 EQUAL3 F.S.1 equal3.vhd Iibrary ieee; F.S.1 equal3.vhd Iibrary ieee; Iibrary ieee; Iibrary ieee; Iibrary is pott(Applic) is in BIT_VECTOR(2 downto 0); EQ: out BIT); Iibrary ieee; Iibrary ed EQUAL3 is Iibrary eductore EBHAVE of EQUAL3 is Iibrary ieee; Iibrary eductore EBHAVE of EQUAL3 is Iibrary is Iibrary is Iibrary eductore Iibrary is Iibrary is Iibrary eductore Iibrary eductore Iibrary eductore Iibrary eductore
F.5 EQUAL3 F.5. EQUAL3 F.5.1 equal3.vuck; inbrary icee; use icee.std_logic_l164.aLt; use icee.std_logic_
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Ilbrary work; Ibrary work; Ibrary work; Ibrary iee; Isrueture of equal3 is Ibrary iee; Ibrary work; Ibrary is Ibrary work; Ibrary work; Ibrary work;
F.S. EQUAL3 EQUAL3 F.S.1 equal3.vhd Iibrary ieee; F.S.1 equal3.vhd Iibrary ieee; Iibrary ieee; Iibrary ieee; Iibrary is in BIT_VECTOR(2 downto 0); EQ: out BIT); Iibrary ieecident and3p end EQUAL3; IIIBRAR of EQUAL3 is IIBRARATIONS Invine architecture BEHAVE of EQUAL3 is IIIBRARATIONS IIIBRARATIONS IIIBRARATIONS IIIBRARATIONS IIIBRARATE OF EQUAL3 is IIIBRARATIONS IIIBRARATIONS IIIBRARATE OF EQUAL3 is IIIBRARATIONS IIIBRARATIONS IIIBRARATE OF EQUAL3 is IIIIBRARATE OF EQUAL3 is IIIIBRARATE OF EQUAL3 is
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd Iibrary tee; Ibrary iee; Iibrary work; Iibrary work; Bibrary w
F.S. EQUAL3 F.S. EQUAL3 F.S.1 equal3.vhd library work; library work; library work; library iece; library iece; recting cound.si library work; library work; li
F.S. EQUAL3 EInternet EInterne EInternet EInternet
F.S. EQUAL3 Ibrary ieee; Ibrary ieeee; Ibrary ieeee; <t< td=""></t<>
F.S. EQUAL3 Iibrary ieee; Iibrary ieee; Iibrary veck;
F.S. EQUAL3 ETHER ETHER F.S.1 equal3.vhd Iibrary work; Iibrary work; Iibrary veck; Iibrary work; Iibrary work; Iibrary ieee; IIIbrary beclantrows IIIbrary work; Iibrary ieee; IIIbrary beclantrows IIIbrary work; Iibrary ieee; IIIbrary beclantrows IIIBrary work; IIIbrary iee; IIIBrary beclantrows IIIBrary work; IIIbrary iee; IIIBrary work; IIIBrary work; IIIbrary iee; IIIBrary work; IIIBrary work; IIIIbrary iee; IIIBrary work; IIIBrary work; IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
F.S. EQUAL3 Ilbrary ieee; F.S.1 equal3.vhd Ilbrary ieee; Ibrary ieee;
F.5 EQUAL3 F.5.1 Equal3.vhd F.5.1 equal3.vhd F.5.1 equal3.vhd Ibbrary icee; ecologic_l164.aL; Ibbrary icee; ecoreacic acticare;
F.S. EQUAL3 Ilbrary teee; Ilbrary work; Ilbrary work; Ibrary work; use teee.std_logic_1164.ALL; Ibrary ieee; Ilforal] or cructured of equal3 is
F.S. EQUAL3 Ibrary ieee; F.S.1 equal3.vhd Ibrary ieee; Ibrary ieee; Icruitecture structurel of equal3 is
F.5 EQUAL3 F.5.1 Elibrary work; F.5.1 equal3.vhd use ieee.std_logic_l164.all; architecture structured of equal3 is use ieee.std_logic_l164.all;
F.5 EQUAL3 F.5.1 Equal3.vhd F.5.1 equal3.vhd Ibbrary work; use ieee.std.logic_l164.ALL; use ieee.std.logic_l164.all; TYPE DECLARATIONS
F.S. EQUAL3 Ilbrary ieee; F.S.1 equal3.vhd Ilbrary work; Use ieee std logic_1164 .ALD; Ilbrary ieee; TYPE DECLAMATIONS CTAPE DECLAMATIONS
F.S. EQUAL3 F.S.1 equal3.vhd Ibrary work; Ibrary isse; Ibrary work; Ibrary work; Ibrary work; Ibrary isse;
F.S. EQUAL3 Ilbrary ieee; F.S.1 equal3.vhd each ieee structured logic_lid6.ALL; library ieee; archieccure structured of equal3 is
F.5 EQUAL3 F.5.1 equal3.vhd F.5.1 equal3.vhd architecture structured of equal3 is
F.S EQUAL3 F.S.1 equal3.vhd University and a structured of equal3.vhd Structured Structured of equal3.vhd Structured Structur
F.S. EQUAL3 F.S.1 equal3.vhd use ieee std_logic_1164.ALL;
F.S. EQUAL3 Ilbrary ieee; F.S.1 equal3.vhd equal3.vhd
F.S EQUAL3 F.S.1 equal3.vhd
F.S. EQUAL3 F.S. EQUAL3 F.S.1 enual3.vhd
F.S. EQUAL3
F.S EQUAL3
F.S EQUAL3
F.S EQUAL3

F.6 GREATER3 F.6.1 greater3.vhd	PORT LIST port(a : in std_logic_vector(2 downto 0);
library ieee; use ieee.std_logic_l164.all;	<pre>b : in std_logic_vector(2 downto 0); gt : out std_logic f, f,</pre>
entity GREATER3 is port(A.B. in INTEGER range -4 to +3; CT: out BIT); end GREATER3:	Grand and a second and a second and a second and a second
architecture BEHAVE of GREATER3 is begin	mentor/B4/eddm/work/greater3's) Generated on: Tue May 26 15:35:41 1998 Generated by: rjh6v restartar/anir/missier3/mester3/mester3/mester3/mester3/mester3/mester3/mester3/mester3
process (A, B) begin if A B then	Source irom: /nome/rjnov/csis/mentor/b4/com/work/greateris/greateris-gen_med_s
$GT <= 10^{-1}$; else $GT <= 0^{-1}$;	library ieee; library work; use ieee std_logic_1164.ALL; archirefinre structure1 of greater3 is
end if; end process; end BEHAVE;	SIGNAL DECLARATIONS COMPONENT_DECLARATIONS component inv
F.6.2 greater3_vectors.txt	component nand2 component nand3 component nord
8 A B GT : time: 8 Class I	component or4
100 100 0 : 500 ms; 101 101 0 : 500 ms; 110 110 0 : 500 ms;	DEGIL CONCURRENT SIGNAL ASSIGNMENTS
111 111 0 : 500 ms; 000 00 0 : 500 ms; 001 001 0 : 500 ms;	end structure1. F.6.4 greater3_structure2.vhd
010 010 0 : 500 ns; 011 011 0 : 500 ns;	VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/
10.1130 0: 500 ms; 1110 0: 500 ms; 111000 0: 500 ms;	greater3/greater3_xi72a_s")ofentity"greater3"(componentinterface"/home/rjh6v/csis/mentor/ B4/eddm/oorK/greater3:greater3) Generated on: Frijul 24 134:00:04 1998
001 010 0 : 500 ns; 101 100 1 : 500 ns;	Generated by: rjh6v - Source from: /honorrjh6v/csis/mentor/B4/eddm/work/greater3/greater3_xi72a_s Errorsm: Untrarite vR 5, 2, a wed Fach 5, 16.56.22 psrn 1997
110 101 1 : 500 ns; 111 110 1 : 500 ns;	rogram: vnuwrite vo:3_6.4 weu reb 3 i:30:42 F31 i39/ 1. librarv iee:
000 111 1 : 500 ms; 001 000 1 : 500 ms; 010 001 1 : 500 ms;	library work; use ieee.std_logic_l164.ALL;
	architecture structure2 of greater3 is TYPE DECLARATIONS STONAL DECLARATIONS
F.6.3 greater3_structure1.vhd	- COMPONENT_DECLARATIONS COMPONENT_DECLARATIONS component and2
VHDLobject: Bntity'greater3' (component interface'/home/rjh6v/csis/mentor/B4/eddm/work/ dreater3's):	component inv component or2 INLINE CONFIGURATIONS
Generated on: Tue May 26 14:42:29 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/greater3/part Promram: VHDLwrite v8.5.2.4 Wed Feb 5 16:56:22 PST 1997	<pre>begin CONCURRENT SIGNAL ASSIGNMENTS CONPONENT INSTANTIATIONS end structure2; </pre>
LIBARY STATEMENT LIBARY jeee; PACKAGE STATEMENT USE jeee.std.logic_1164.ALL; DATFLY TOTAPTAT	. 23
	4

- WDL object: Architecture 'structure' (commentic 'home/rjh6v/csis/mentor/B4/eddm/work/ greater3/greater3; sub_s')6fmtity greater3; sub_s')6fmtit greate

GREATER3

. •

٠,

. •

F.7 LES	architecture structured of le5 is
F.7.1 le5.vhd	SIGNAL DECLARATIONS COMPONENT, DECLARATIONS
library ieee;	component nor2
use ieee.std_logic_1164.all; entity LE5 is	COMPONENT OLZ INLINE CONFIGURATIONS
port(A: in INTEGER range -16 to +15; IE. out PITU).	begin CONCURRENT SIGNAL ASSIGNMENTS
end LES;	COMPONENT INSTANTIATIONS
architecture BEHAVE of LE5 is	F.7.4 le5_structure2.vhd
process (A)	
begin if A < 5 then	VHDLobject:Architecture"structure"(schematic"/home/rjh6v/csis/mentor/B4/eddm/work/le5/ le5_xi72a_s") of entity "le5" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/
	le5:le5") Generated on: Fri Jul 24 13:48:28 1998
us <= '0'; end if; end brocess;	Generated by: rjh6v - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/le5/le5_ri72a_s - Procorsm:
end BEHAVE;	
F.7.2 le5_vectors.txt	library ieee; library work;
8 A. LE : time;	use ieee.std_logic_l164.ALL; architerture structure? of le5 is
11111 1 : 500 ns; 000111 1 : 500 ns;	- SIGNAL DECLARATIONS
00101 1 : 500 ms; 00110 0 : 500 ms;	CONFONDAL_DELEARMILONS COMPONENT ANd2
	component inv component or2
	INLINE CONFIGURATIONS
	DEGIN CONCURRENT SIGNAL ASSIGNMENTS
	COMPONENT INSTANTIATIONS end structure2;
le5:le5") Generated on: Fri May 29 14:29:46 1998	
Generated by: rjh6v	
Source from: /home/rjh6v/csis/mentor/B4/eddm/work/le5/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	
LIBRARY STATEMENT	
LIBRARY iece;	
PACKAGE STATEMENT USE ieee.std_logic_l164.ALL;	
entity le5 is	
a : in std_logic_vector(4 downto 0); le : out std_logic	
); and 1a5.	
un to./ 	
vnuuooject.Arcuiteccure structure (schematic / nome/r)1000/ Cats/mentor/Par/catam/work//rej/ le5_gen_med_s*) of entity "le5" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/	
le5:le5") Generated on: Fri May 29 14:29:46 1998	
Generated by: rjh6v	
Source from: /nome/rjnov/csis/mencor/b4/ecom/work/fe3/fe3_fe3_fea_med_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	
library ieee;	
library work; use ieee.std_logic_l164.ALL;	23
	6

 And and a state of a	· GE23u			237
1 1 1 <th>se ieee.std_logic_l164.ALL; - TYPE DECLARATIONS TYPE DECLARATIONS SIGNAL DECLARATIONS COMPONENT_DECLARATIONS COMPONENT and2 component and4 component and4 compo</th> <th><pre>vill object: Architecure 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ e23u/ge23u.xi72a.s')ofentity'ge23u'(componentinterface'/home/rjh6v/csis/mentor/B4/eddm ork/ge23uige23u')ri Jul 24 13:44:22 1998 - Generated on: Fri Jul 24 13:44:22 1998 - Generated by: rjh6v - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/ge23u_xi72a_s - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: Frogram:</pre></th> <th></th> <th></th>	se ieee.std_logic_l164.ALL; - TYPE DECLARATIONS TYPE DECLARATIONS SIGNAL DECLARATIONS COMPONENT_DECLARATIONS COMPONENT and2 component and4 component and4 compo	<pre>vill object: Architecure 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ e23u/ge23u.xi72a.s')ofentity'ge23u'(componentinterface'/home/rjh6v/csis/mentor/B4/eddm ork/ge23uige23u')ri Jul 24 13:44:22 1998 - Generated on: Fri Jul 24 13:44:22 1998 - Generated by: rjh6v - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/ge23u_xi72a_s - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Frogram: Frogram:</pre>		
to 31 is 14.11 14.11 14.11 14.11 14.11 14.11 14.11 14.11 14.11 14.11	to 31; to 31; to 31; to 31; F.8.4 gc231 .		<pre>(component interface /nome/rjnov/csis/mentuct/b*/reduit wein/ 14:15:26 1998 v/csis/mentor/B4/eddm/work/ge23u/part v8:5_2.4 Wed Feb 5 16:56:22 PST 1997 </pre>	<pre>*structure' (schematic '/home/rjh6v/csis/mentor/B4/eddm/work/ tty'ge23u"(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/ 14:15:26 1998 v/csis/mentor/B4/eddm/work/ge23u/ge23u_gen_med_s v8.5_2.4 Wed Feb 5 16:56:22 PST 1997</pre>



library work; use ieee.std_logic_l164.ALL; use ieee.std_logic_l164.ALL; architecture structure of 1t12u is -- TYPE DECLARATIONS -- SIGNAL DECLARATIONS -- SIGNAL DECLARATIONS -- COMPONENT DAC2 -- UNLINE CONFIGURATIONS Degin -- COMPONENT INSTANTIATIONS end structure;

<u>librarv jeee</u> :	library work; use ieee.std_logic_l164.ALL; architecture structure of ot3n is	TYPE DECLARATIONS SIGNAL DECLARATIONS	- CORFORMAT_DELIARATIONS COMPONENT NANG2 - INLINE CONFIGURATIONS begin	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure;						k/							
F.10 GT3n	F.10.1 gt3n.vhd	library ieee; use ieee.std_logic_l164.all;	<pre>entity Gr3n is port(A: in INTEGER range -16 to +15; c</pre>	architecture BEHAVE of GT3n is begin process(a)	begin begin if A > -3 then GT <= 11'; else else -11';	end if. end process; end process;	F.10.2 gt3n_vectors.txt	B GT time; 10111 0 500 ms; 11011 0 500 ms; 11110 1 500 ms; 11110 1 500 ms; 00000 1 500 ms;	F.10.3 gt3n_structure.vhd		gusugusun) Generated on: Fri May 29 14:24:50 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/gt3n/part Frogram: VIDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997	 entry grow is PORT LIST Nort (gt : out std_logic_vector(4 downto 0); gt : out std_logic); end gt3n;	 	Source from: /home/rjh6v/csis/mentor/B4/eddm/work/gt3n/gt3n/gt3n_gen_med_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	

٠

.

•

F.11 COMPARE	F11.4 compare_structure1.vhd
F.11.1 compare.vhd	
library ieee; use ieee.std_logic_1164.all;	vnu uject: Antity compare (componentinteriace / compare:compare) Generated on: Sat Jan 31 11:48:58 1998
entity COMPARE is port(A,B: in BIT_VECTOR(1 downto 0); Y2,Y1,Y0: in BIT; Z: out BIT);	
end COMPARE; architerture REMAUE of COMDARE is	LIBRARY STATEMENT LIBRARY STATEMENT
	PACKAGE STATEMENT USE immession india 11.
process (A, B, IZ, II, IU) becin	entity compare is
if A > B then	PORT LIST
Z <= Y2;	port(
elsif A < B then	a : in std_logic_vector(1 downto 0);
	υ : in sta_logic_vector(I downto U); v0 : in sta logic:
	y1 : in std_logic;
end if;	y2 : in std_logic;
end process;	z : out std_logic
end BEHAVE;	1; end compare;

;

.

F.11.2 compare_vectors.txt

8 A B Y2 Y1 Y0 Z : time ; ••• ••

F.11.3 compare_vectors.alt

COMPARE - VID.object: Architecture "structure!" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ compare/compare.low_s") offentity"compare" (componentinterface "/home/rjh6v/csis/mentor/B4/ eddm/work/compare: compare")
 - denerated on: Sat Jan 31 11:48:58 1998
 - Generated by: rjh6v
 - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare_compare_low_s
 - Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 home/rjh6v/csis/mentor/B4/eddm/work/ ork/compare/part 6:56:22 PST 1997 ibrary ieee; library vork; use ieee.std_logic_l164.ALL; use ieee.std_logic_l164.ALL; architecture structurel of compare is - TYPE DECLARATIONS - SIGNAL DECLARATIONS - COMPONENT DECLARATIONS - COMPONENT and2 component inv component inv component nard2 component nard -- INLINE CONFIGURATIONS begin -- CONCURRENT SIGNAL ASSIGNMENTS -- CONFONERT INSTANTIATIONS end structurel;

240

F.11.5 compare_structure2.vhd

-- VHDL OJJSCT: ATCHITECTURE STLUTTER?' (Schematic '/howerk/pikey/csis/mentor/B4/eddm/work/ commare/compare_med_s') of entity' compare' (component interface '/home/rjh6y/csis/mentor/B4/ eddm/work/compare.med_s') of entity' compare '(component interface '/home/rjh6y/csis/mentor/B4/ eddm/work/compare.med_s') of entity' compare '(compare_med_s') eddm/work/compare.compare') is 11:56:27 1998 -- Generated Dy: rjh6y -- Generated Dy: rjh6y -- Source from: /home/rjh6y/csis/mentor/B4/eddm/work/compare_med_s' -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: Program: PST 1997 -- Program: PST 1987 -- Program: PST 1997 -- PST 1997 -- Program: PST 1997 -- PST 1

COMPARE

COMPARE3	ntor/B4/eddm/work/	entor/B4/eddm/work/ n6v/csis/mentor/B4/ re3_low_s re3_low_s	_ 242
001 000 100 1 500 ns 010 001 10 1 500 ns 100 011 0 500 ns 101 101 101 1 500 ns 101 101 101 1 500 ns 111 111 110 011 0 500 ns 111 111 100 11 0 500 ns 111 110 011 1500 ns 100 101 1500 ns 101 101 101 100 101 100 101 </th <th><pre>VHDLobject: Entity 'compare3' (componentinterface '/home/rjh6v/csis/me compare3:compare3') Generated on: Mon Feb 9 10:39:48 1998 Generated by: rjh6v/csis/mentor/B4/eddm/work/compare3/part Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare3/part Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare3/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBRANY isse: LIBRANY isse: PACKAGE STATEMENT PACKAGE STATEMENT -</pre></th> <th><pre></pre></th> <th>Degin CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structurel;</th>	<pre>VHDLobject: Entity 'compare3' (componentinterface '/home/rjh6v/csis/me compare3:compare3') Generated on: Mon Feb 9 10:39:48 1998 Generated by: rjh6v/csis/mentor/B4/eddm/work/compare3/part Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare3/part Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare3/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBRANY isse: LIBRANY isse: PACKAGE STATEMENT PACKAGE STATEMENT -</pre>	<pre></pre>	Degin CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structurel;
	- <u></u>	· .	
Z: OUL BIT) /		•.	
,Y1,Y0: in BIT;			
1; (2 downto 0); Y2, (PARE3 is	ţx	ialt	
ARE3 re3.vhd ogic_1164.al i BIT_VECTOR i BIT_VECTOR remave of COI r2, Y1, Y0) r2, then c then c then	(1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	 500 nts 	: 500 ns ; : 500 ns ; : 500 ns ;
<pre>COMI L1 compa ary ieee, std_l ieee.std_l itecture F compare3; compare3; compare3; n n begin s z < y elsif a > E</pre>	12		101 010 1 110 101 0 111 101 0 foup II

F.12.5 compare3_structure2.vhd

- 'VHD. Object: Architecture structure2' (schematic 'Nome/rjh6V/csis/mentor/B4/eddm/work/ compared) compared_s med_s form: eddm/work/compared_sequences'(compared) (compared) med_rjh6V/csis/mentor/B4/eddm/work/compared) eddm/work/compared) med_s form: Mome/rjh6V/csis/mentor/B4/eddm/work/compared by: rjh6V - Generated by: rjh6V - Generated by: rjh6V - Source from: Mome/rjh6V/csis/mentor/B4/eddm/work/compared_med_s - Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1999 - Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1999 - Program: VHDLwrite V8.5_2.4 Med Feb 5 16:56:22 PST 1999 - Program: VHDLwrite V8.5_2.4 Med Feb 5 16:56:22 PST 1999 - Program: VHDLwrite V8.5_2.4 Med Feb 5 16:56:22 PST 1999 - Program: VHDLwrite V8.5_2.4 Med Feb 5 16:56:22 PST 1999 - Program: VHDLwrite V8.5_2.4 Med Feb 5 16:56:22 PST 1999 - Program: Program Pada - Component and2 - Component and3 - Component and3 - Component invr Component navd3 - Component navd3 - Component navd3 - Program: PraNTIATIONS - Program: PraNTIATIONS - Program: PraNTIATIONS - Program: PraNTIATIONS - Component or2 - CONPONENT NSTANTIATIONS - Component or2 - CONPONENT NSTANTIATIONS - CONPONENT NSTA

<pre>F.13.4 compared_structure.vhd VHDLobject: Entity'compared'(componentinterface'/home/rjh6v/csis/mentor/B4/eddm/work/ compared:compared') UHDLobject: Entity'compared'(compared/part compared by: rjh6v </pre>	 - WIDL Object: Architecture "structure" (schematic "home/rjh6v/csis/mentor/B4/eddm/work/compared.compared. - Obmerated on: Won Feb 2 13:16:34 1998 - Generated on: Won Feb 2 13:16:34 1998 - Generated on: Mon Feb 2 13:16:34 1999 - Generated on: Mon Feb 2 13:16:34 1998 - Generated on: Mone/rjh6v/csis/mentor/B4/eddm/work/compared/compared. - Source from: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 FST 1997 - Program: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 FST 1997 - Tryp Dsclawrots - Tryp Dsclawrots - Tryp Dsclawrots - Stoket Dsclawrots Stoket Dsclawrots
<pre>F.13 COMPARE4 F.13. Compare4.vhd library ieee: use lese.std_logic_1164.all; entity COWPARE4 is port(A.B. in BIT_VECTOR(1 downto 0); Y2,Y1,Y0: in BIT_VECTOR(3 downto 0)); ent(A.B. in BIT_VECTOR(3 downto 0)); port(A.B. in BIT_VECTOR(3 downto 0)); end COMPARE4: architecture BEHAVE of COMPARE4 is port(A.B. Y2,Y1,Y0) begin process(A.B.Y2,Y1,Y0) if A > B then if A > B + B + B + B + B + B + B + B + B + B</pre>	F.13.2 compared_vectors.txt * A B Y2 Y1 Y0 Z : time ; * Croup II 0000 1111 0000 500 ms; 1010 1111 0000 1111 1500 500 ms; 1111 0000 1111 0000 1111 500 ms; * Croup II 1000 0100 1111 500 ms; 2 croup II 1000 0000 1111 500 ms; 2 croup III 1000 0000 1111 500 ms; 2 croup III 1000 0000 1111 500 ms; 2 croup III 111 0000 0000 500 ms; 2 croup III 1111 1111 500 ms; 2 croup III 1111 111 0000 0000 500 ms; 2 croup III 1111 1111 500 ms; 2 croup III 1111 1111 0000 0000 500 ms; 2 croup III 1111 1111 1000 0000 500 ms; 2 croup III 1111 1111 0000 0000 500 ms; 2 croup III 1111 1000 0000 500 ms; 2 croup III 1111 0000 1111 500 ms; 2 croup III 1000 1111 500 ms; 2 croup III 1000 1111 500 ms; 2 croup IIII 1111 0000 0000 500 ms; 2 croup IIII 1111 0000 0000 500 ms; 2 croup IIII 1111 0000 0000 1111 500 ms; 2 croup IIII 1111 0000 0000 1111 500 ms; 2 croup IIII 1111 0000 0000 1111 1111 0000 0000 1111 500 ms; 2 croup IIII 1111 0

· COMPARE4

001 000 1111 500 1111 500 1111 1110 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 0000 1111 1111 0000 1111 1111 1000 1111 1111 1000 1111 1111 0000 1111 1111 0000 1111 1111 0000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 1111 1111 1000 11111 1111 1000 <t< th=""><th><pre>- VHDLobject: Entity 'compare34' (componentinterface '/home/rjh6v/csis/mentor/B4/eddm/work/ compare34:compare34') - Generated on: Fri Mar 13 15:42:43 1998 - Generated by: Tjh6v - Generated by: Tjh6v/csis/mentor/B4/eddm/work/compare34/part - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Program: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Program: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Program: /home/rjh6v/csis/mentor/B4/eddm/work/compare34 is nort(</pre></th><th></th></t<>	<pre>- VHDLobject: Entity 'compare34' (componentinterface '/home/rjh6v/csis/mentor/B4/eddm/work/ compare34:compare34') - Generated on: Fri Mar 13 15:42:43 1998 - Generated by: Tjh6v - Generated by: Tjh6v/csis/mentor/B4/eddm/work/compare34/part - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Program: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Program: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/compare34/part - Program: /home/rjh6v/csis/mentor/B4/eddm/work/compare34 is nort(</pre>	
<pre>F.14 COMPARE34 F.14 COMPARE34 F.14.1 compare34.vhd library ieee: use ieee.std_logic_1164.all; use ieee.std_logic_1164.all; entiry COMPARE34 is port(A.B: in BHT_VECTOR(2 downto 0); Y2,Y1,Y0: in BHT_VECTOR(3 downto 0); entity COMPARE34 is port(A.B: in BHT_VECTOR(3 downto 0)); entity compares (a between the all of a state of compares (a begin pegin if A > B then if A > B then</pre>	<pre>Z <= Y0; else Z <= Y1; end if; end process; end bEHAVE; end process; end process; f.14.2 compare34_vectors.txt F.14.2 compare34_vectors.txt % A B Y2 Y1 Y0 Z : time; % G G 000 1111 0000 1111 0000 500 ns; 000 000 1111 0000 1111 0000 500 ns; 010 0110 0000 1111 0000 1111 500 ns; 110 110 0000 1111 0000 1111 500 ns; 110 110 0000 1111 0000 500 ns; 111 111 0000 1111 0000 500 ns; 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</pre>	001 000 000 1111 1110 0000 500 1111 010 011 1111 0000 1111 1000 500 111 110 101 0000 0000 1111 500 111 111 1110 0000 500 111 500 111 111 1110 0000 500 111 500 111 111 1111 0000 500 111 500 111 001 011 1111 0000 500 111 500 111 001 011 1111 1111 0000 500 111 500 111 011 100 011 1111 1111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500 111 500

. COMPARE34

. 245

LIERARY STATEMENT LIERARY STATEMENT ENCAGE STATEMENT PORT LIST PORT	 VHDL object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/add4/add4_low_s')ofentity'add4 (componentinterface 'home/rjh6v/csis/mentor/B4/eddm/work/add4:add4') Generated on: Thu Feb 26 08:52:45 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/add4/add4_low_s Program: VHDiwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Ibrary ieee; 	<pre>library work; uues ieee.std.logtc_li64.ALL; architecture structurel of add4 is - TYPE DECLARATIONS - SIGNAL DECLARATIONS - SIGNAL DECLARATIONS - COMPONENT_DECLARATIONS component and2 component buf component buf component tor3 - INLINE CONFIGURATIONS Degn - CONFORMENT INSTANTIATIONS Degn - CONFORMENT INSTANTIATIONS end structure1;</pre>	F.15.5 add4_structure2.vhd	<pre>11hrary ieee: 11hrary ieee: 11hrary work: 12hrary wor</pre>
<pre>F.15 ADD4 F.15. add4.vhd library IEEE, ARITHMETIC; use IEEE.std_logic_li64.all; use ARITHMETIC.std_logic_arith.all; use ARITHMETIC.std_logic_arith.all; use ARITHMETIC.std_logic_vector(3 downto 0);</pre>	end add4; architecture behave of add4 is begin process(A,B) begin D <= A + B; end process; end behave;	F.15.2 add4_vectors.txt * A B D: time ? 0000 0000 0000 : 500 ms; 0000 1111 1111 : 500 ms; 1111 0000 1111 : 500 ms; 1111 0000 1111 : 500 ms; 0001 1111 0000 : 500 ms; 0101 1010 : 500 ms; 1010 1010 : 500 ms; 1011 1000 : 500 ms;	A B D: time ; 0000 0000 500 ms; 0000 1111 1111 500 ms; 1111 0000 1111 500 ms; 1111 1111 500 ms; 1111 1111 1101 500 ms; 1111 0001 1111 1000 500 ms; 0001 1111 0000 500 ms; 0001 1111 0000 500 ms; 0101 1010 500 ms; 1010	0010 1111 0001 1500 ns; 0001 1101 1100 500 ns; 0001 1011 1100 500 ns; 1011 1000 500 ns; ns; 0001 1011 1100 500 ns; ns; 0001 1011 1100 500 ns; ns; ns; 011 1000 rertated ns; retated ns; ns; ns; ns; 011 retated ns; rinkov/csis/mentor/sais/mento

- VHDL Object: Architecture 'structure' (schematic 'Dome/rjh6V/csis/mentor/B4/schem/ add4_nor/schematic') of entity 'add4_nor' (component interface '/home/rjh6V/csis/mentor/B4/ schem/add4_nor'add4_nor') of entity 'add4_nor' (component interface '/home/rjh6V/csis/mentor/B4/ - Generated on: Thu Feb 26 09:56:16 1998 - Generated Db: rjh6V - Scurce from: /home/rjh6V/csis/mentor/B4/schem/add4_nor/schematic - Frogram: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite V8:5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: Proceeding in 'Dome/rjh6V/csis/mentor/B4/schem/add4_nor/schematic' - Program: Proceeding is - TYPE DECLARATIONS - OCHONERN'DECLARATIONS - OCHONERN'DECLARATIONS - Component inv - TLINE CONFIGURATIONS - Component inv - COMPONENT INSTANTIONS - Component inv - COMPONENT INSTANTIONS - COMPONENT INS

·

•

0001 1111 0 10000 500 ms; 0010 1111 0 10011 500 ms; 0000 1111 0 10011 500 ms; 0000 1110 10 1111 500 ms; 0000 1111 10 1100 500 ms; 0000 1111 10 1100 500 ms; 0000 1111 0 1100 500 ms; 0001 0111 0 1100 500 ms; 0010 0111 0 1101 500 ms; 0010 0111 0 1001 500 ms; <td< th=""><th> VHDL Object: Entity "addawc" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ addawc:addawc") Otherated by: rjh6v Generated by: rjh6v Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/R4/eddm/work/addawc/part Program: VHDLwrite v6.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBBARY STATEMENT Frogram: VHDLwrite v6.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBBARY STATEMENT Frogram: VHDLwrite v6.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBBARY STATEMENT LIBBARY STATEMENT PACKAE STATEMENT PAC</th><th><pre></pre></th></td<>	VHDL Object: Entity "addawc" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ addawc:addawc") Otherated by: rjh6v Generated by: rjh6v Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/R4/eddm/work/addawc/part Program: VHDLwrite v6.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBBARY STATEMENT Frogram: VHDLwrite v6.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBBARY STATEMENT Frogram: VHDLwrite v6.5_2.4 Wed Feb 5 16:56:22 FST 1997 LIBBARY STATEMENT LIBBARY STATEMENT PACKAE STATEMENT PAC	<pre></pre>
<pre>F.16 ADD4wc F.16.1 add4wc.vhd library IEEE, ARITHMETIC; use IEEE.std_logic_li64.all; use ARITHMETIC.std_logic_arith.all; use ARITHMETIC.std_logic_vector(3 downto 0);</pre>	<pre>architecture behave of add4wc is begin process(A.B.CIN) variable operand1, operand2 : std_logic_vector(4 downto 0); variable sum : std_logic_vector(1 downto 0); variable carry_in : std_logic_vector(1 downto 0); pegin pegin operand1 := '0' & B; carry_in := '0' & B; carry_in := '0' & CIN'; sum := operand2 + carry_in; b <= sum(3 downto 0); cOUT <= sum(4); end process; end behave;</pre>	A B CLW COUP D : time ; 0000 1111 0 0 1111 : 500 ms; 11111 111 111 1111 : 500 ms; 11111 111 111 1111 : 500 ms; 11111 111 111 1111 : 500 ms; 0000 1111 1 1 0000 : 500 ms; 11111 0000 1 1 0000 : 500 ms; 11111 0000 : 500 ms; 11111 0000 1 1 0011 : 500 ms; 1010 1010 1 1 0101 : 500 ms; 1010 1111 1 0101 : 500 ms; 1010 111 1 0101 : 500 ms; 1111 0000 0 0 01010 0000 1111 0 0 1111 : 500 ms; 1111 111 11 1111 11 1111 : 500 ms; 1111 0000 0 1 0 01111 : 500 ms; 1111 0000 0 1 0 10101 : 500 ms; 1111 0000 1 1 0010 : 500 ms; 1111 0000 1 0 1111 : 500 ms; 1111 1 11 1 1 1111 : 500 ms; 1111 0000 1 0 10101 : 500 ms; 1111 0000 1 1 00101 : 500 ms; 1111 0010 1 1 1 0101 : 500 ms; 1111 0010 1 1 1 0101 : 500 ms; 1111 0010 1 1 1 0101 : 500 ms; 1010 1 101 1 1 0101 : 500 ms;

ADD4wc

,

•

F.16.5 add4wc_structure2.vhd

- Unit. object. Machineture "tructure" (schemetic "homor/sjh6v/csis/mentoc/pd/schemetic "homor/sjh6v/cs

•

F.17 ADD8	00000010 1110111 1111001 : 500 ms;
F.17.1 add8.vhd	00000010 1111111 1100001 : 500 ns; 00000010 1011111 1100001 : 500 ns;
<pre>library IEEE, ARTTHMETIC; use IEEE.std_logic_l164.all; use ARITHMETIC.std_logic_arith.all;</pre>	& 00000100 11110111 1111011 : 500 ms; 00000100 11101111 11110011 : 500 ms; 00000100 1101111 11100011 : 500 ms;
entity add8 is	
<pre>port A: in std_logic_vector(7 downto 0); B: in std_logic_vector(7 downto 0); D: out std_logic_vector(7 downto 0);</pre>	00001000 11101111 11101111 : 500 ns; 00001000 11011111 11100111 : 500 ns; 00001000 10111111 11000111 : 500 ns;
); end add8;	00010000 1101111 11001111 : 500 ms; 00010000 1011111 11001111 : 500 ms;
architecture behave of add8 is begin process(A.B)	F.17.4 add8_vectors.opt
begin best A + B; b <= A + B; end behave;	© A B D : time; 0000000 00000000 : 500 ms; 00000000 0111111 1111111 : 500 ms; 1111111 : 500 ms; 11111111 0000000 11111111 : 500 ms; 11111111 : 500 ms;
F.17.2 add8_vectors.txt	
& A B D : time ; 00000000 00000000 00000000 : 500 ns; 000000000 11111111 1111111 : 500 ns;	1111111 00000001 00000000 : 500 ns; 01010101 01010101 10101010 : 500 ns; 10101010 10101010 01010100 : 500 ns;
1111111 00000000 1111111 : 500 ns; 11111111 11111111 : 500 ns;	0000010 1111111 00000001 : 500 ms; 00000100 1111111 00000011 : 500 ms;
00000001 1111111 00000000 : 500 ms; 1111111 00000001 00000000 : 500 ms; 10101011 01010101 10101010 : 500 ms; 10101010 10101010 0 : 500 ms;	8 00000001 1111101 1111110 : 500 ns; 00000001 11111011 1111100 : 500 ns; 00000001 11110111 11111000 : 500 ns;
F.17.3 add8_vectors.cla	00000010 1111011 1111101 : 500 ms. 00000010 11110111 11111001 : 500 ms.
<pre>% A B D : time ; 00000000 0000000000 : 500 ns;</pre>	00010000 1111111 00001111 : 500 ms; 00100000 1111111 00011111 : 500 ms;
0000000 11111111 : 500 ms; 11111111 0000000 11111111 : 500 ms; 11111111 11111111 1111110 : 500 ms;	00001000 11101111 11101111 : 500 ns; 00001000 11011111 11001111 : 500 ns; 00001000 1011111 11000111 : 500 ns;
00000001 1111111 00000000 : 500 ms; 1111111 00000001 00000000 : 500 ms; 01010101 01010101 10101010 : 500 ms; 101010101 01010101 0 10010100 : 500 ms;	8 00010000 1101111 11001111 : 500 ns; 00010000 1011111 11001111 : 500 ns;
& 00000010 1111111 00000001 : 500 ms; 00000100 1111111 00000111 : 500 ms; 00001000 1111111 00001111 : 500 ms; 00010000 1111111 00001111 : 500 ms;	F.17.5 add8_structure1.vhd VHDL object: Entity *add8* (component interface */home/rjh6v/csis/mentor/B4/eddm/work/add8*add8) add8:add8)
& 00000001 1111101 1111100 : 500 ns; 00000001 11111011 1111100 : 500 ns; 00000001 11110111 11111000 : 500 ns;	Generated by: rjh6v Generated by: rjh6v/csis/mentor/B4/eddm/work/add8/part Program: VHDWrite V8.5_2.4 Wed Feb 5 16:55:22 PST 1997
0000001 1101111 1110000 : 500 ms; 00000001 11011111 11100000 : 500 ms; 00000001 1011111 11000000 : 500 ms; & 00000010 11111011 1111101 : 500 ms;	LIBRARY STATEMENT LIBRARY ieee; DeCKAGE STATEMENT USE ieee.std_logic_l164.ALL; entity add8 is

-

ADD8

ċ

÷

F.18 SUB4	F.18.4 sub4_structure1.vhd
F.18.1 sub4.vhd	B VHN: object: Entity 'suph' (commonent interface '/home/rih6y/csis/mentor/B4/eddm/work/ P
library IEEE, ARITHMETIC; use IEEE.std_logic_1164.all;	-view of subders mattery and the provident included view of subders and the subders of the subde
use ARITHMBTIC.std_logic_arith.all;	Generated by: rjubv Source from: /home/rjh6v/csis/mentor/B4/eddm/work/sub4/part Prooram: VHDLwrite v8.5 2.4 Wed Feb 5 16:522 PST 1997
entity sub4 is port {	- I.TRRARY STATEMENT
A: in std_logic_vector(3 downco v); B: in std_logic_vector(3 downto 0); D:t_logic_vector(3 downto 0);	LIBRARY ieee; Package Statement
); b; b; b; b; b; b; b; b; b; b; b; b; b;	USE ieee.std_logic_1164.ALL; entity sub4 is
enu suue;	PORT LIST DOXI(
architeccure benave of suby is begin	a : in std_logic_vector(3 downto 0); b : in std_logic_vector(3 downto 0);
process (A, B) begin	d : out std_logic_vector (3 downto 0)
D <= A - B; end process;	end sub4;
end behave;	
F.18.2 sub4_vectors.txt	VHDL object: Architecture "structure" (schematic "home/rjh6v/csis/mentor/B4/e0dm/work/ sub4/sub4_med_s") ofentity"sub4" (componentinterface"/home/rjh6v/csis/mentor/B4/eddm/work/
8 A B D : time ;	sub4:sub4") Generated on: Thu Apr 2 11:55:41 1998
0000 0000 0000 : 500 ns;	Generated by: theme.veiter.veite.veite.veiter.veite.veiter.veiter.veiter.veiter.veiter.veiter.veiter.veiter
	Source Arons: /HDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
1111 1111 0000 : 500 ns; 8	1 library ieee;
0000 0001 1111 : 500 ns;	library work;
	ase received in the structure of subd is
1010 0101 0101 : 500 ns; 8	TYPE DECLARATIONS SIGNAL DECLARATIONS
F.18.3 sub4_vectors.cla	COMPONENT_DECLARATIONS component and2
	component buf
	component invr component invr
0000 1111 0001 : 500 ms; 1111 0000 1111 : 500 ms;	component nand2p component or3
1111 1111 0000 : 500 ns; A	
0000 0001 1111 : 500 ns;	INLINE CONFIGURATIONS begin
	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS
601 0101 0101 · 100 · 101 0101 010	end structurel;
0001 0000 0001 : 500 ns; 0010 0000 0010 : 500 ns;	
8 0000 0001 1111 : 500 ns;	
0000 0010 1110 : 500 ns; 0000 0100 1100 : 500 ns;	
8 0001 0010 1111 : 500 ns; 0001 1101 : 500 ns;	
St ADC : TATT AATA TAAA 8	
0010 0100 1110 : 200 ns:	
	25:



-- VHDL object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/schem/ subd_Cla/schematic') of entity 'sub4_cla' (component interface 'home/rjh6v/csis/mentor/B4/ schem/sub4_cla:sub4_cla') of entity 'sub4_cla' (component interface 'home/rjh6v/csis/mentor/B4/ - Generated on: Thu Apr 2 15:31:30 1998 - Generated on: Thu Apr 2 15:31:30 1998 - Generated on: Thu Apr 2 15:31:30 1998 - Generated by: Tjh6v - Source from: home/rjh6v/csis/mentor/B4/schem/sub4_cla/schematic - Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 - Program: Pettor Structure2 of sub4 is - TYPE DECLARATIONS - COMPONENT DECLARATIONS - ORPONENT INC - TWLNE CONFIGURATIONS - Component inv - COMPONENT INC - COMPONENT INSTANTIATIONS - COMPONENT IN

•

SUB4

•

ADDSUB4	01 1010 1011 : 500 ms; 10 0101 0101 : 500 ms; 11 0111 0100 : 500 ms; 11 0110 1000 : 500 ms; 12 0110 1000 : 500 ms; 13 CLA 10 0000 0011 : 500 ms; 10 0000 01111 : 500 ms; 10 0010 1111 : 500 ms; 10 0010 1111 : 500 ms; 10 0100 1101 : 500 ms; 10 0100 1100 : 500 ms; 10 0000 0000 : 500 ms; 10 0 0000 000 0000 ms; 10 0 0000 000 0000 ms; 10 0 0000 000 0	<pre>HDL object: Entity addsub4 (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/ sub_4taidd_sub_4') sub_ataidd_sub_4') senerated on: Thu Apr 2 13:27:01 1998 Senerated by: rjh6v r 2 13:27:01 1998 Senerated by: r 2 1091c_lt64.ALL; ity addsub4 is - PORT LIST - P</pre>	<pre>HDL object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ sub_l add_sub_4_med's)offentity add_sub_4'(componentinterface'/home/rjh6v/csis/mentor/ addm/work/add_sub_4:add_sub_4:) Senerated on: Thu Apr 2 12:27:01 1998 Senerated by: /home/rjh6v/csis/mentor/B4/eddm/work/add_sub_4/add_sub_4_med_s Senerated by: /home/rjh6v/csis/mentor/B4/eddm/work/add_sub_4/add_sub_4_med_s Program: VHDLwrite v8:5_2.4 Wed Feb 5 16:56:22 PST 1997</pre>
ADDSUB40 001I addsub4.vhd0 110Cy IEEE. All'HUTC;0 011Cy IEEE. std_logic_lid4.all;Cy IEEE. std_logic_arith.all;Cy IEEE. std_logic_arith.all;Cy IEEE. std_logic_arith.all;Ch IEEE. std_logic_arith.all;Ch IEEE. std_logic_arith.all;Ch IEEE. std_logic_arith.all;Ch IEEE. std_logic;Ch IEEE. std_logic;Ch IEEE. std_logic;Addsub4 isCh IEEE. std_logic;All is std_logic;	<pre>: in std_logic_vector(3 downto 0); : out std_logic_vector(3 downto 0); sub4; cture behave of addsub4 is cture behave of addsub4 is cture behave of addsub4 is ss(OP,A.B) ss(OP,A.B) OP = '0' then 'c= A + B; 'c= A - B; 'c= A + B; 'c= A + B;</pre>	0000 0000 500 ms; 1111 1111 500 ms; 1111 1111 500 ms; 1111 1111 500 ms; 1111 1100 500 ms; 1111 1111 1000 500 ms; 0001 0000 500 ms; mdd. 0101 1010 1010 500 ms; 1010 1010 500 ms; mdd. 1010 1010 500 ms; mdd. 1011 1111 500 ms; mdd. 1111 1111 500 ms; mdd. 1111 1111 500 ms; mdd. 1111 1000 500 ms; mdd. 1111 1111 500 ms; mdd. 1111 1111 500 ms; mdd. 1111 1000 500 ms; mdd. 1111 1111 500 ms; mdd. 10101 <td>addsub4_vectors.cla v B D : time : add 1111 1111 : 500 ns; 6 0000 1111 : 500 ns; 6 1111 1110 : 500 ns; 6 1111 1110 : 500 ns; 7</td>	addsub4_vectors.cla v B D : time : add 1111 1111 : 500 ns; 6 0000 1111 : 500 ns; 6 1111 1110 : 500 ns; 6 1111 1110 : 500 ns; 7

•

library ieee; library vork; use ieee.std_logic_l164.ALL; architecture structure1 of addsub4 is -- FYEB DECLARAATIONS -- SIGNAL DECLARAATIONS -- COMPONENT DECLARAATIONS component and2 component unv component unv component or 2 component or 2

.

F.19.5 addsub4_structure2.vhd

-- VHDL Object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/schematic') of entity "addsub4_cla" (schematic 'home/rjh6v/csis/mentor/ addsub4_cla/schematic') of entity "addsub4_cla" (componentinterface 'home/rjh6v/csis/mentor/ B4/schem/addsub4_cla: 14:56:03 1998 -- Generated by: Tjh6v -- Generated by: Tjh6v -- Broute from: /home/rjh6v/csis/mentor/B4/schem/addsub4_cla/schematic -- Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 -- TYPE DECLARATIONS -- TYPE DECLARATIONS -- TYPE DECLARATIONS -- TYPE DECLARATIONS -- TWLINE CONFIGHTANTONS -- TWLINE CONFIGHTANTONS -- COMPONENT INTONS -- COMPONENT INTONS -- COMPONENT INTONS -- CONFORT INTONS -- CONFORT INTONS -- CONFIGHTANTONS -- CONFIGHTAN

F.20 INC4 F.20 ind the	use ieee.std_logic_1164.ALL; architecture structurel of inc4 is 	n.~
r.2011 HRC4.VHG Library IEBE, ArthMRFIC; Library FEBE fici 115A 311.	COMPONENT_DECLARATIONS component and2 component inv	
use ARITHWETIC.std_logic_arith.all/	<pre>component xor2 INLINE CONFIGURATIONS</pre>	
entiry inc4 is port (in INTEGER range -8 to +6;	Degical concurrent signal assignments concorrent instantiations component instantiations	
2: OUT INTEGER FANGE -/ CO +/); end inc4;	F.20.4 inc4_structure2.vhd	
architecture behave of inc4 is	VIBL object: Architecture "structure" (schematic "/home/rih6v/csis/mentor/B4/eddm/work/ inc4/inc4_xi72a_s "bentity'inc4"(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/work/	
begin $Z = Y + 1$; end process;		
end behave;	Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	
F.20.2 inc4_vectors.txt	library work; library work;	
& Y .Z : time; 000 0001 : 500 ms; 1111 0000 : 500 ms;	are received and and the structure? of inc4 is SIGNAL DECLARATIONS	
8 0111 1000 : 500 ms;	COMPONENT_DECLARATIONS component and 2 commonent int	
1011 1100 : 500 ms; 1101 1110 : 500 ms; 1110 1110 : 500 ms:	Component or 11V component or 21V INLINE CONFIGURATIONS	
F.20.3 inc4 structure1.vhd	begin CONCURRENT SIGNAL ASSIGNMENTS COMPONENT THETANTTANE	
VHDL object: Entity 'inc4' (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/	end structure2;	
inc4:inc4*) Generated on: Tue May 26 16:18:51 1998		
Generated by: rJh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/inc4/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997		
LIBRARY STATEMENT LIBRARY STATEMENT		
FACKAGE STATEMENT USE ieee.std_logic_l164.ALL;		
entity inc4 is PORT LIST		
y : in std_logic_vector(3 downto 0); y : out std_logic_vector(3 downto 0)); end inc4:		
VHDL object: Architecture *structure* (schematic */home/rjh6v/csis/mentor/B4/eddm/work/ inc4/inc4_gen_low_s*) ofentity*inc4* (componentinterface*/home/rjh6v/csis/mentor/B4/eddm/		
work/inc4:inc4") Generated on: Tyue May 26 16:18:51 1998		
Generated by: /home/rjh6v/csis/mentor/B4/eddm/work/inc4/inc4_gen_low_s - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/inc4/inc4_gen_low_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997		
library ieee; library work;	. 256	256
_	5	5

F.21 INC8 F.21.1 inc8.vhd	Generated on: Wed Jun 3 10:53:06 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/inc8/inc8_gen_med_s Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997
<pre>library IEEE. ARTHMETIC; use IEEE.std_logic_1164.all; use ARTHMETIC.std_logic_arith.all; entity inc8 is port(</pre>	<pre>library ieee; library work; use ieee.std_logic_l164.ALL; architecture of inc8 is - TYPE DECLARATIONS - SIGMAL DECLARATIONS - COMPONENT DECLARATIONS</pre>
Z: out INTEGER range -1Z/ to +1Z/); end inc8;	component and component inv component inv
architecture behave of inc8 is begin process(Y) begin $Z \le Y + 1;$ end process; end behave:	INLINE CONFIGURATIONS ben
F.21.2 inc8_vectors.txt	
<pre>% Y Z : time; 0000000 00000001 : 500 ns; 1111111 00000000 : 500 ns;</pre>	
0111111 10000000 500 ms; 1011111 11000000 500 ms; 11101111 1110000 500 ms; 11110111 1111000 500 ms; 1111101 1111100 500 ms; 11111101 1111110 500 ms; 11111101 11111110 500 ms; 11111101 11111111 500 ms;	
F.21.3 inc8_structure.vhd	
 VHDL object: Entity "inc8" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ inc8:inc8") Generated on: Wed Jun 3 10:53:06 1998 Generated by: rjh6v Fourcated by: rjh6v/csis/mentor/B4/eddm/work/inc8/part Source from: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 	
LIBRARY STATEMENT LIBRARY ieee; PACKAGE STATEMENT USE ieee.stdlogic_l164.ALL; entity ine8 is PORT LIST	
<pre>port(</pre>	

•

4
Ū.
Ē.
=
Ω.
2
N.
2.2
TT.

F.22.1 dec4.vhd

library IEEE, ARITHMETIC; use IEEE.std_logic_l164.all; use ARITHMETIC.std_logic_arith.all;

entity dec4 is

port(Y: in INTEGER range -7 to +7; Z: out INTEGER range -8 to +6); end dec4;

architecture behave of dec4 is process(Y) begin Z <= Y - 1; end process; end behave; begin

 -- VHDL Object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ dec4/dec4_gen_med_s')ofentity'dec4* (componentinterface 'home/rjh6v/csis/mentor/B4/eddm/ work/dec4.edc4*)
 -- Generated On: Wed May 27 14:10:55 1998
 -- Generated Dy: 'h16v/csis/mentor/B4/eddm/work/dec4/dec4_gen_med_s
 -- Surre from: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 VHDL object: Entity 'dec4' (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/dec4')
 dec4:dec4')
 denerated on: wed May 27 14:10:55 1998
 Generated by: rjh6v
 rjh6v/sis/mentor/B4/eddm/work/dec4/part
 Source from: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 y : in std_logic_vector(3 downto 0); z : out std_logic_vector(3 downto 0) F.22.3 dec4_structure.vhd F.22.2 dec4_vectors.txt % Y Z : time; 0000 1111 : 500 ns; 1111 1110 : 500 ns; % 0001 0000 : 500 ns; 0010 0001 : 500 ns; 0100 0011 : 500 ns; 1000 0111 : 500 ns;); end dec4;

library ieee; library work; use ieee std_logic_l164.ALL; architecture structure of dec4 is - TYPE DECLARATIONS - SIGNAL DECLARATIONS - COMPONENT_DECLARATIONS component inv component inv component nor2 component nor2 - INLINE CONFIGURATIONS begin -- CONFONENT SIGNAL ASSIGNMENTS -- CONFONENT INSTANTIATIONS end structure;

NC4	F.23 ADDINC4 F.23.1 addinc4.vhd	<pre>a : in std_logic_vector(3 downto 0); b : in std_logic_vector(3 downto 0); d : out std_logic_vector(3 downto 0)</pre>
12 	library IEEE, ARITHMETIC; use IEEE.std_logic_1164.al; use abTHMMETIC etd locic arith all.	end addinc4;
	entity adding is culour at the set of the se	<pre>addinc4/addinc4_gen_low_s*)ofentity*addinc4*(componentinterface*/home/rjh6v/csis/mentor/ B4/addm/xork/addinc4:addinc4*) Generated on: FriJun 5 10.26:56 1998</pre>
52 Main and Main a	A: in std_logic_vector(3 downto 0); B: in std_logic_vector(3 downto 0); D: out std_logic_vector(3 downto 0)	 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/addinc4/addinc4_gen_low_s Program: VHDLwrite v8.5_2.4 Med Feb 5 16:56:22 PST 1997
15 15 <td< td=""><td>end addinc4; erchitecture behave of addinc4 is</td><td>library ieee; library work; use sef loric 1164 att.</td></td<>	end addinc4; erchitecture behave of addinc4 is	library ieee; library work; use sef loric 1164 att.
15.04 10.04 <td< td=""><td>Degin Process (OP, A, B) Decin</td><td>architecture structure of addinc4 is SIGNAL DECLARATIONS</td></td<>	Degin Process (OP, A, B) Decin	architecture structure of addinc4 is SIGNAL DECLARATIONS
15: 1	if OP = '0' then D <= A + B; close	COMPONENT_DECLARATIONS component and2 component inv
52 addited_vectors.nt. 7.3.3 addited_vectors.nt. 1.3.3 addited_vectors.nt. 7.3.0 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.1 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.2 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.1 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.2 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.3 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.3 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.1 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.1 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.1 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.2 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.3 addited_vectors.nt. 1.3.1 addited_vectors.nt. 7.3.4 med_reb_r_s_files(-reference.nt.) 1.3.1 addited_vectors.nt. 7.4 med_reb_r_s_f_s_s_s_s_s_s_s_s_s_s_s_s_s_s_s_s_s	D <= A + "0001"; end if;	component nard2 component nor2 commonent or2
52 0000 <t< td=""><td>end behave; end behave; F.23.2 addinc4 vectors.txt</td><td>component xor2 INLINE CONFIGURATIONS herin</td></t<>	end behave; end behave; F.23.2 addinc4 vectors.txt	component xor2 INLINE CONFIGURATIONS herin
57 0000 0000 0000 000 000 000 000 000 000	6 OP A B D : time :	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS
0001 1111 0000 500 ms; 01010 1010 1010 500 ms; 0100 1010 1010 500 ms; 0100 1010 1010 500 ms; 0100 1010 1010 500 ms; 0111 1010 XXX 001 500 ms; 0111 1111 XXX 001 500 ms; 1101 XXX 1101 XXX 1101 1101 XXX 1101 500 ms; 1101 1110 XXX 1101 500 ms; 1101 1111 XXX 1101 XXX 1101 1110 XXX 1101 500 ms; 1101 1111 XXX 1101 XXX 1101 1110 XXX 1101 500 ms; 1101 1111 XXX 1101 500 ms; 1000 1111 XXX 1101 200 ms; 1000 1111 XXX 1101 201 ms; 1000 1111 XXX 1101 201 ms;	© ADD 0 0000 0000 111111 : 500 ms; 0 0000 1111 1111 : 500 ms; 0 1111 1111 1110 : 500 ms; 0 1111 1111 1110 : 500 ms;	end structure;
1 1011 XXXX 0001 : 500 ms; 1 10111 XXXX 0000 : 500 ms; 1 1011 XXXX 1100 : 500 ms; 1 1011 XXXX 1100 : 500 ms; 1 1011 XXXX 1100 : 500 ms; 1 1011 XXXX 1110 : 500 ms; 1 1110 XXXX 1111 : 500 ms; 1110 XXXX 1111 : 500 ms; 1010 stock	& 0 0001 1111 0000 : 500 ms; 0 1111 0001 0000 : 500 ms; 0 0101 0101 : 500 ms; 0 1010 1010 0100 : 500 ms;	
10111 XXXX 1000 : 500 ms; 11011 XXXX 1100 : 500 ms; 11110 XXXX 1110 : 500 ms; 11110 XXXX 1111 : 500 ms; 11110 XXXX 1111 : 500 ms; F.23.3 addinc4 structure.vhd vHBL04 (component interface 'home/rjh6v/csis/mentor/B4/eddm/work/ WBL04 10:26:56 1998 Generated on: Fri Jun 5 10:26:51 1997 Billinda, refer: VHDLwrite v8:5_2.14 Wed Feb 5 16:56:22 FST 1997 LIBRANT ieee; USE ieee-std_Iogic_1164 ALL; FORT ISF	& B INC 1 0000 XXXXX 0001 : 500 ms; 1 1111 XXXXX 0000 : 500 ms;	
<pre>F.23.3 addinc4_structure.whd VHDLobject: Entity "addinc4" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ addinc4:addinc4") Cenerated on: Fri Jun 5 10:26:56 1998 Cenerated on: Fri Jun 5 10:26:56 1998 Cenerated by: Pin6v/ Cenerated by: Pin6v/csis/mentor/B4/eddm/work/addinc4/part Porce from: NubLerite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Port in VHDLerite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 ULBRARY ieee: USE ieee std_logic_l164.ALL; PoRT Lin6 port in std logic; PoRT Lin6 port in std logic;</pre>	8 1 0111 XXXX 1000 : 500 ms; 1 1011 XXXX 1100 : 500 ms; 1 1101 XXXX 1110 : 500 ms; 1 1102 VXXX 1111 : 500 ms;	
	F.23.3 addinc4_structure.vhd	
LIBRARY ieee ieee. LUE ieee ieee. ULE ieee ieee. ULE ieee ieee. PORT LIST PORT LIST PORT (in std logic:	 	
	LIBRARY ieee; USS ieee.std logic_1164.ALL; entity addinc4 is PORT LIST port(op : in std_logic;	25

F.24 PLUS3	Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
F.24.1 plus3.vhd library IEBE, ARITHWETIC; use IEBE.std_logic_l164.all;	<pre>library ieee; library work; use ieee.std_logic_l164.ALL; architecture structure of plus3 is</pre>
use ARTTHMETIC.std_logic_arith.all; entity plus3 is port(- TITRE DECLARATIONS SIGNL DECLARATIONS COMPONENT_DECLARATIONS component and2
Y: in std_logic_vector(4 downto 0); Z: out std_logic_vector(4 downto 0)); end plus3;	component inv component or2 INLINE CONFIGURATIONS howin
<pre>architecture behave of plus3 is begin process(Y) begin z <= Y + 00011'; end process;</pre>	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure;
F.24.2 plus3_vectors.txt	
& Y Z : time; 00000 00011 : 500 ns; 11111 00010 : 500 ns;	
% 11100 11111 : 500 ns; 10011 11100 : 500 ns; 100111 01010 : 500 ns; 001110 10001 : 500 ns;	
F.24.3 plus3_structure.vhd	
 VHDL object: Entity "plus3" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ plus3:plus3") Generated on: Wed Aug 5 14:25:42 1998 Generated by: rjh6v Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/plus3/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 	
LIBRARY STATEMENT LIBRARY STATEMENT PACKAGE STATEMENT USE ielee.std_logic_1164.ALL; entity plus3 is PORT LIST	
<pre>port(</pre>	
 VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/plus3/plus3/plus3.plus3 vork/plus3.plus3 of entity "plus3(component interface "/home/rjh6v/csis/mentor/B4/eddm/work/plus3.plus3 vork/plus3.plus3 vork 	

PLUS3

F.25 MINUSS	Source from: /howe/rjh6v/csis/mentor/B4/eddm/work/minus/minus/minus/minus/minus/minus/minus/minus/minus/minu
F.25.1 minusS.vhd	library ieee;
library IEEE, ARITHMETIC; use IEEE.std_logic_lid4.all; ee Abrumentc std_lodic_lid4.all;	use ieee.std_logic_1164.AL <i>i</i> ; architecture structure of minus5 is
entity minus5 is	TYPE DECLARATIONS SIGNAL DECLARATIONS COMPONEMEN DECLARATIONS
port(Y: in std_logic_vector(5 downto 0); 7: nut std_logic_vector(5 downto 0);	component and2 component inv
); end minus5;	component nor2 component nor3 component or2
architecture behave of minus5 is begin	COMPONENT XOT2 INLINE CONFIGURATIONS
process (Y) begin $z \le Y - 00101$; end process; end behave:	Degin - CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure;
F.25.2 minus5_vectors.txt	
& Y Z : time; 000000 111011 : 500 ns; 111111 111010 : 500 ns;	
8 000101 000000 : 500 ms; 001010 000101 : 500 ms; 010100 100011 : 500 ms; 010000 00110 : 500 ms; 010001 00110 : 500 ms;	
F.25.3 minus5_structure.vhd	
 VHDL object: Entity "minus5" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ minus5:minus5") Generated on: Wed Aug 5 14:30:28 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/minus5/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 	
LIBRARY STATEMENT LIBRARY STATEMENT PACKAGE STATEMENT USE ieee std_logic_1164.ALL; entity minus5 is PORT LIST	
<pre>port(y : in std_logic_vector(5 downto 0); z : out std_logic_vector(5 downto 0)</pre>	
); end minus5;	

MINUS5

F.26 PLUS25 F.26.1 plus25.vhd	Generated on: Wed Jul 29 15:03:17 1998 Generated by: rjh6v Source from: /hom/rjh6v/csis/mentor/B4/eddm/work/plus25/plus25_gen_low_s Program: /HDNarite v8.5,2.4 Wed Feb 5 16:56:22 PST 1997
library IEEE, ARTTHMETIC; use IEEE.std_logic_1164.all; use ARTTHMETIC.std_logic_arith.all;	library work:
entity plus25 is	use ieee.std_logic_l164.ALL; architecture structurel of plus25 is wore ner.nawrrow
port(Y: in INTEGER range -128 to +102; Z: out INTEGER range -103 to +127	TIPE DECLARATIONS SIGNAL DECLARATIONS COMPONENT_DECLARATIONS
); end plus25;	component and2 component and3 commonant inv
architecture behave of plus25 is bedin	component mand component nor2
process (Y) begin Z <= Y + 25;	COMPORENT OR2 COMPORENT VO22 INLINE CONFIGURATIONS
end process; end behave;	Degin CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS
F.26.2 plus25_vectors.txt	end structurel;
8 Y Z : time; 00000000 00011001 : 500 ns;	F.26.4 plus25_structure2.vhd
11111111 00011000 : 500 ns; 8	
11100110 11111111 : 500 ms; 11001101 11100110 : 500 ms;	plus25/plus25.tri7a_s') of entity "plus25" (component interface "/home/rjh6v/csis/mentor/B4/ eddm/ork/plus25:plus25:
10011011 1011000 : 500 ms; 00110111 01010000 : 500 ms;	Generated VII. med VII. 22 13:03:43 1370 Generated VII. ThinkV Source from: /home/r/reis/mentor/BA/sddm/incvt/n]us35.n]us35.vi173 s
01101110 10000111 : 500 ms; 11011100 11110101 : 500 ms;	BOORTAM: / HOUME/LINGY/SALYMENCOLOR/PA/COUNTWOLN/PIDS27/PIDS27/LIZG_S
10111001 11010010 : 500 ns; 01110011 10001100 : 500 ns;	library ieee; library work;
F.26.3 plus25_structure1.vhd	use ieee.std_logic_l164.ALL; architectures structures of plus25 is - TYPE DECLADATIONS
VHDL object: Entity "plus25" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/	SIGNAL DECLARATIONS COMPONENT_DECLARATIONS
plus25:plus25") Generated on: Wed Jul 29 15:03:17 1998 Generated by: Tjh6v Generated by: Tjh6v/reis/menter/Rd/eddm/work/nlus25/nart	component and2 component inv component or2 TMI.NE CONFIGURATIONS
Program: VHDLmrite V8.5_2.4 Wed Feb 5 16:56:22 PST 1997	begin CONCURRENT SIGNAENTS
LIBRARY STATEMENT LIBRARY ieee;	COMPONENT INSTANTIATIONS end structure2;
FACRAGE STATEMENT USE ieee.std_logic_1164.ALL; entity plus55 is	
PORT LIST port (
<pre>y : in std_logic_vector(7 downto 0); z : out std_logic_vector(7 downto 0)</pre>	
); end plus?5;	
- VIDL object: Architecture "structure" (schematic "home/rjh6v/csis/mentor/B4/eddm/work/ plus25/plus25_gen_low_s")ofentity"plus25"(componentinterface'/home/rjh6v/csis/mentor/B4/ eddm/work/plus25:plus25")	

PLUS25

C.T. International interval int	Transv. ram. of month Transv. ram. of month Transv. ram. of month Transv. ram. of month <th>F.27 MINUS25</th> <th> Source from: /home/rjh6v/csis/mentor/B4/eddm/work/minus25/minus25_gen_low_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997 </th>	F.27 MINUS25	 Source from: /home/rjh6v/csis/mentor/B4/eddm/work/minus25/minus25_gen_low_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 FST 1997
Minery and Streamst	<pre>Mathematics and states and s</pre>	F.27.1 minus25.vhd	ibrary ieee;
attraction attraction attraction attraction attraction attraction <td><pre>alt of mundard is if in a transmer mase rish to +230; if in a transmer mase rish to +230; if in a transmer mase rish to +230; add in a transmer mase risk to +230; add in a tr</pre></td> <td><pre>library IEBE.stdlogic_logic_lib(all; use IEBE.stdlogic_lib(all; use ARTTMMETTC.std_logic_arith.all;</pre></td> <td>ibrary work; se ieee.stgl.logic_l164.ALL; rchitecture_structure_of minus25 is</td>	<pre>alt of mundard is if in a transmer mase rish to +230; if in a transmer mase rish to +230; if in a transmer mase rish to +230; add in a transmer mase risk to +230; add in a tr</pre>	<pre>library IEBE.stdlogic_logic_lib(all; use IEBE.stdlogic_lib(all; use ARTTMMETTC.std_logic_arith.all;</pre>	ibrary work; se ieee.stgl.logic_l164.ALL; rchitecture_structure_of minus25 is
 T. G. INTERNET FRAME (1996) T. S. G. INTERNET FRAME (1996) T. S. G. INTERNET FRAME (1996) T. S. G. MARTINA (1996) T. S. G. G. MARTINA (1996) T. S. G. G. MARTINA (1996) T. S. G. G.	<pre>% 10 control cont</pre>	entity minus25 is	TYPE DECLARATIONS - SIGNAL DECLARATIONS COMPONENT DECLARATIONS
and intensity: Second second	ted intensity ted intensity termination t	Viti INTEGER range -103 to +127; Z: out INTEGER range -128 to +102	component and2 component and3 component inv
Answer	Second if	end minus25;	component nor2 component nor3 component or3
and the second s	arguing	arcinteccure benave of minus23 is begin bgin process (Y)	component xor2 INLINE CONFIGURATIONS herin
F.7.1 minus25_vectors.Mt V a 1 imit W 1 1111111110111 500 mt W 1 1111111111111111111111111111111111	F.7.1 minus.5. vectors.kt W 0 0 1 11811 M 0 0 1 11811 M 10000000 500 mm M 10000000 500 mm M 10000000 100 mm M 10000000 100 mm M 10000000 100 mm M 10000000 100 mm M 1000000000 100 mm M 10000000 1000000 100 mm M 10000000 1000000 100 mm M 10000000 1000000 100 mm M 10000000 10000000 1000000 M 10000000 10000000 1000000 M 100000000 10000000 100000000000000000	Degin 2 < - 25; end process; end behave;	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure;
000 000 000 000 000 000000000000000000000000000000000000	MY 24 (11) 11, 110 MY 24 (11) 11, 111 MY 130 (11) 100 (11) MY 140 (11) 110 (11)	F.27.2 minus25_vectors.txt	
0001001 0001000 500 mt 0001000 000 mt 500 mt 0001000 0001111 500 mt 0010001 0001100 500 mt 0010001 0001100 500 mt 0010001 0001100 500 mt 0010010 0001110 500 mt 0010010 500 mt 3154527 Mt 0010010 510 mt 5156527 Mt 0010000 5150 mt 5156527 Mt 0010000 5151 mt 5156527 Mt 00100000 5151 mt 5156527 Mt 00100000 5150 mt 5155527 Mt 00100000 5150 mt 5155527 Mt 00100000000 51565	0.0011001 0001000 000 mt 0.0010001 000 mt 000 mt 0.0010001 000 mt 000 mt 0.00010001 000 mt 000 mt 0.00010001 000 mt 000 mt 0.00010001 000 mt 000 mt 0.0001000 1000 mt 000 mt 0.0001000 1000 mt 100 mt 0.0001000 1000 mt 100 mt 0.0001000 1000 mt 1000 mt 0.00010000 1000 mt 1000 mt 0.00010000 1000 mt 1000 mt 0.00000000 1000 mt 1000 mt 0.00000000000000000000000000000000000	& Y Z : time; 0000000 11100111 : 500 ns; 1111111 11100110 : 500 ns;	
10000001 0000011 500 ms; 10000010 0000110 500 ms; 0000010 0001100 500 ms; 0000010 0001100 500 ms; 0000010 001100 154.57 0000010 00100 154.57 0000010 00100 154.57 0000010 00100 154.56 0000010 001000 154.56 0000010 001000 156.52 0000010 001000 156.52 0000010 00100000 156.55 0000010 00100000 156.55 0000010 001000000 156.55 0000010 0010000000 156.55 0000010 0010000000000000000000000000000000000	University 500 min. 0000010 00111011 500 min. 0000010 00111011 500 min. 1000110 500 min. 514 st.71 1399 1000110 516 st.72 1399 515 st.71 399 10000111 510 min. 516 st.72 1397 10000111 510 min. 516 st.72 1397 10000111 510 min. 516 st.72 1397 100001111 510 min. 516 st.72 1397 100001111 510 min. 516 st.72 min.	\$ 00011001 00000000 : 500 ms; 00011001 00011001 : 500 ms;	
<pre>Double in the initial is and it is a second interface 'how'r:how'cais /heelm/work/ Double initial's furture.hd T=17.3 minus/5 structure.hd T=17.3 minus/5 structure.hd T=17.3 minus/5 structure.hd T=17.4 minus/5 structure.h</pre>	<pre>figure in the interface interfa</pre>	01100100 0101011 : 500 ns; 11001000 10101111 : 500 ns; 10010001 01111000 : 500 ns;	
F.27.3 minus25_structure.vhd	F.27.3 minus25_structure.vhd -vulu.object: Entity minus25* (component interface */home/rjh6v/csis/mentor/B4/eddm/work/-vulu.object: Entity minus25* (component interface */home/rjh6v/csis/mentor/B4/eddm/work/-vulu.object: Entity minus25* (component interface */home/rjh6v/csis/mentor/B4/eddm/work/-ucreated on: rjh6v-ucreated on: rjh6v/csis/mentor/B4/eddm/work/minus25* (part-emersted on: rjh6v-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-emersted on: virulwrite v8.5_2.4 wed Feb 5 16:55:22 FBT 1997-entry minus2;-entry minus2;-optic file-volue of the file-volue v1.12-volue v1.	UDIOUULI UDUCULUU : 500 MS; 10000110 00101101 : 500 MS; 10001100 01110011 : 500 MS;	
<pre>UND.object: Entity "minus25' (component interface ' home/rjh6v/csis/mentor/B4/eddm/work/ minus25): minus25' (component interface ' home/rjh6v/csis/mentor/B4/eddm/work/ minus25): minus25' (component interface ' home/rjh6v/csis/mentor/B4/eddm/work/ - Generated by: rjh6v - Source from / home/rjh6v/csis/mentor/B4/eddm/work/minus25 psr 1997 - EntRANY SENTERN - ENTRANY SENTERNS - ENTRANY SENTERN - ENTRAN</pre>	- "UID. Object: Entity' minus23' (component interface 'home/rjh6v/csis/mentor/B4/eddm/work/ minus25:") may 3 15:45:27 1998 - Generated by: rjh6v - Source from : Money 19:45:27 1998 - Source from : Money 1997 - LIBARY SAVEREN - LIBARY SAVEREN - LIBARY 1660 - POOTCA - LIBARY 1660 - POOTCA - LIBARY 1660 - POOTCA - POOTC	F.27.3 minus25_structure.vhd	
<pre></pre>	<pre></pre>	VHDLobject: Entity "minus25" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/	
<pre>- Source from: Name/rih6v/csis/mentor/B4/eddm/work/mins25/part - Program: VHDwrite V8:5_1.4 Med Feb 5 16:56:22 BST 1997 - PROMOS TAYTEMENT LIBRANY ieles: - PACMOS TAYTEMENT USE ieles stal_logic_uector(7 domine 2) - PACMOS TAYTEMENT - Compression - Compression - Compression - Compression - Compression - Compression - Compression - Compression - Compression - PACMOS TAYTEMENT - PACMOS TA</pre>	<pre>- Source from 'nome/rjh6v/csis/mentor/B4/eddm/work/minus25/part - Program: ViDUartie Va.5_2.4 Wed Feb 5 16:56.22 PST 1997 - Program: ViDUartie Va.5_2.4 Wed Feb 5 16:56.22 PST 1997 LIBRARY iere: LIBRARY iere: Part ier is a context LIBRARY iere: Part iere: Part</pre>	minuszs:minuszs-1 Generated on: Mon Aug 3 15:45:27 1998 Commersted hu: rihtu	
LIBRARY STATEMENT DIRARY STATEMENT PACKAGE STATEMENT PACKAGE STATEMENT USE ieee std_logic_l164.ALL; USE ieee std_logic_vector (7 downto 0); y : in std_logic_vector (7 downto 0); y : in std_logic_vector (7 downto 0); y : : out std_logic_vector (7 downto 0); VHDL object: Architecture *structure *(schematic */home/rjh6v/csis/mentor/b4/eddm/work/ minus25:minus25:minus25: (componentinterface*/home/rjh6v/csis/mentor/ Generated by: rjh6v Generated by: rjh6v	<pre> LIBRARY STATERET LIBRARY STATERET PACKAGE STATERET USE ieee std logic_l164.ALL; Porr LIST Porr LIST port(</pre>	Source from: /home/rjh6v/csis/mentor/B4/eddm/work/minus25/part Source from: /home/rjh6v/csis/mentor/B4/eddm/work/minus25/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	
<pre> PackAGE STATEMENT USE ieles std_logic_l164.ALL; entity mius25 is PRT LIST port (</pre>	<pre> PACKAGE STATEMENT PACKAGE STATEMENT USE if see std_logic_vector(7 downto 0); y : in std_logic_vector(7 downto 0) z : out std_logic_vector(7 downto 0)); VHDL object: Architecture *structure* (schematic */home/rjh6v/csis/mentor/B4/eddm/work/ end minus25; VHDL object: Architecture *structure* (schematic */home/rjh6v/csis/mentor/B4/eddm/work/ B4/eddm/work/minus25 (componentinterface*/home/rjh6v/csis/mentor/ B4/eddm/work/minus25 15; Generated on: Mon Aug 3 15:45:27 1998 Generated by: rjh6v</pre>	LIBRARY STATEMENT LIBRARY STATEMENT	
<pre> PORT LIST port (y: (y: std_logic_vector(7 downto 0); z: out std_logic_vector(7 downto 0)); end minus25; </pre>	<pre> FORT LIST FORT LIST port(</pre>	PACKAGE STATEMENT USE : ree.std.logic_1164.ALL; Dotter.ee.std.logic_1164.ALL;	
<pre>Y: in std_logic_vector(7 downto 0); z: out std_logic_vector(7 downto 0)); end minus25; VHDL object: Architecture *structure *(schematic *home/rjh6v/csis/mentor/B4/eddm/work/ minus25/minus25_gen_low_s*)ofentity*minus25*(componentinterface*/home/rjh6v/csis/mentor/ enerated by: rjh6v</pre>	<pre>y: in std_logic_vector(7 downto 0); z: out std_logic_vector(7 downto 0)); end minus25; VHDL object: Architecture *structure (schematic *home/rjh6v/csis/mentor/B4/eddm/work/ minus25/minus25 gen_low_s*)ofentity*minus25*(componentinterface*/home/rjh6v/csis/mentor/ mentocrk/minus25; Generated by: rjh6v</pre>		
);); end minus25; VHDLoinest: Architecture *tructure (schematic '/home/rjh6v/csis/mentor/B4/eddm/work/ minus25.fminus25.gen_low_s')ofentity'minus25'(componentinterface'/home/rjh6v/csis/mentor/ B4/eddm/work/minus25:) Generated on: Mon Aug 3 15:45:27 1998 Generated by: rjh6v	z: in std_logic_vector(7 downto 0); z: out std_logic_vector(7 downto 0)	
VHDL object: Architecture 'structure' (schematic '/home/rjh6v/csis/mentor/B4/eddm/work/ minus25/minus25_gen_low_s')ofentity'minus25`(componentinterface'/home/rjh6v/csis/mentor/ af/eddmaverk/minus25:) Generated by: rjh6v Generated by: rjh6v	VHDL object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ minus25/minus25_gen_low_s")ofentity"minus25'(componentinterface"/home/rjh6v/csis/mentor/ B4/eddm/work/minus25:) Generated by: rjh6v Generated by: rjh6v); end minus25;	
= 4 deductor minus25:minus25:7 1998 - Edenerated by: rjh6v Generated by: rjh6v	= 4 / edemarkec //minus25.minus25.) - Edemarkec //minus25.minus25.7 1998 Generated by: rjh6v Generated by: rjh6v	VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ minus25/minus25_gen_low_s")ofentity"minus25" (componentinterface"/home/rjh6v/csis/mentor/	
		B4/Gedm/work/minus/s/s/ininus/s/ Generated on: Mon Aug 3 15:45:27 1998 Generated by: rjh6v	
		-	

<pre>11brary ieee; 11brary work; work; architecture attucture1 of sop1 is - TYPE DECLARATIONS - TYPE DECLARATIONS - SIGML DECLARATIONS - SIGML DECLARATIONS - COMPONENT_DECLARATIONS component or2 component or2 - INLINE CONFIGURATIONS begin - CONCURRENT SIGNAL ASSIGNMENTS - CONCURRENT SIGNAL ASSIGNMENTS med structure1; F.28.4 sop1_structure2.vhd</pre>	<pre> VHDL object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ sopl:sopl'sopl') sopl:sopl'sopl' sopl:sopl'sopl' sopl:sopl'sopl' Generated by: rjh6v Generated by: rjh6v/csis/mentor/B4/eddm/work/sopl/sop_nand_s Brogram: /home/rjh6v/csis/mentor/B4/eddm/work/sopl/sop_nand_s Program: /home/rjh6v/csis/mentor/B4/eddm/work/sopl/sop_nand_s Program: /home/rjh6v/csis/mentor/B4/eddm/work/sopl/sop_nand_s Brogram: /home/rjh6v/csis/mentor/B4/eddm/work/sopl/sop_nand_s Program: /home/rjh6v/csis/mentor/B4/eddm/work/sopl/sop_nand_s SignALDBCLARATIONS SignALDBCLARATIONS INLINE CONFINUMATIONS INLINE CONFICUMATIONS INLINE CONFI</pre>	DIKK
<pre>% SOP1 %1 sop1.vhd rrsry IEEE; rrsry IEEE; ity sop1 is port(</pre>	<pre>d process; behave; B.2 sopl_vectors.txt B.C D Z : time ; 1 0 : 500 ns; 1 1 : 500 ns; x 1 : 500 ns; x 1 : 500 ns; x 1 : 500 ns; x 2 : 500 ns; x 1 : 500 ns; x 2 : 500 ns; x 1 : 500</pre>	<pre>MARY ieee: MARY ieee: MARY ieee: MARY ieee: PacKadS TWTEMENT ity sop1 is ity std_logic_l164.ALL;</pre>

VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ sop4/sop4_auto_s")ofentity'sop4"(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/work/
sop4:sop4.) Generated on: Tue May 5 14:49:47 1998

SOP4

. •
	hd library vice; library vice; library with use ieee.std_logic_l164.ALt; use ieee.std_logic_l164.ALt; use ieee.std_logic_ std_logic library vice; library with architecture stducturel of posl is - TFP DECLARATIONS - SIGNAL DECLARATIONS - SIGNAL DECLARATIONS - CONFORMENT DECLARATIONS - CONFORMENT DECLARATIONS - CONFORMENT SIGNAL ASSIGNMENTS - CONFORMENT INSTANTIATIONS end structurel; posl is F.30.4 posl_structure2.vhd	 a) bok (C 008 D); b) bok (C 008 D); c) bok (C 008 D);<	on: Thu Apr 30 12:44:54 1998 by: rjh6v m: /home/rjh6v/csis/mentor/B4/eddm/work/pos1/pos1_auto_s
F.30 POS1	<pre>F.30.1 posl.whd library IEEE; use IEEE.std_logic_l164.all; entity posl is port(</pre>	<pre>begin construction begin constructions constructions constructions construction constructin</pre>	

•

POS1

<pre>F.31 GT F.31 gt.vhd library IEEE: std_logic_l164.all; use IEEE.std_logic_vector(1 downto 0);</pre>	<pre>Architecture structure1 of gt is TYPE DECLARATIONS SectanATIONS Secondoment and component inv component inv component inv component or2 inture component or2 component or</pre>
F.31.2 gl_vectors.txt * A B GT : time ; 00 0X 0 : 500 ns; 01 1 : 500 ns; 01 10 0 : 500 ns; 11 10 0 : 500 ns; 11 11 0 : 500 ns; F.31.3 gl_structurel.vhd enerated by: rjh6v e. Generated by: rjh6v e. Source from: vhome/rjh6v/csis/mentor/B4/eddm/work/gt.gt* e. Program: vhome/rjh6v/ssis/mentor/B4/eddm/work/gt.gt*	<pre>library iee; library work; use ieee.std_logic_l164.ALL; architecture structure2 of gt is - TYPE DECLARATIONS - TYPE DECLARATIONS - COMPONENT DECLARATIONS - COMPONENT and3 component inv component inv component inv component inv component or2 - INLINE CONFIGURATIONS Begin - CONCURRENT SIGNAL ASSIGNMENTS - CONCURRENT SIGNAL ASSIGNMENTS Begin - CONCURRENT INSTANTIATIONS end structure2; F.31.5 gf_structure2;</pre>
<pre> LIBRARY STATEMENT LIBRARY dees PACKAGE STATEMENT USE ieee.std_logic_l164.ALL; usE ieee.std_logic_vector(1 downto 0); ort(</pre>	VHDLobject: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/gt/ gt_gen_low_s" lofentitygt"(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/work/gt:gt") - Generated on: Thu Apr 30 13:33:54 1998 - Generated Dy: Th6V - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/gt/gt_gen_low_s - Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Ned Feb 5 16:56:22 PST 1997 - Source from: /home/rjh6v/csis/mentor/B4/eddm/work/gt/gt_gen_low_s - Program: VHDLwrite v8.5_2.4 Ned Feb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Ned Feb 5 16:56:22 PST 1997 - From the structure of gt is - StGNAL DSCLARATIONS - SIGNAL DECLARATIONS - SIGNAL DECLARATIONS - COMPONENT_DECLARATIONS - COMPONENT_DECLARATIONS - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS
library ieee; library work; use ieee.std_logic_1164.ALL;	Degin CONCURRENT SIGNAL ASSIGNMENTS CONFONENT INSTANTIATIONS end structure3; end structure3;

•

. •

VHDL object: Architecture "structure" (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ VHDL object: Architecture "structure" (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ O	 xor4:xor4* xor4:xor4* Cenerated on: Fri May 8 10:35:14 1998 Generated on: Fri May 8 10:35:14 1998 Generated by: rjh6v Fource from: /home/rjh6v/csis/mentor/B4/eddm/work/xor4/xor4_auto_s Program: /HDMwrite v8.5_2.4, Wed Feb 5 16:56:22 PST 1997 	library ieee; library work; use isto_logic_l164.ALL; architecture structurel of xor4 is	TYPE DECLARATIONS SICMAL DECLARATIONS COMPONERT DECLARATIONS - COMPONENT DECLARATIONS	INLIME CONFIGURATIONS bein CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structurel;	F.32.5 xor4_structure2.vhd VHDL object: Architecture * schematic * /home/rjh6v/csis/mentor/B4/eddm/work/ VHDL object: Architecture * schematic * /home/rjh6v/csis/mentor/B4/eddm/work/xor4/sor4c2 s*1 of Entity * sor4*	<pre>xor4:xor4") Generated on: Fri May 8 10:48:50 1998 Generated by: /home/rjh6v/csis/mentor/B4/eddm/work/xor4/xor4_c2_s Source from: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997</pre>	<pre>library ieee; library work; use ieee.std_logic_1164.ALL; architecture a structure2 of xor4 is propreservements</pre>	SIGNAL DECLARATIONS COMPONENT DECLARATIONS INLINE CONFIGURATIONS INLINE CONFIGURATIONS	Degin 			268
F.32 XOR4	F.32.1 XOF4.VNG Library ieee; Use ieee.std_logic_1164.all; entity XOR4 is Dort(<pre>A, B, C, D : in std_logic; Z : out std_logic); end XOR4;</pre>	architecture behave of XOR4 is begin $Z \le A$ XOR B XOR C XOR D; end behave;	F.32.2 xor4_vectors.txt % ABCD Z : time ;	0001 1 : 500 ms; 0010 1 : 500 ms; 0011 0 : 500 ms; 0100 1 : 500 ms; 0100 1 : 500 ms; 0100 0 : 500 ms;	1100 0 : 500 пс; F.32.3 xor4_vectors.opt	& ABCD Z: time; 0000 0: 500 ns; 0111 1: 500 ns; 1001 0: 500 ns; 1110 1: 500 ns;	F.32.4 xor4_structure1.vhd VHDL object: Entity 'xor4' (commonent interface '/home/rih6v/csis/mentor/B4/eddm/work)	<pre>xor4:xor4') Generated on: Fri May 8 10:35:14 1998 Generated by: rjh6v Cenerated by: rjh6v/csis/mentor/B4/eddm/work/xor4/part Fource from: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997</pre>	LIBRARY STATEMENT LIBRARY ieee: PACKAGE STATEMENT USE ieee.std_logic_l164.ALL; entity xord is PORT LIST	<pre>port(</pre>	

F.32.6 xor4_structure3.vhd

F.33 XOR5	architecture structure1 of xor5 is TYPE DECLARATIONS O
F.33.1 xor5.vhd	SIGNAL DECLARATIONS COMPONENT_DECLARATIONS
Library ieee; Use ieee.std_logic_1164.all;	component xor2 INLINE CONFIGURATIONS havin
entity XOR5 is port(CONCURRENT SIGNAL ASSIGNMENTS
A, B, C, D, E : in std_logic; Z : out std_logic	COMPONENT INSTANTIATIONS end structurel;
); end XORS;	F.33.4 xor5_structure2.vhd
architecture behave of XOR5 is	/ drani/ mbha/ NB/ radanani a faranani a a samanani a samanani a samanani a samanani a samani a samani a saman
DEGIN Se A XOR B XOR C XOR D XOR E; end hehave:	VHOL ODJECT: AFCATTECTURE "STLUCTURE 'SCHEMALLC' / NOME/I JUOV/CAIS/MENLOT/DA/CAUM/WOLK/ xor5/xor5_c2_s*)ofentity*xor5*(componentinterface*/home/rjh6v/csis/mentor/B4/eddm/work/ vorc5*vor5*)
F.33.2 xor5_vectors.txt	
% ABCDE Z : time ;	Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
00000 0 : 500 ns; 10110 1 : 500 ns;	library ieee; library work:
01011 1 : 500 ms/ 11101 0 : 500 ms/	use ieee.std_logic_1164.ALL; architecture structure2 of xor5 is
F.33.3 xor5_structure1.vhd	TYPE DECLARATIONS SIGNAL DECLARATIONS
in a start a sta start a start	
VHDL object: Entity 'xor5' (component interface '/nome/rjnov/csis/mentor/pa/eddm/work/ xor5:xor5')	INLINE CONFIGURATIONS begin
Generated on: Fri May 8 11:04:14 1998 Generated bv: rihév	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS
Source from: /home/rjh6v/csis/mentor/B4/eddm/work/xor5/part Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	end structure2;
LIBRARY STATEMENT	F.33.5 xor5_structure3.vhd
LIBRARY ieee; Package Statement	/ matheway (201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201 / 201
USE ieee.std_logic_1164.ALL;	VHUL ODJECC: ALCHITECCUTE SLUCLULE ISCHEMALLY /HOWE/LJHOV/CSEA/MELLOV/DF/PC/DE/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/CSEA/MELLOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/CSEA/MELLOV/DF/ACHEMALLY /ACME/LJHOV/CSEA/MELLOV/CSEA/MELMA/MELLOV/CSEA/MELLOV/CSEA/MELMA/MELLOV/CSEA/MELMA/MELDOV/CSEA/MELMA/MELDOV/CSEA/MELMA/MELDOV/CSEA/MELMA
entity XOC5 is PORT LIST	schem/xor5_nand:xor5_nand") Generated on: Fri May 8 12:42:25 1998
port(a : in std_logic;	Generated by: rjh6v course from: /homesrih6v/reis/mentor/BA/schem/xor5 nand/schematic
b : in std_logic;	Bource Liou: / Hower's Jury/Cars/Mencor/Par/Acteur/Act_Adams/ Sourcements Program: VHDLMrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
d : in std_logic;	library ieee;
e : in std_logic z : out std_logic	library work; use ieee.std looic 1164.ALL;
); end xor5;	architecture structure3 of xor5 is
	ILLE DECLARATIONS SIGNAL DECLARATIONS
VHDL object: Architecture *structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ xor5/xor5 auto.s")ofentity*xor5*(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/work/	COMPONENT_DECLARATIONS component nand2 reminimations
xor5:xor5")	INCINE CONFIGURATIONS begin
Generated on: rii may billowii 4 1970 Generated by: rjh6v	COMPONENT INSTANTIATIONS end structure3;
Source from: /home/rjh6v/csis/mentor/84/eddm/work/xors/xors_auco_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	
library ieee;	. 2
ildrary work; use ieee.std_logic_1164.ALL;	270
·	·

F.33.6 xor5_structure4.vhd

.

•

·

<pre>Mark field: Mark field:</pre>	F.34 ABS4 F.34.1 abs4.vhd	component nand2p component or2 component xnr2 TNLTNE CONFICURATIONS
And and state F.44 and structure And structure and state F.44 and structure And structure and state F.44 and structure And structure F.44 and structure <td><pre>library ieee: use ieee.std_logic_l164.all; entiry ABS4 intrecer range -7 to +7; port(x: in INTEGER range 0 to +7); z: out INTEGER range 0 to +7);</pre></td> <td><pre>begin</pre></td>	<pre>library ieee: use ieee.std_logic_l164.all; entiry ABS4 intrecer range -7 to +7; port(x: in INTEGER range 0 to +7); z: out INTEGER range 0 to +7);</pre>	<pre>begin</pre>
<pre>Mode in the interval inte</pre>	end ABS4;	F.34.4 abs4_structure2.vhd
F.3.1 Jaby: vectorskitt ************************************	archtecture BEMAVE OI ABS4 IS begin begin 2 <= abs X; end BEHAVE;	 - VHDL object: Architecture 'structure' (schematic '/home/rjh6v/csis/mentor/B4/eddm/work/ abs//absval_gen_low_s')ofentity'abs4 (componentinterface'/home/rjh6v/csis/mentor/B4/eddm/ work/abs4:abs4") The May 12 14:41:34 1998 - Generated on: 'Th6v - Generated by: 'rjh6v - Source from: 'home/rjh6v/csis/mentor/B4/eddm/work/abs4/absval_gen_low_s'
0.0.0.1.111:500:00.1.1111:500:00.1.111:500:00.1.111:500:00.1.111:	F.34.2 abs4_vectors.txt	Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
F.3.3 abid_structure1.vtdi	<pre>% X Z : time; 0000 000 : 500 ns; 0111 111 : 500 ns; 1011 111 : 500 ns; 1010 110 : 500 ns; 1100 110 : 500 ns;</pre>	<pre>library work; use ieee.std_logic_l164.ALL; architecture structure2 of abs4 is SIGNAL DECLARATIONS COMPONENT_DECLARATIONS component and2 component buf</pre>
<pre></pre>	F.34.3 abs4_structure1.vhd	component or2 component xor2 INLINE CONFIGURATIONS
LIRRAW ieee: USR isee stats USR isee stats or is not logic_vector(1 downto 0); z out sed_logic_vector(2 downto 0); is not sed_logic_vector(2 downto 0); end as4; 	 Will object: Entity abs4 (component interface */home/rjh6v/csis/mentor/B4/eddm/work/ Will object: Entity abs4 (component interface */home/rjh6v/csis/mentor/B4/eddm/work/ Generated on: Wed May 13 14:59:21 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/abs4/part Program: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 	begin - CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure2;
<pre> YND.object: Architecture *eructure (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ abs4/abs4]low_s)ofBritlyPabs4'(componentitlerface'home/rjh6v/csis/mentor/B4/eddm/work/ abs4.abs4) abs4.abs4) abs4.abs4 abs4.abs4.abs4.abs4.abs4.abs4.abs4.abs4.</pre>	<pre>LIBRARY ieee; USE ieee.std_logic_l164.ALL; entity abs4 is std_logic_vector(3 downto 0); port(x in std_logic_vector(2 downto 0) z : out std_logic_vector(2 downto 0)); end abs4;</pre>	
<pre>library ieee; library work; use ieee studingic_l164.ALL; architecturel of abs4 is - SIGNAL DECLARATIONS - COMPONENT_DECLARATIONS component DACLANATIONS component buf component buf</pre>	 VHDL Object: Architecture "structure" (schematic "home/rjh6v/csis/mentor/B4/eddm/work/ abs4/absval_low_s 'lofentity'abs4"(componentinterface /home/rjh6v/csis/mentor/B4/eddm/work/ abs4:abs4.) Generated on: Wed May 13 14:59:21 1998 Generated by: rjh6v Sis/mentor/B4/absval_low_s Succe from: /home/rjh6v/csis/mentor/B4/eddm/work/abs4/absval_low_s Program: VHDLwrite V8.5_2.4 Wed Feb 5 16:56:22 FST 1997 	
	<pre>library ieee; library work; use ieee.std_logic_l164.ALJ; architecture structurel of abs4 is - SIGNAL DECLARATIONS - COMPONENT_DECLARATIONS component and2 component buf component inv</pre>	

ABS4

.

| F35. AUSS F35. AUSS F35.1 AUSS F35.1 AUSS F35.1 AUSS F35
 | 13.5. ANS 13.5. ANS 14.5. ANS 13.5. ANS 15.5. ANS 14.5. ANS 15.5. ANS
 | F35 ANS F35 ANS F31 abs/rid F34 abs/rid F33 abs/rid F34 abs/rid F34 abs/rid F34 abs/rid <td>F3 ARS F3 ARS F3 JARS F3 JARS F4 F3 JARS F3 JARS F5 JARS F5 JARS <td< td=""><td>F3 ANS F3 ANS F3 ANS F3 ANS F4 ANS F4 ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<></td></td<></td></td<></td></td<></td>
 | F3 ARS F3 ARS F3 JARS F3 JARS F4 F3 JARS F3 JARS F5 JARS F5 JARS <td< td=""><td>F3 ANS F3 ANS F3 ANS F3 ANS F4 ANS F4 ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<></td></td<></td></td<></td></td<> | F3 ANS F3 ANS F4 ANS F4 ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<></td></td<></td></td<>
 | E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<></td></td<> | E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<>
 | E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<> | E3: ANS E3: ANS <td< td=""></td<> | |

--

--

--
--
--
---|--
--|--|--
---|
| F35.5 ANSS Librory version F35.1 ANSS Librory version F45.1 ANS Librory version
 | E35 ANS E35 ANS E31 ANAMI E31 ANAMI E31 ANAMI
 | F35 ARS F35 ARS F31 abs/risk F31 abs/risk
 | E35 AR8 F35 AR8 F31 answert F31 answert F31 answert F31 answert <td>F3 ARS F3 ARS F3 ARS F3 ARS F31 abs/bid F31 abs/bid F31 abs/bid F31 abs/bid</td> <td>F3: ANS F3: ANS F3: ANS F3: ANS</td> <td>F3: ANS F3: ANS F3: ANS F3: ANS</td> <td>F3: ANS F3: ANS F3: ANS F3: ANS</td> <td>F3: ANS F3: ANS F3: ANS F3: ANS</td>
 | F3 ARS F3 ARS F3 ARS F3 ARS F31 abs/bid F31 abs/bid
 | F3: ANS F3: ANS | F3: ANS F3: ANS
 | F3: ANS F3: ANS | F3: ANS F3: ANS | |
| F32.5. ANSI
53.5. Laboration
in the new control in the intervention
of the intervention
o | 53. ANS 53. ANS 5.1. Solution 5.1. ANA 7.2. Solution 5.1. ANA 7.3. Solution 5.1. ANA 8.1. S

 | F35 ARS F35 ARS F31 biokondi F31 biokondi
 | F35 ARS
 | F3 ANS F3 ANS F3 ANS F3 ANS <td< td=""><td>F35 ARS F35 ARS F31 biokshot F31 biokshot F31 biokshot F31 biokshot</td><td>F35 ARS F35 ARS F31 biokshot F31 biokshot F31 biokshot F31 biokshot</td><td>F35 ARS F35 ARS F31 biokshot F31 biokshot F31 biokshot F31 biokshot</td><td>F35 ARS F35 ARS F31 biokshot F31 biokshot F31 biokshot F31 biokshot</td></td<>
 | F35 ARS F35 ARS F31 biokshot F31 biokshot | F35 ARS F35 ARS F31 biokshot F31 biokshot | F35 ARS F35 ARS F31 biokshot F31 biokshot
 | F35 ARS F35 ARS F31 biokshot F31 biokshot | |
| F35. ANS F35. ANS F35. Internet in the second of the sec | E35 ANS E35 ANS E35 ANS E31 Answer E31 Answer E31 Answer E31 Answer E31 Answer E31 Answer E31 Answer E31

 | F35 ARS F35 ARS F31 abs/red F31 abs/red F31 abs/red F31 abs/red <td>F35 ARS F35 ARS F31 answirt F31 answirt F31 answirt F31 answirt <td>F3 ARS F3 ARS F3 ARS F3 ARS F31 abs/bit F31 abs/bit F31 abs/bit F31 abs/bit</td><td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und </td></td></td></td></td> | F35 ARS F35 ARS F31 answirt F31 answirt F31 answirt F31 answirt <td>F3 ARS F3 ARS F3 ARS F3 ARS F31 abs/bit F31 abs/bit F31 abs/bit F31 abs/bit</td> <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und </td></td></td></td>
 | F3 ARS F3 ARS F3 ARS F3 ARS F31 abs/bit F31 abs/bit
 | F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und </td></td></td> | F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und </td></td> | F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und <td>F35 ARS F35 ARS F31 abs/und F31 abs/und F31 abs/und F31 abs/und </td> | F35 ARS F35 ARS F31 abs/und F31 abs/und
 | |
| F35.5 ANSS F35.5 ANSS F35.1 ANSS F35.1 ANSI F35.1 ANSI F35.1 ANSI F35.2 ANS F35.1 ANSI F35.2 ANS F35.1 ANSI F35.3 ANS F35.1 ANS
 | E3.3. ANSE
E.3.1. ANSI:
E.3.1. ANSI:
E.3.2. ANSI:
E.3.2. ANSI:
E.3.3. ANSI:
E.3.4.3.4.4.4.4.4.4.4.4.4.4.4.4.4.4.4.4.
 | F35 ARS F35 ARS F31 Interview F31 Interview F31 Interview
 | E35 AR8 E35 AR8 E31 M85/401 E31 M85/401 E31 M85/401 E31 M85/401 <td>F3 ANS F3 ANS F3 ANS F3 ANS F31 Answert F31 Answert F32 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert</td> <td>F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li</td> <td>F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li</td> <td>F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li</td> <td>F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li</td>
 | F3 ANS F3 ANS F3 ANS F3 ANS F31 Answert F31 Answert F32 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert F31 Answert
 | F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li | F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li | F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li
 | F3: ARS F3: ARS F3: Likety view F3: Likety view F3: Li | |
| F35.5 AUSS F35.5 AUSS F35.1 AUSS F35.1 AUSS F35.1 AUSS <td>E3.3. ANSE
E.3.1. MANAN E.3.1. MANAN E.3.1. MANAN E.3.1. MANAN Manan E.3.1. MANAN <tr< td=""><td>F35 ANS F35 ANS F35 ANS F35 ANS F15 ANS F35 ANS F35 ANS F34 ANS F34 ANS F34 ANS</td><td>F35 ANS F35 ANS F31 above F34 above F34 above</td><td>F3: ANS F3: ANS F3: ANS F3: ANS</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td></tr<></td> | E3.3. ANSE
E.3.1. MANAN E.3.1. MANAN E.3.1. MANAN E.3.1. MANAN Manan E.3.1. MANAN <tr< td=""><td>F35 ANS F35 ANS F35 ANS F35 ANS F15 ANS F35 ANS F35 ANS F34 ANS F34 ANS F34 ANS</td><td>F35 ANS F35 ANS F31 above F34 above F34 above</td><td>F3: ANS F3: ANS F3: ANS F3: ANS</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td><td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher</td></tr<>
 | F35 ANS F35 ANS F15 ANS F35 ANS F35 ANS F34 ANS F34 ANS F34 ANS
 | F35 ANS F35 ANS F31 above F34 above F34 above
 | F3: ANS F3: ANS
 | F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher
 | F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher | F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher | F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F35 answitcher F36 answitcher F36 answitcher F36 answitcher F37 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F33 answitcher F36 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher F34 answitcher
 | |
| F35.1 AU38 F35.1 AU38 F35.1 AU38 Farst and the farst and | R35 AR86 R31 R36 R32 R36 R32 R36 R32 R36 R32 R36 R32 R36 R32 R36 R33 R36 R33 R36 R33 R36 R33 R36 R34 R36 R35 R36 R36 R36 R

 | F35 ANS F35 ANS F35 ANS F34 ANS
 | F35 ANS F35 ANS F31 above F34 above F34 above
 | F3: ANS F3: ANS
 | F35 ANS
F35 ANS | F35 ANS
F35 ANS
 | F35 ANS
F35 ANS | F35 ANS
F35 ANS | |
| F35.1 ANS8 F35.1 ANS8 F35.1 ANS8 <td>K35 AK86 K31 Ak86 K32 Ak96 K44 Ak96 K44 Ak96 K44 Ak96 K44 Ak96</td> <td>F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 ANS F36 ANS F36 ANS F37 ANS F36 ANS F33 ANS F37 ANS F33 ANS F37 ANS F33 ANS F37 ANS F33 ANS F31 ANS</td> <td>F3 ARS F3 ARS F3 Last action F3 Last action F3 Last action F3 Last action <!--</td--><td>F3: ANS F3: ANS F3: ANS F3: ANS</td><td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td><td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td><td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td><td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td></td> | K35 AK86 K31 Ak86 K32 Ak96 K44 Ak96 K44 Ak96 K44 Ak96 K44 Ak96
 | F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F35 ANS F36 ANS F36 ANS F36 ANS F37 ANS F36 ANS F33 ANS F37 ANS F33 ANS F37 ANS F33 ANS F37 ANS F33 ANS F31 ANS
 | F3 ARS F3 ARS F3 Last action F3 Last action F3 Last action F3 Last action </td <td>F3: ANS F3: ANS F3: ANS F3: ANS</td> <td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td> <td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td> <td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td> <td>E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along</td>
 | F3: ANS F3: ANS
 | E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along
 | E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along | E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along | E35 ANS E35 ANS E33 along E33 along E33 along E34 along E34 along E34 along E35 along |
 |
| F3.5 ANS8 F3.5 ANS8 F3.5 ANS8
 | K35 AK86 K31 AK86 K32 AK86
 | F35 ANS F35 ANS F31 abs/nd F31 abs/nd F32 abs/nd F31 abs/nd F33 abs/nd F31 abs
 | F3 AR8 F3 AR8 F3 J MSH F3 J MSH F3 J MSH F3 MSH F3 J MSH F3 MSH F3 J MSH F3 MSH F3 MSH F3 MSH <td>F3 ANS F3 ANS F3 ANS F3 ANS F4 PAN F4 PAN <td< td=""><td>E3: ANS E3: ANS E3: ANS E3: ANS</td><td>E3: ANS E3: ANS E3: ANS E3: ANS</td><td>E3: ANS E3: ANS E3: ANS E3: ANS</td><td>E3: ANS E3: ANS E3: ANS E3: ANS</td></td<></td>
 | F3 ANS F3 ANS F4 PAN F4 PAN F4 PAN F4 PAN <td< td=""><td>E3: ANS E3: ANS E3: ANS E3: ANS</td><td>E3: ANS E3: ANS E3: ANS E3: ANS</td><td>E3: ANS E3: ANS E3: ANS E3: ANS</td><td>E3: ANS E3: ANS E3: ANS E3: ANS</td></td<> | E3: ANS E3: ANS
 | E3: ANS E3: ANS | E3: ANS E3: ANS | E3: ANS E3: ANS
 | |
| F3.5 AUSS F3.5 AUSS F3.51 AUSS F3.51 AUSS F3.51 AUSS
 | R35 AR86 R36 AR86 R36 AR86 R36 AR86 R36 AR86 R36 AR86 R36 AR86
 | F35 ANS
F36 ANS
F36 ANS
F36 Last factor
and set factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
factor
 | F3 ANS F3 ANS F4 ANS F3 ANS F3 ANS F4 ANS F4 ANS F4 ANS <td< td=""><td>F3 ANS F3 ANS F3 ANS F3 ANS F3.1 ANAMI F3.1 ANAMI F3.2 ANAMI F3.1 ANAMI F3.2 ANAMI F3.2 ANAMI F3.2 ANA F3.2 ANAMI F3.2 ANAMI F3.2 ANAMI</td><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<></td></td<></td></td<>
 | F3 ANS F3 ANS F3 ANS F3 ANS F3.1 ANAMI F3.1 ANAMI F3.2 ANAMI F3.1 ANAMI F3.2 ANAMI F3.2 ANAMI F3.2 ANA F3.2 ANAMI F3.2 ANAMI F3.2 ANAMI
 | E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<></td></td<> | E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<></td></td<> | E3: ANS E3: ANS <td< td=""><td>E3: ANS E3: ANS <td< td=""></td<></td></td<>
 | E3: ANS E3: ANS <td< td=""></td<> | |
| F35.3 ANS8 F35.4 ANS8 F35.1 Ansk/10 F35.1 Ansk/10 F35.2 Ansk/10 F35.1 Ansk/10 F35.2 Ansk/10 F35.2 Ansk/10 F35.3 Ansk/10 F35.2 A
 | 1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
 | F35 ARS F35 ARS F31 about F34 about F34 about
 | F3 ANS F3 ANS F3 ANS F4 ANS F3 ANS F4 ANS F3 ANS F4 ANS F3 ANS F4 ANS F3 ANS F5 ANS <td< td=""><td>F3 ANS F3 ANS F3.1 ANS F3.1 ANS F3.2 ANS F3.1 ANS F3.1 ANS F3.1 ANS F3.2 ANS<</td><td>E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3: F3: F3: F3: F3: F3: F3: F3: F3: F3:</td><td>E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3: F3: F3: F3: F3: F3: F3: F3: F3: F3:</td><td>E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3: F3: F3: F3: F3: F3: F3: F3: F3: F3:</td><td>E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3: F3: F3: F3: F3: F3: F3: F3: F3: F3:</td></td<>
 | F3 ANS F3 ANS F3.1 ANS F3.1 ANS F3.2 ANS F3.1 ANS F3.1 ANS F3.1 ANS F3.2 ANS<
 | E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3:
 | E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3: | E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3: | E3: ARS F3: ARS F3: La doction F3: La doction F3: La doction F3: La doction F3: La doction F3:
 | |
| F3S ANS F3S ANS F3S and fast fast fast fast fast fast fast fast
 | C3.5. ANS C3.5. ANS C3.5. Ans
 | F35 ARS F35 ARS F35 ARS F35 ARS F11 mark form for the factor of the f
 | F3: ARS F3: ARS F3: Also failed for the failed
 | F3 AR8 F3 AR8 F3.1 MeVinit F3.1 MeVinit F3.1 MeVinit F3.1 MeVinit F3.2 MeVinit F3.2 MeVinit F3
 | F35 ARS
 | F35 ARS | F35 ARS | F35 ARS | |
| F3S3 AN38 F35 AN38 F351 Juby Chill 110000 100000 10000 10000 10000 100000 10000 10000 10000
 | E3:5. ANS E3:5. ANS E3:5. ANS E3:5. ANS E1:5.1 ads/nd E1:5.1 ads/nd E1:5.1 ads/nd
 | F35 AR85 F351 AR85 F351 Ar85 F151 Ar85 F151 Ar85 F151 Ar85 F151 Ar85 F151 Ar85 F151 Ar85 F151 Ar85 F151 Ar85 F151 Ar85
 | E35 ANS E35 ANS E3.1 answind E45 ANS E15.1 answind E45 Answind E15.2 answind E45 Answind E15.3 answind E55 Answind E15.3
 | F3 ARS F3 ARS F3.3 Jakkind Face set_lot(::1); Face set_lot(::1); Alse set_lot(::1); Alses
 | F35 AR8 F35 AR8 F3.1 abs/red F35 abs/red F18.1 abs/red F35 abs/red F
 | F35 AR8 F35 AR8 F3.1 abs/red F35 abs/red F18.1 abs/red F35 abs/red F | F35 AR8 F35 AR8 F3.1 abs/red F35 abs/red F18.1 abs/red F35 abs/red F | F35 AR8 F35 AR8 F3.1 abs/red F35 abs/red F18.1 abs/red F35 abs/red F |
 |
| F353 AN38 F353 AN38 F351 AN38 F354 AN38 F1551 AN38 F1551 AN44 F1551 AN38 F1551 AN44 F1551 AN38 F1551 AN44 F1551 AN48 F1551 AN48 F1551 AN48 F1551
 | F3S ANS F3S ANS F3S and short F1000 (10000 (1000 (1000 (1000 (1000 (1000 (1000 (1000(
 | F35 ARS F35 ARS F33 abs/hd East abs/file East abs/file East abs/file East abs/file <td< td=""><td>E35 ANS E35 ANS E35 ANS E35 ANS E35 ANS E35 ANS E35 Answirt E35 Answirt<td>F3: A MSS F3: A Jack Harry and
in the text of the fact
and tex</td><td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td><td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td><td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td><td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td></td></td<> | E35 ANS E35 ANS E35 ANS E35 ANS E35 ANS E35 ANS E35 Answirt E35 Answirt <td>F3: A MSS F3: A Jack Harry and
in the text of the fact
and tex</td> <td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td> <td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td> <td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td> <td>F35 AR8 F35 AR8 F35 answitch F35 answitch F35 answitch F35 answitch</td>
 | F3: A MSS F3: A Jack Harry and
in the text of the fact
and tex
 | F35 AR8 F35 AR8 F35 answitch F35 answitch | F35 AR8 F35 AR8 F35 answitch F35 answitch
 | F35 AR8 F35 AR8 F35 answitch F35 answitch | F35 AR8 F35 AR8 F35 answitch F35 answitch | |
| F3.5 ANS8 F3.5 ANS8 F1.5.1 Above F1.5.1 Above F1.5.2 Above F1.5.1 Above
 | F35 ANS F35.1 ANS F15.3 ANS F15.4 ANS F15.5 ANS F15.4 Answer F15.5 ANS F15.4 Answer F15.5 ANS F15.5
 | F35 ANS F35 ANS F33 answitch F35 answitch Family rest and fam
 | E35 ARS F35 ARS F3.51 abs/nd F35 abs/nd Horn is the stall operating is the st
 | F3: Stands
 | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F1000 1000 1000 1000 1000 1000 1000 100
 | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F1000 1000 1000 1000 1000 1000 1000 100 | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F1000 1000 1000 1000 1000 1000 1000 100 | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F1000 1000 1000 1000 1000 1000 1000 100
 | |
| F33 ANS8 F34 ANS8 F35 ANS8 F34 ANS8 F35 ANS8 F34 ANS1 F34 ANS1 F34 ANS1 F35 ANS8 F34 ANS1 F34 ANS1 F34 ANS1 F34 A
 | E35 ANS E35 ANS E3.1 abs/nd Interve vert E18.3 abs/nd Interve vert E18.4 abs/nd Interve vert E18.5 abs/nd Intervert E18.5 abs/nd Interv
 | F35 ARS F35 ARS F35 ARS F35 ARS F18 F1 abs/rdf F18 abs/rdf F18
 | F35 ARS F35 ARS F35 Jabbind Harry weit Harry weit Harryw
 | F3: A MSS F3: S1 MSWMIG F3: S1 MSVMIG F1: S1: S1 MSVMIG F1: S1: S1 MSVMIG F1: S1: S1: MSVMIG F1: S1: S1: MSVMIG F1: S1: S1: MSVMIG F1: S1: S1: MSVMIG F1: MSVMIG F1: MSVMIG F1: MSVMIG F1: MSVMIG F1: MSVMIG <td>F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86<</td> <td>F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86<</td> <td>F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86<</td> <td>F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86<</td>
 | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86< | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86<
 | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86< | F35 AR8 F35 AR8 F35 AR8 F35 AR8 F35 Ar85 F35 Ar85 F35 Ar85 F35 Ar86 F35 Ar86< | |
| F35 AN38 F35 AN38 F35 AN38 F31 Initiany 1 and 1 F35 Initiany 1 and 1 F35 Initiany 1 and 1 F35 Initiany 1 and 1 F35 Initiany 1 F35 Initiany 1 F35
 | E35 AR8 11kerov (end) 11kerov (end) 11kerov (end) E31 abs/Ard 11kerov (end) 11kerov (end) 11kerov (end) E31 abs/Ard 11kerov (end) 11kerov (end) 11kerov (end) E32 abs/Ard 11kerov (end) 11kerov (end) 11kerov (end) E32 abs/Ard 11kerov (end) 11kerov (end) 11kerov (end) E32 abs/Ard 11kerov (end) 11kerov (end) 11kerov (end) E33 abs/Ard 11kerov (end) 11kerov (end) 11kerov (end) E44 abs/Ard <
 | F35 AR8 F35 AR8 F31 abs/rdf F31 abs/rdf F31 abs/rdf F31 abs/rdf <td>F3 AR8 F3 AR8 F3 abshind F3 abshind F3 abshind F3 abshind F3 abshind F4 abs eff. (apt. (att. att.)) F5 abshind F5 abshind <td>F3 AR8 F3 AR8 F31 Maximum F31 Maximum F31 Maximum F31 Maximum minimum F31 Maximum minimum</td><td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td><td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td><td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td><td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td></td> | F3 AR8 F3 AR8 F3 abshind F3 abshind F3 abshind F3 abshind F3 abshind
 F4 abs eff. (apt. (att. att.)) F5 abshind F5 abshind <td>F3 AR8 F3 AR8 F31 Maximum F31 Maximum F31 Maximum F31 Maximum minimum F31 Maximum minimum</td> <td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td> <td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td> <td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td> <td>F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd</td> | F3 AR8 F3 AR8 F31 Maximum F31 Maximum F31 Maximum F31 Maximum minimum
 | F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd
 | F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd | F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd | F35 ARS F35 ARS F35 ARS F34 Ars F31 abs/rdim F34 abs/rdim F34 abs/rd |
 |
| F33 ANS8 F33 ANS8 F33 ANS8 F34 ANS4 F33 ANS8 F34 ANS4 F34 ANS4 F34 ANS4 F34 A | F35 ADS ADS F31 abs/Add F31 abs/Add F32 abs/Add F32 abs/Add F32 abs/Add F32 abs/Add F32 abs/Add F32 abs/Add | F35 A03 F35 A03 F31 abs/And F31 abs/And F32 abs/And F32 abs/And <td>F35 ARS F35 ARS F31 abs/ndi F31 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F32 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F51 abs/ndi <td>F3_S_NNS F3_S_NNS F3_S_1 abs/Md F3_S_1 abs/Md Intervention Intervention Intervention Intervention</td><td>F35 ARS
F35 ARS
F35 answird
F35 answird</td><td>F35 ARS
F35 ARS
F35 answird
F35 answird</td><td>F35 ARS
F35 ARS
F35 answird
F35 answird</td><td>F35 ARS
F35 ARS
F35 answird
F35 answird</td></td> | F35 ARS F35 ARS F31 abs/ndi F31 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F32 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F31 abs/ndi F32 abs/ndi F51 abs/ndi <td>F3_S_NNS F3_S_NNS F3_S_1 abs/Md F3_S_1 abs/Md Intervention Intervention Intervention Intervention</td> <td>F35 ARS
F35 ARS
F35 answird
F35 answird</td> <td>F35 ARS
F35 ARS
F35 answird
F35 answird</td> <td>F35 ARS
F35 ARS
F35 answird
F35 answird</td> <td>F35 ARS
F35 ARS
F35 answird
F35 answird</td> | F3_S_NNS F3_S_NNS F3_S_1 abs/Md F3_S_1 abs/Md Intervention Intervention | F35 ARS
F35 ARS
F35 answird
F35 answird | F35 ARS
F35 ARS
F35 answird
F35 answird | F35 ARS
F35 ARS
F35 answird
F35 answird | F35 ARS
F35 ARS
F35 answird
F35 answird | |
| F35 ANBS F35 ANBS F31 Abs/Adit F31 Abs/Adit F32 Abs/Adit F31 Abs/Adit F31 Abs/Adit F31 Abs/Adit F31 Abs/Adit F31 Abs/Adit F32 Abs/Adit F31 Abs/Adit F32 Abs/Adit F32 Abs/Adit F32 Abs/Adit F
 | F35 ADS ADS F31 abs/And F31 abs/And F32 abs/And F31 abs/And F32 abs/And F31 abs/And F32 abs/And F31 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F33 abs/And F31 abs/And F33 abs/And F32 abs/And
 | F35 ADS F35 ADS F31 abs/ind F31 abs/ind F32 abs/ind F32 abs/ind F32 abs/ind F32 abs/ind F32 abs/ind F33 abs/ind F33 abs/ind F34 abs/ind F33 abs/ind F34 abs/ind F33 abs/ind F34 abs/ind F33 abs/ind F34 abs/ind <td>F3 AR8 F3 AR8 F3 AB8 F3 AB8 F3 Abbited F3 Abb</td> <td>F3_S_MBS F3_S_MBS F3_S_MBS F3_S_MBS F3_S_MBS F3_S_MBS Main F3_S_MBS <t< td=""><td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit </td></td></td></td></t<></td>
 | F3 AR8 F3 AR8 F3 AB8 F3 AB8 F3 Abbited F3 Abb
 | F3_S_MBS F3_S_MBS F3_S_MBS F3_S_MBS F3_S_MBS F3_S_MBS Main F3_S_MBS <t< td=""><td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit </td></td></td></td></t<>
 | F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit </td></td></td> | F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit </td></td> | F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit <td>F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit </td> | F35 ANS F35 ANS F31 abs/bit F31 abs/bit F32 abs/bit F32 abs/bit F32 abs/bit F33 abs/bit F32 abs/bit F33 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit F33 abs/bit F34 abs/bit |
 |
| F35.3 ANBS F35.3 ANBS F35.1 ANBS F35.1 ANSV F35.1 ANSV F35.1 ANV F35.1 ANSV F35.1 ANV F35.1 ANSV F35.1 ANV F35.1 ANSV F35.1 ANV F35.1 ANV F35.2 ANV F35.1 ANV F35.2 ANV F35.2 ANS F35.2 ANV F35.3 ANV F35.2 ANV F35.3
 | F35 ADS ADS F31 abs/And F31 abs/And F32 abs/And F31 abs/And F31 abs/And F31 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F32 abs/And F33 abs/And F32 abs/And F33 abs/And F32 abs/And F33 abs/And F31 abs/And F33 abs/And F32 abs/And F33 abs/And F32 abs/And F33 abs/And F32 abs/And F33 abs/And
 | F35 ABS F35 ABS F31 abs/red F31 abs/red F31 abs/red F31 abs/red F31 abs/red F31 abs/red F32 abs/red F32 abs/red F33 abs/red F33 abs/red F33 abs/red F33 abs/red F33 abs/red F33 abs/red F33 abs/red F33 abs/red F33 abs/red F34 abs/red F33 abs/red F34 abs/red F33 abs/red F34 abs/red F34 abs/red F34 abs/red <td>F3: ARS F3: ARS F3: JakSwid F3: JakSwid H: Statistical and statististical and statistististical and statistis</td> <td>F3_S_NNS F3_S_NNS F3_S_1_100_SUID F3_S_1_100_SUID T_S_1_100_SUID T_S_1_100_SUID T_S_2_100_SUID T_S_2_100_SUID T_S_2_100_SUID T_S_2_10_SUID T_S_2_100_SUID T_S_2_10_SUID</td> <td>F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd <</td> <td>F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd <</td> <td>F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd <</td> <td>F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd <</td>
 | F3: ARS F3: ARS F3: JakSwid F3: JakSwid H: Statistical and statististical and statistististical and statistis
 | F3_S_NNS F3_S_NNS F3_S_1_100_SUID F3_S_1_100_SUID T_S_1_100_SUID T_S_1_100_SUID T_S_2_100_SUID T_S_2_100_SUID T_S_2_100_SUID T_S_2_10_SUID
 | F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd < | F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd <
 | F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd < | F35 ANS F35 ANS F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F31 abs/hd F32 abs/hd F32 abs/hd F33 abs/hd F32 abs/hd F33 abs/hd F33 abs/hd F33 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F32 abs/hd F34 hd F33 abs/hd F34 hd < | |
| F35 AN88 F35.1 Million (Million (M | F35 AB8 F35 AB8 F31 abs/Adi F34 abs/Adi F33 abs/Adi F34 abs/Adi F34 abs/Adi F34 abs/Adi F34 abs/Adi F34 abs/Adi F35 abs/Adi F34 abs/Adi F34 abs/Adi F34 abs/Adi F35 abs/Adi F34 abs/Adi <td>F35 ARS F35 Ars</td> <td>F35 ABS F35 ABS F53 JabS/Hd F51 JabS/Hd T55 JabS/Hd Tester et
instruction of data is intervented in the structure instructing is inter</td> <td>F3 AB3 F3 AB3 F3 Law Mut Table Law Line L</td> <td>F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F</td> <td>F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F</td> <td>F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F</td> <td>F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F</td> | F35 ARS
 | F35 ABS F35 ABS F53 JabS/Hd F51 JabS/Hd T55 JabS/Hd Tester et instruction of data is intervented in the structure instructing is inter
 | F3 AB3 F3 AB3 F3 Law Mut Table Law Line L
 | F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F
 | F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F | F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F | F3_ANS F3_ANS F3_S1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F1.1 Abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.1 abs/bit F3.2 abs/bit F3.2 abs/bit F |
 |
| F35 AN88 F35.1 AN88 History Jeen:
International production of the state of the | F35 AB8 F35 AB8 F35 abs/rdit F35 abs/rdit F35 abs/rdit F35

 | F35 ARS
 | F35 ARS F35 ARS F3.1 abs/rdi F35.1 abs/rdi F3.1 abs/rdi F35.1 abs/rdi F35.1 abs/rdi F35.1 abs/rdi F35.2 abs/rdi F35.1 abs/rdi F35.2 abs/rdi F35.2 abs/rdi F35.2 abs/rdi <t< td=""><td>F3 AR8 F3 AR8 F.S.1 abs/hdi F.S.1 abs/hdi Historic Lister all Historic Lister all Historic Lister all Historic Listerallister all <t< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""></tr<></td></tr<></td></tr<></td></tr<></td></t<></td></t<>
 | F3 AR8 F3 AR8 F.S.1 abs/hdi F.S.1 abs/hdi Historic Lister all Historic Lister all Historic Lister all Historic Listerallister all <t< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""></tr<></td></tr<></td></tr<></td></tr<></td></t<> | F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""></tr<></td></tr<></td></tr<></td></tr<>
 | F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""></tr<></td></tr<></td></tr<> | F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""><td>F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""></tr<></td></tr<> | F35 ARS F35 ARS F3.1 abstrait F35 ars F3.1 abstrait F35 ars F35 ars F36 ars <tr< td=""></tr<> | |
| F35 AN88 F35 AN88 F35 AN84 F35 AN84 F35 AN84 F35 AN14 F35 AN14 F35 AN14 F35 AN14 F15 AN14 F16 AN14
 | F35 AR8 F35 Ar88 F351 abs/Add F351 abs/Add F31 abs/Add F31 abs/Add
 | F35 ARS F35 ARS F3.1 above F35.1 above F3.1 above F35.1 above F35 above F45.1
 | F3: ARS F3: ARS F3: J abs/rad F3: J abs/rad F3: J abs/rad
 | F3 AR8 F3 AR8 F.S.1 abs/vbd F.S.1 abs/vbd Historic Lister all Historic Lister all Historic Lister all Historic Listerallister all <t< td=""><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--></td></td></td></td></t<>
 | F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst </td <td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--></td></td></td> | F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst </td <td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--><td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--></td></td> | F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst </td <td>F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst<!--</td--></td> | F35 ARS F35 ARS F35 ARS F35 ARS F35 Arst F35 Arst F35 Arst F35 Ars F35 Arst </td
 | |
| F.3. ANS8 F.3. ANS8 F.3.1 Market issue: F.3.1 Market issue: F.3.1 Market issue: F.3.1 Market issue: F.3.1 Market issue: F.3.1 Market issue: F.3.2 Market issue: F.3.1 Market issue: F.3.3 Market issue: F.3.1 Market issue: F.3.1 Market issue: F.3.2 Market issue: F.3.2 Market issue: F.3.2 Market issue: F.3.3 Market issue: F.3.2 Market issue: F.3.3 Market issue: F.3.2 Market issue: F.3.3 Market issue: F.3.3 Market issue: F.3.3 Market issue: F.3.3 Market issue: F.3.3 Market issue: F.3.4 Market issue: F.3.4 Market issue:
 | F35 AB8 F35 AB8 F351 BA6/Md F351 BA6/Md F351 BA6/Md F350 Ba6/Md <td>F35 ARS F35 ARS F3.1 above F3.1 above F3.2 above F3.1 above F3.2 above F3.2 above F3.2 above F3.2 above F3.3 above F3.1 above F3.1 above F3.1 ab</td> <td>F3_ANS F3_ANS F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F15_S1 abs/bit F3_S1 abs/bit F15_S</td> <td>F3 AR8 F3 AR8 F.S.1 abk/ndi F.S.1 abk/ndi History Long History Long History Long History Long</td> <td>F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt</td> <td>F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt</td> <td>F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt</td> <td>F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt F3: Jakkwidt</td> | F35 ARS F35 ARS F3.1 above F3.1 above F3.2 above F3.1 above F3.2 above F3.2 above F3.2 above F3.2 above F3.3 above F3.1 above F3.1 above F3.1 ab
 | F3_ANS F3_ANS F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F3_S1 abs/bit F15_S1 abs/bit F3_S1 abs/bit F15_S

 | F3 AR8 F3 AR8 F.S.1 abk/ndi F.S.1 abk/ndi History Long History Long
 | F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt | F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt | F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt
 | F3: ARS F3: ARS F3: Jakkwidt F3: Jakkwidt | |
| F35.3 ANSS F35.3 ANSS F3.5.1 Abs/rdf F35.1 Abs/rdf F15.5.1 Abs/rdf F15.5.1 Abs/rdf F15.5 Abs/rdf F15.5.1 Abs/rdf F15.5 Abs/rdf F15.5.1 Abs/rdf F15.5 Abs/rdf F15.5 Abs/rdf <td>F35 AR8 F35 AR8 F33 a back with
a site set of logic_list, site
and sets
a contractions Library sets
a contractions F33 a back with
a site sets (logic_list, sit);
a contractions Library sets
a contractions F33 a back with
a site sets (logic_list, sit);
a contractions Library sets
a contractions F33 a back with
a site sets (logic_list, sit);
a contractions Library sets
a contractions F33 a back with
a contractions Contractions</td> <td>F35 AR8 F35 AR8 F35 Jakk F35 Jakk</td> <td>F3_ ARS F3_ ARS F.S.1 abs/bit F.S.1 abs/bit Hittery last Hittery last <t< td=""><td>F3_S_NS F3_S_NS F3_S_1_abs/rdi F3_S_1_abs/rdi F3_S_1_abs/rdi F3_S_1_abs/rdi F15_S_1_abs/rdi F</td><td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind </td></td></td></td></t<></td> | F35 AR8 F35 AR8 F33 a back with
a site set of logic_list, site
and sets
a contractions Library sets
a contractions F33 a back with
a site sets (logic_list, sit);
a contractions Library sets
a contractions F33 a back with
a site sets (logic_list, sit);
a contractions Library sets
a contractions F33 a back with
a site sets (logic_list, sit);
a contractions Library sets
a contractions F33 a back with
a contractions Contractions
 | F35 AR8 F35 AR8 F35 Jakk
 | F3_ ARS F3_ ARS F.S.1 abs/bit F.S.1 abs/bit Hittery last Hittery last Hittery last Hittery last <t< td=""><td>F3_S_NS F3_S_NS F3_S_1_abs/rdi F3_S_1_abs/rdi F3_S_1_abs/rdi F3_S_1_abs/rdi F15_S_1_abs/rdi F</td><td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind </td></td></td></td></t<>
 | F3_S_NS F3_S_NS F3_S_1_abs/rdi F3_S_1_abs/rdi F3_S_1_abs/rdi F3_S_1_abs/rdi F15_S_1_abs/rdi F
 | F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind </td></td></td> | F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind </td></td> | F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind <td>F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind F3: Jakkind F3: Jakkind </td> | F3: AR8 F3: AR8 F3: Jakkind F3: Jakkind
 | |
| F3.5. ANSS F3.5. ANSS F3.5.1 abs/rdf F3.5.1 abs/rdf F1.5.1 abs/rdf F3.5.1 abs/rdf F2.5.1 abs/rdf F3.5.1 abs/rdf F2.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5.1 abs/rdf F3.5
 | F35 ANS
 | F3: ANS F3: ANS F3: Janksvidi F3: Janksvidi F3: Janksvidi F5: Janksvidi F3: Janksvidi
 | F3_5_0_188 F3_5_0_188 F3_5_1_0_0 F3_5_1_0
 | F3 ADS F3 ADS F.S.1 abs/rdf F.S.1 abs/rdf There is the interview of the state is the state is the interview of the state is the stat
 | F3: AR8 F3: AR8 F3: JakeNad F3: JakeNad F3: JakeNad F3: JakeNad TakeNation F3: JakeNation TakeNation F3
 | F3: AR8 F3: AR8 F3: JakeNad F3: JakeNad F3: JakeNad F3: JakeNad TakeNation F3: JakeNation TakeNation F3 | F3: AR8 F3: AR8 F3: JakeNad F3: JakeNad F3: JakeNad F3: JakeNad TakeNation F3: JakeNation TakeNation F3 | F3: AR8 F3: AR8 F3: JakeNad F3: JakeNad F3: JakeNad F3: JakeNad TakeNation F3: JakeNation TakeNation F3 | |
| F3.5. ANSS F3.5. ANSS F3.5.1 Abs/rdf F3.5.1 Abs/rdf F1.5.1 Abs/rdf F1.5.1 Abs/rdf F2.5.1 Abs/rdf F1.5.1 Abs/rdf F2.5.1 Abs/rdf F1.5.1 Abs/rdf F2.5.1 Abs/rdf F1.5.1 Abs/rdf F3.5.1 Abs/rdf F1.5.1 Abs/rdf F3.5 Abs/rdf F2.5.1 Abs/rdf F3.5 A
 | F35 AB8 F35 AB8 F331 abs/ind F34 abs/ind Hirty Assis Fast Fast Fast Fast Fast Fast Fast Fast
 | F3. ARS F3. ARS F.S. Janksvid F.S. Janksvid F.S. Janksvid F.S. Janksvid Hitser, Jac, Jac, Jac, Jac, Jac, Jac, Jac, Jac
 | F3: AR8 F3: Jakkvid
 | F3 ABS F3 ABS F3.1 abs/vid F3.5 abs/vid Threat views. Threat views. Threat views. <tdt< td=""><td>F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid</td><td>F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid</td><td>F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid</td><td>F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3
abs/vid</td></tdt<> | F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid
 | F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid | F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid | F3_AR8 F3_S AR8 F.S.1 abs/vid F.S.1 abs/vid T.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.3 abs/vid F.S.4 abs/s/vid F.S.3 abs/vid |
 |
| F3.5 ANSS F3.5 ANSS F3.5 ANSS F3.5 ANSS F1.5.1 Ansk ref F1.5.1 Ansk ref F2.5.1 Ansk ref F1.5.1 Ansk ref F2.5.1 Ansk ref F1.5.1 Ansk ref F2.5.1 Ansk ref F2.5.1 Ansk ref F2.5.1 An
 | F35 AR8 F35 AR8 F331 abs/rdd F351 abs/rdd F1531 abs/rdd F351 abs/rdd
 | F35 ARS F35 ARS F35 and Statistic F35 and Statistic F33 and Statistic F34 and Statistic F34 and Statistic F34 and Statistic
 | F35 AR8 F35 AR8 F.5.1 abs/vid F.5.1 abs/vid There's few: set_loci
 | F.3. ARS F.3. ALS F.5.1 Abs/vid F.5.2 Abs/vid F.5.2 Abs/vid F.5.3 Abs/vid F.5.3 Abs/vid F.5.4 Abs/vid F.5.5 Abs/vid F.5.5 Abs/vid F.5.5 Abs/vid F.5.5 Abs/vid F.5.5 Abs/vid F.5.5 Abs/vid F.5.6 Abs/vid F.5.7 Abs/vid F.5.7 Abs/vid F.5.7 Abs
 | F3 ARS F3 ARS F.S.1 abs/vid F.S.1 abs/vid There is the isotron in the is
 | F3 ARS F3 ARS F.S.1 abs/vid F.S.1 abs/vid There is the isotron in the is | F3 ARS F3 ARS F.S.1 abs/vid F.S.1 abs/vid There is the isotron in the is | F3 ARS F3 ARS F.S.1 abs/vid F.S.1 abs/vid There is the isotron in the is | |
| F.3. ARS F.3. ARS F.3.1 Abs/Mit F.3.1 Abs/Mit F.3.1 Abs/Mit F.3.1 Abs/Mit T.4.1 Abs/Mit F.3.1 Abs/Mit T.4.2 Abs/Mit F.3.1 Abs/Mit T.4.2 Abs/Mit F.3.1 Abs/Mit T.4.2 Abs/Mit F.3.1 Abs/Mit T.4.2 Abs/Mit F.3.1 Abs/Mit T.4.3 Abs/Mit F.3.1 Abs/Mit T.4.4 Abs/Mit F.3.1 Abs/Mit T.4.5 Abs/Mit F.3.1 Abs/Mit T.4.6 Abs/Mit F.3.1 Abs/Mit T.4.7 Abs/Mit F.3.1 Abs/Mit T.4.8 Abs/Mit F.3.1 Abs/Mit T.4.8 Abs/Mit F.3.1 Abs/Mit Mit F.3.1 Abs/Mit
 | F35 ABS F35 ABS F35 ABS F35 ABS F35 ABS F55 ABS
 | F35 ARS F35 ARS F35 also Ars F35 also Ars F35 also Ars F35 also Ars Table Ars F35 also Ars
 | F35 AR8 F35 AR8 F35 Jakswid F3 Jakswid <td>F.3. ARS F.3. ARS F.5.1 abs/vid F.3.1 abs/vid F.5.1 abs/vid F.5.1 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.3 abs/vid F.5.1 abs/vid F.5.3 abs/vid</td> <td>F3_5_0_0000 F35_0000 F35_0000</td> <td>F3_5_0_0000 F35_0000 F35_0000</td> <td>F3_5_0_0000 F35_0000 F35_0000</td> <td>F3_5_0_0000 F35_0000 F35_0000</td>
 | F.3. ARS F.3. ARS F.5.1 abs/vid F.3.1 abs/vid F.5.1 abs/vid F.5.1 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.2 abs/vid F.5.3 abs/vid F.5.1 abs/vid F.5.3 abs/vid
 | F3_5_0_0000 F35_0000 | F3_5_0_0000 F35_0000 | F3_5_0_0000 F35_0000
 | F3_5_0_0000 F35_0000 | |
| F.3. ANS F.3. ANS F.3. ANS F.3. Jakova F.3. Jakova F.3. Jakova <t< td=""><td>F35 AR8 F35 Ar8</td><td>F3. ABS F3. ABS F.3.1 abs&rd F.3.1 abs&rd T.5.1 abs&rd Text: return at the retrortered of the ret</td><td>F35 ABS8 F35.1 abs/vid F.5.1 abs/vid F35.1 abs/vid F.5.1 abs/vid F15.1 abs/vid F.5.1 abs/vid F15.2 abs/vid F.5</td><td>F.3. NRS F.3. NRS F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid</td><td>F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2</td><td>F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2</td><td>F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2</td><td>F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2</td></t<> | F35 AR8
 | F3. ABS F3. ABS F.3.1 abs&rd F.3.1 abs&rd T.5.1 abs&rd Text: return at the retrortered of the ret
 | F35 ABS8 F35.1 abs/vid F.5.1 abs/vid F35.1 abs/vid F.5.1 abs/vid F15.1 abs/vid F.5.1 abs/vid F15.2 abs/vid F.5
 | F.3. NRS F.3. NRS F.S.1 abs/vid F.S.1 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.2 abs/vid F.S.3 abs/vid F.S.1 abs/vid F.S.3 abs/vid
 | F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2
 | F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2 | F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2 | F3_5_AR8 F3_5_AR8 F35_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_1 abs/vdi F3_2 abs/vectors F3_1 abs/vdi F3_2 abs/vectors F3_2
 | |
| F.3. ANS F.3. ANS <td< td=""><td>F35 AISS F35 A</td><td>F35 ABS F35 ABS F351 abs.Md F351 abs.Md F31 abs.Md F31 abs.Md F31 abs.Md F31 abs.Md</td><td>F3_5_NNS F3_5_NNS F3_5_1 abs/vd F3_5_1 abs/vd F3_5_1 abs/vd</td><td>F.3. ARS F.3. ARS F.3.1 abs/vd F.3.1 abs/vd T.5.1 abs/vd T.5.1 abs/vd T.5.2 abs/vd T.5.1 abs/vd <t< td=""><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td></t<></td></td<> | F35 AISS F35 A
 | F35 ABS F35 ABS F351 abs.Md F351 abs.Md F31 abs.Md F31 abs.Md F31 abs.Md F31 abs.Md
 | F3_5_NNS F3_5_NNS F3_5_1 abs/vd F3_5_1 abs/vd F3_5_1 abs/vd
 | F.3. ARS F.3. ARS F.3.1 abs/vd F.3.1 abs/vd T.5.1 abs/vd T.5.1 abs/vd T.5.2 abs/vd T.5.1 abs/vd <t< td=""><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td><td>F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd F.S.1 abs/vd</td></t<>
 | F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd | F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd
 | F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd | F3. ARS F3. ARS F.S.1 abs/vd F.S.1 abs/vd | |
| F35 ANS F35 ANS F31 Abs/vid F31 Abs/vid F32 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F32 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F31 Abs/vid F32 Abs/vid F31 Abs/vid F31 Abs/vid <
 | F35 AR8
 | F3: ARS F3: ARS F3: Jackwid F3: Jackwid F3: Jackwid F3: Jackwid F3: Jackwid F3: Jackwid Tarter / set F
 | F3_5_ANS F3_5_ANS F.3.1 abs.rds F.3.1 abs.rds Table of the state of the
 | F.3. ARS F.3. ARS F.3.1 abs/vid F.3.1 abs/vid T.5.1 abs/vid F.5.1 abs/vid F.5.1 abs/vid F.5.1 abs/vid F.5.2 abs/vid
 | F3: ARS F3: ARS F.3.1 abs/rdi F.3.1 abs/rdi Table is even in the intervence of th | F3: ARS F3: ARS F.3.1 abs/rdi F.3.1 abs/rdi Table is even in the intervence of th
 | F3: ARS F3: ARS F.3.1 abs/rdi F.3.1 abs/rdi Table is even in the intervence of th | F3: ARS F3: ARS F.3.1 abs/rdi F.3.1 abs/rdi Table is even in the intervence of th | |
| F.35 ANSS F.31 1000000 F.31 10000000 F.31 100000000 F.32 100000000 F.33 100000000 F.33 100000000 F.33 1000000000 F.33 1000000000 F.33 100000000 F.33 1000000000000 1000000000000000000000000000000000000
 | F35 AR8
 | F35 ARS F35 ARS F351 abs/vid F351 abs/vid Table of the state of the sta
 | F3. ANS F3. ANS F.3.1 abs/rdf F.3.1 abs/rdf Table references of the state structures of the s
 | F3 ABS F3 ABS F3.1 abs/rdi F3.1 abs/rdi F3.51 abs/rdi F3.51 abs/rdi F15.1 abs/rdi F3.51 abs/rdi F15.2 abs/rdi F3.51 abs/rdi F15.3 abs/rdi F3.5 abs/rdi F15.4 abs/rdi F3.5 abs/rdi F15.5 abs/rdi F3.5 abs/rdi F15.5 abs/rdi F3.5 abs/rdi F15.5 abs/rdi F3.5 abs/rdi F15.5 abs/rdi F3.5 abs/rdi F2.5 abs/rdi F3.5 abs/rdi F3.5 abs/rdi F3.5 abs/rdi
 | F35 ABS F35 ABS F35.1 abs.rds F35.1 abs.rds Table abs.rds F35.1 abs.rds F35.1 abs.rds F35.1 abs.rds F35.2 abs.rds F35.2 abs.rds F35.3 abs.rds F35.3 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds
 | F35 ABS F35 ABS F35.1 abs.rds F35.1 abs.rds Table abs.rds F35.1 abs.rds F35.1 abs.rds F35.1 abs.rds F35.2 abs.rds F35.2 abs.rds F35.3 abs.rds F35.3 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds | F35 ABS F35 ABS F35.1 abs.rds F35.1 abs.rds Table abs.rds F35.1 abs.rds F35.1 abs.rds F35.1 abs.rds F35.2 abs.rds F35.2 abs.rds F35.3 abs.rds F35.3 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds | F35 ABS F35 ABS F35.1 abs.rds F35.1 abs.rds Table abs.rds F35.1 abs.rds F35.1 abs.rds F35.1 abs.rds F35.2 abs.rds F35.2 abs.rds F35.3 abs.rds F35.3 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds F35.4 abs.rds F35.3 abs.rds |
 |
| F.3. ANSS F.3. ANS F.3. ANS <t< td=""><td>F35 ASS F35 ASS F35 ASS F35 ASS F35 A ASS <t< td=""><td>F35 ARS F35 ARS F351 abs.rds F351 abs.rds Table abs.rds F352 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds</td><td>F3. ARS F3. ARS F.3.1 abs/vd F.3.1 abs/vd Table of the factor of the</td><td>F3 ABS F3 ABS F3.1 abs/vid F3.1 abs/vid F3.1 abs/vid Table references Table r</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td></t<></td></t<> | F35 ASS F35 ASS F35 ASS F35 ASS F35 A ASS <t< td=""><td>F35 ARS F35 ARS F351 abs.rds F351 abs.rds Table abs.rds F352 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds</td><td>F3. ARS F3. ARS F.3.1 abs/vd F.3.1 abs/vd Table of the factor of the</td><td>F3 ABS F3 ABS F3.1 abs/vid F3.1 abs/vid F3.1 abs/vid Table references Table r</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td><td>F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf</td></t<> | F35 ARS F35 ARS F351 abs.rds F351 abs.rds Table abs.rds F352 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F353 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds Table abs.rds F354 abs.rds
 | F3. ARS F3. ARS F.3.1 abs/vd F.3.1 abs/vd Table of the factor of the
 | F3 ABS F3 ABS F3.1 abs/vid F3.1 abs/vid F3.1 abs/vid Table references Table r
 | F35 ABS F35 ABS F35.1 abs/rdf F35.1
abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf | F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf | F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf Table abs/rdf | F35 ABS F35 ABS F35.1 abs/rdf F35.1 abs/rdf Table abs/rdf Table abs/rdf
Table abs/rdf | |
| F.3. ANSS F.3. ANSS F.3. ANSS F.3. ANSS F.3. ANSS F.3. ANSS F.3. ANS
 | F35 ASS F35 ASS F3.1 abs.vid F3.1 abs.vid F3.2 abs.vid F3.2 abs.vid F3.2 abs.vid F3.2 abs.vid F3.3 abs.vid F3.3 abs.vid
 | F.3. ARS F.3. ARS F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.1.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.2 abs.rd F.3.1 abs.rd F.3.3 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.2 abs.rd F.3.2 abs.rd F.3.3 abs.rd F.3.3 abs.rd F.3.3 abs.rd F.3.4 abs.rd F.3.3 abs.rd F.3.4 abs.rd F.3.3 abs.rd F.3.4 abs.rd F.3.3 abs.rd F.3.4 abs.rd <t< td=""><td>F3. MSS F3. MSS F.3.1 absNuk F.3.1 absNuk F.3.1 absNuk F.3.1 absNuk E.1.1 absNuk F.3.1 absNuk E.2.2 absNuk F.3.1 absNuk E.3.2 absNuk F.3.1 absNuk E.3.3 absNuk F.3.1 absNuk E.3.1 absNuk F.3.1 absNuk E.3.2 absNuk F.3.1 absNuk E.3.3 absNuk F.3.1 absNuk</td><td>F.3. ABS F.3. ABS F.3.1 absNut F.3.1 absNut Thread F.4.1 absNut Thread<td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td><td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td><td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td><td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td></td></t<> | F3. MSS F3. MSS F.3.1 absNuk F.3.1 absNuk F.3.1 absNuk F.3.1 absNuk E.1.1 absNuk F.3.1 absNuk
 E.2.2 absNuk F.3.1 absNuk E.3.2 absNuk F.3.1 absNuk E.3.3 absNuk F.3.1 absNuk E.3.1 absNuk F.3.1 absNuk E.3.2 absNuk F.3.1 absNuk E.3.3 absNuk F.3.1 absNuk
 | F.3. ABS F.3. ABS F.3.1 absNut F.3.1 absNut Thread F.4.1 absNut Thread <td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td> <td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td> <td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td> <td>F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd</td> | F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd
 | F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd | F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd | F3. ABS F3. ABS F.3.1 abs.rbd F.3.1 abs.rbd F.3.1 abs.rbd F3.1 abs.rbd F15.1 abs.rbd F3.2 abs.rbd |
 |
| F.35 ANSS F.35.1 abs/hdi F.35.1 abs/hdi F.35.2 abs/hdi F.35.2 abs/hdi F.35.3 abs/hdi F.35.3 abs/hdi
 | F35 ASS F35.1 Jabs.vid F35.1 Litters vector F35.1 Litters vector F35.1 Litters vector F35.1 Litters vector F35.2 Litters vector F35.3 Litters vector
 | F.3. ABS F.3. ABS F.3.1 abs/vbd F.3.1 abs/vbd F.3.1 abs/vbd F.3.1 abs/vbd F.3.1 abs/vbd F.3.1 abs/vbd F.3.1 abs/vbd F.4.1 bb/rbd F.4.1 bb/rbd F.4.1 bb/rbd <t< td=""><td>F.3. ANS F.3. ANS F.3.1 abs.rdi F.3.1 abs.rdi F.4.1 bit F.3.1 abs.rdi F.3.1 abs.rdi F.3.1 abs.rdi F.4.2 bit F.3.1 abs.rdi F.4.1 bit F.3.1 abs.rdi F.4.2 bit F.4.2 bit F.4.1 bit F.4.2 bit F.4.2 bit F.4.2 bit F.4.2 bit F.4.2 bit F.4.1 bit F.4.2 bit F.4.1 bit F.4.2 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.2 bit F.4.2 bit F.4.1 bit F.4.1 bit F.4.2 bit F.4.2 bit F.4.1 bit F.4.1 bit F.4.2 bit</td><td>F.3. ABS F.3. ABS F.3.1 abs/rdf F.3.1 abs/rdf F.3.1 abs/rdf F.3.1 abs/rdf T.1.1 bitrory is received on the intercence of t</td><td>F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.</td><td>F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.</td><td>F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.</td><td>F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.</td></t<> | F.3. ANS F.3. ANS F.3.1 abs.rdi F.3.1 abs.rdi F.4.1 bit F.3.1 abs.rdi F.3.1 abs.rdi F.3.1 abs.rdi F.4.2 bit F.3.1 abs.rdi F.4.1 bit F.3.1 abs.rdi F.4.2 bit F.4.2 bit F.4.1 bit F.4.2 bit F.4.2 bit F.4.2 bit F.4.2 bit F.4.2 bit F.4.1 bit F.4.2 bit F.4.1 bit F.4.2 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.1 bit F.4.2 bit F.4.2 bit F.4.1 bit
F.4.1 bit F.4.2 bit F.4.2 bit F.4.1 bit F.4.1 bit F.4.2 bit
 | F.3. ABS F.3. ABS F.3.1 abs/rdf F.3.1 abs/rdf F.3.1 abs/rdf F.3.1 abs/rdf T.1.1 bitrory is received on the intercence of t
 | F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35. | F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35. | F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.
 | F35 ABS F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F15.1 abs/rbf F35.1 abs/rbf F35.1 abs/rbf F35. | |
| F.35< AR88
 | F35 ABS F35 ABS F35.1 abs.vid F35.1 abs.vid F35.1 abs.vid F35.1 abs.vid F1847 Versit F1847 Versit F1847 Versit F1847 Versit F1847 Versit F1847 Versit F1847 Versit F1847 Versit F1847 Versit F1844 Versit F1848 Versit F1844 Versit
 | F.3. ABS F.3. ABS F.3.1 absNukd F.3.1 absNukd F.3.1 absNukd F.3.1 absNukd E.5.1 absNukd F.4.1 bit for the filter of t
 | F35 ANS F35 ANS F.351 absNet/di F.351 absNet/di F.351 absNet/di F151 apsNet/distribution F152 absS.futcture1.in F152 absS.futcture2.in F152 </td <td>F.3. ABS F.3. ABS F.3.1 abs/vid F.3.1 abs/vid Therry interviewed Therry interviewed Therry interviewed Therry int</td> <td>F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(</td> <td>F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(</td> <td>F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(</td> <td>F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(</td>
 | F.3. ABS F.3. ABS F.3.1 abs/vid F.3.1 abs/vid Therry interviewed Therry interviewed Therry interviewed Therry int
 | F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(
 | F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(| F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(| F3 ANS F3 ANS F3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd F.3.1 Jbsv/shd There's face set(
 | |
| F.35< ABS8
 | F35 ARS
 | F.3. ARS F.3. ARS F.3.1 abs/vd F.3.1 abs/vd F.3.2 abs/vd F.3.2 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.3 abs/vd F.3.4 abs/vd F.3.3 abs/vd F.4.4 abs/vd F.3.3 abs/vd F.4.4 abs/vd F.3.3 abs/vd F.4.4 abs/vd F.3.4 abs/vd F.4.4 abs/vd <t< td=""><td>F.3. ABS F.3. ABS F.3.1 abs.rdi F.3.1 abs.rdi F.3.1 abs.rdi F.3.1 abs.rdi There's endingent construction of Abss is in the activity for the internation of Abss is in the Abs. activity for the Abs.</td><td>F.3. M8% F.3.1 abs/vid F.3.1 abs/vid Therry vert; Therry vert;</td><td>F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd()</td><td>F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd()</td><td>F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd()</td><td>F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd()</td></t<>
 | F.3. ABS F.3. ABS F.3.1 abs.rdi F.3.1 abs.rdi F.3.1 abs.rdi F.3.1 abs.rdi There's endingent construction of Abss is in the activity for the internation of Abss is in the Abs. activity for the Abs.
 | F.3. M8% F.3.1 abs/vid F.3.1 abs/vid Therry vert;
 | F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd() | F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd() | F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3
abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd() | F.3. ANS F.3. ANS F.3.1 abs.rd() F.3.1 abs.rd() F.5.1 abs.rd() F.3.1 abs.rd() T.5.1 abs.rd() F.3.1 abs.rd() T.5.2 abs.rd() F.3.2 abs.rd() T.5.3 abs.rd() F.3.4 abs.rd() T.5.4 abs.rd() F.3.4 abs.rd() T.5.5 abs.rd() F.3.4 abs.rd() | |
| F.35 ABSS F.35.1 abs/vid INPRAY VEC: INPRAY VEC: INPRAY VEC: INPRAY VEC: INPRAY ABS: INPRAY VEC:
 | F.3. AISS F.3. AISS F.3.1 ab&Md F.3.1 ab&Md F.3.1 ab&Md F.3.1 ab&Md F.3.1 ab&Md F.3.1 ab&Md F.3.1 ab&Md F.3.1 ab Therey acc F.3.1 ab T.5.1 ab F.3.1 b
 | F.3. ARS F.3. ARS F.3.1 abs.vd F.3.1 abs.vd F.3.1 abs.vd F.3.1 abs.vd F.3.1 abs.vd F.3.1 abs.vd There is interview: F.3.2 abs.vd There is interview: F.3.4 abs.sd T
 | F.3. AIS8 F.3. AIS8 F.3.1 abs.rdd F.3.1 abs.rdd F.3.1 abs.rdd F.3.1 abs.rdd There's feection There's feection There's feeting There's feeting There's feeting There's feeting <tr< td=""><td>F.3. ABS F.3. Abs.rd </td><td>F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet:</td><td>F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet:</td><td>F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet:</td><td>F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet:</td></tr<>
 | F.3. ABS F.3. Abs.rd
 | F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet:
 | F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet: | F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet: | F.3. ANS F.3. ANS F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.4.1 [16 AAL; There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.1 abs/Adi There's feet: F.3.2 abs/Adi There's feet: |
 |
| F.35 ABS8 Library dec: F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd us free set_lot_Lift.all; F.31 abs.vhd us free set_lot_Lift.all; F.32 abs.vhd us free set_lot_Lift.all; F.31 abs.vhd us free set_lot_Lift.all; F.32 abs.vhd us free set_lot_Lift.all; F.32 abs.vhd us free set_lot_Lift.all; F.33 abs.vhd us free set_lot_Lift.all; F.34 abs.vhto us free set_lot_Lif
 | F.3. ARS F.3. ARS F.3.1 abs/wd F.3.1 abs/wd F.3.2 abs/md F.3.1 abs/wd F.3.1 abs/wd F.3.1 abs/wd F.3.2 abs/md F.3.2 abs/md F.3.3 abs/md F.3.1 abs/md F.3.1 abs/md F.3.1 abs/md F.3.2 abs/md F.3.2 abs/md F.3.3 abs/md F.3.1 abs/md <t< td=""><td>F.3. ARS F.3. ARS F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.5.1 ab&vid F.4.1 application F.5.1 ab for the state F.4.1 application F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.2 ab for the state F.5.2 ab for the state F.5.2 ab for the state F.5.1 ab for the state F.5.2 ab for the state F.5.1 ab for the state F.5.2 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.2 ab for the state</td><td>F.3. AINS F.3. AINS F.3.1 abs./vd F.3.1 abs./vd F.3.2 abs./vd F.3.2 abs./vd F.3.3 abs./vd F.3.2 abs./vd F.3.4 abs./vd F.3.2 abs./vd F.3.5 abs./vd F.3.2 abs./vd F.3.5 abs./vd F.3.2 abs./vd F.3.5 abs./vd F.3.5 abs./vd F.3.5 abs./vd</td><td>F.3. AUSS F.3. AUSS F.3.1 abs:Ard F.3.1 abs:Ard F.3.1 abs:Ard F.3.1 abs:Ard Interviewe: F.3.2 abs:Ard</td><td>F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) <</td><td>F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) <</td><td>F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) <</td><td>F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) <</td></t<> | F.3. ARS F.3. ARS F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.3.1 ab&vid F.5.1 ab&vid F.4.1 application F.5.1 ab for the state F.4.1 application F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.2 ab for the state F.5.2 ab for the state F.5.2 ab for the state F.5.1 ab for the state F.5.2 ab for the state F.5.1 ab for the state F.5.2 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.3 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.1 ab for the state F.5.2 ab for the state

 | F.3. AINS F.3. AINS F.3.1 abs./vd F.3.1 abs./vd F.3.2 abs./vd F.3.2 abs./vd F.3.3 abs./vd F.3.2 abs./vd F.3.4 abs./vd F.3.2 abs./vd F.3.5 abs./vd F.3.2 abs./vd F.3.5 abs./vd F.3.2 abs./vd F.3.5 abs./vd F.3.5 abs./vd F.3.5 abs./vd
 | F.3. AUSS F.3. AUSS F.3.1 abs:Ard F.3.1 abs:Ard F.3.1 abs:Ard F.3.1 abs:Ard Interviewe: F.3.2 abs:Ard
 | F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) < | F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) <
 | F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) < | F35 ANS F35 ANS F3.5 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd F.3.1 abs.vbd ustrate (acc) ustrate (acc) ustrate (acc) < | |
| F.35 ABS8 ABS8 F.3.1 abs.vhd F.3.1 abs.vhd F.3.2 abs.vhd F.3.2 abs.vhd F.3.3 abs.futureture F.3.3 abs.futureture F.3.3 abs.futureture/vhd F.3.3 abs.futureture F.3.3 abs.futureture/vhd F.3.4 abs.futureture F.3.3 abs.futureture/vhd F.3.4 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.4 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.4 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.1 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.1 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.1 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.3 abs.futureture/vhd F.3.3 abs.futureture/vhd/vhork/vhd/vhork/vhd/s/s/s/s/s/s/s/s/s/s/s/s/s/s/s/s/s/s/s
 | F.3. ANS F.3. ANS F.3.1 abs.Md F.3.1 abs.Md T.5.1 abs.Md There is end of control if ALL The iser is a state
 | F.3. AINS F.3. AINS F.3.1 abs&rdd F.3.1 abs&rdd F.3.1 abs&rdd F.3.1 abs&rdd The freque ericle of Last and the state of the sta
 | F.3. AISS F.3. AISS F.3.1 abst.view F.3.1 abst.view F.3.1 abst.view F.3.1 abst.view There is the interview F.3.1 abst.view T.3.1 abst.view F.3.1 abst.view T.3.1 abst.view F.3.1 abst.view T.3.2 abst.view F.3.1 abst.view T.3.1 abst.view F.3.1 abst.view T.3.2 abst.view F.3.2 abst.view T.3.3 abst.view F.3.3 abst.view T.3.3 abst.view F.3.3 abst.view T.3.3 abst.view F.3.3 abst.view T.3.3 abst.view F.3.3 abst.view T.3.3 abst.view F.3.4 abst.view T.3.4 abst.view F.3.4 abst.view T.3.5 abst.view F.3.4 abst.view T.3.5 abst.view F.3.3 abst.view T.3.5 abst.view F.3.4 abst.view T.3.5 abst.view F.3.4 abst.view T.3.5 abst.view F.3.4 abst.view T.3.5 abst.view F.3.4 abst.view
 | F.3. AUS8 F.3. AUS8 F.3.1 abs/Aud F.3.1 abs/Aud F.3.1 abs/Aud F.3.1 abs/Aud use for the formation of
 | F.3. AINS F.3. AINS F.3.1 abs&rd
 | F.3. AINS F.3. AINS F.3.1 abs&rd | F.3. AINS F.3. AINS F.3.1 abs&rd | F.3. AINS F.3. AINS F.3.1 abs&rd
 | |
| F.35 ABS8 ABS8 F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.32 ANS8 F.31 abs.vhd F.31 abs.vhd F.31 abs.vhd F.32 abs.vhd F.31 abs.vhd F.32 abs.vhd F.32 abs.vhd F.32 abs.vhd F.33 abs.vhd F.33 abs.vhd F.33 abs.vhd F.33 abs.vhd F.33 abs.vhd F.33 abs.vhd F.34 abs.vhd F.35
 | F.3. ABS F.3. ABS F.3.1 abs8.vbd F.3.1 abs8.vbd T.5.1 abs8.vbd Therery accilitation in the receiver and origination in the receiver and origin the receiver and origination origination in the receiver and or
 | F.3. AINS F.3. AINS F.3.1 abs&rdf F.3.1 abs&rdf F.3.1 abs&rdf F.3.1 abs&rdf There is the state of th
 | F.3. ANS F.3. ANS F.3.1 abs.vdd F.3.1 abs.vdd There relation is free relation in the relation the relation the relation in the relation the relation in the r
 | F.3. ANS F.3. ANS F.3.1 Abs.vid F.3.1 Abs.vid The reference fraction The reference fraction
 | F.3. ANS F.3. ANS F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd The reference entropy about the reference of th | F.3. ANS F.3. ANS F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd The reference entropy about the reference of th
 | F.3. ANS F.3. ANS F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd The reference entropy about the reference of th | F.3. ANS F.3. ANS F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd F.3.1 abs.rd The reference entropy about the reference of th | |
| F.35 ABS ABS F.3.1 abs.vid F.3.1 abs.vid F.3.2 abs.vid F.3.2 abs.vid F.3.2 abs.vid F.3.2 abs.vid F.3.3 abs.vid F.3.3 abs.vid F.3.3 abs.vid F.3.4 abs.vid F.3.3 abs.vid F.3.4 abs.vid F.3.3 abs.vid F.3.4 abs.vid F.3.1 abs.vid F.3.4 abs.vid F.3.3 abs.vid F.3.4 abs.vid F.3.1 abs.vid F.3.4 abs.vid F.3.1 abs.vid F.3.4 abs.vid F.3.1 abs.vid F.3.4 abs.vid F.3.1 abs.vid <td< td=""><td>F.3. ANS F.3. ANS F.3.1 abs8.hd F.3.1 abs8.hd F.3.2 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.3 abs8.hd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.4.1 abs8.fd F.3.3 abs8.fd F.3.4 abs8.fd F.3.3 abs8.fd</td><td>F.3. AINS F.3. AINS F.3. AINS F.3. AINS F.3. AINS F.3. AINS F.3. AINS AFAI F.3.</td><td>F.3. ANS F.3. ANS F.3.1 abs.rd </td><td>F.35. AISSF.35. AISSF.3.1 abs&rdiF.3.1 abs&rdiIntervention</td><td>F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi</td><td>F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi</td><td>F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi</td><td>F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi</td></td<> | F.3. ANS F.3. ANS F.3.1 abs8.hd F.3.1 abs8.hd F.3.2 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.2 abs8.hd F.3.3 abs8.hd F.3.3 abs8.hd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.3.3 abs8.fd F.4.1 abs8.fd F.3.3 abs8.fd F.3.4 abs8.fd F.3.3 abs8.fd
 | F.3. AINS F.3. AINS F.3. AINS F.3. AINS F.3. AINS F.3. AINS F.3. AINS AFAI F.3.
 | F.3. ANS F.3. ANS F.3.1 abs.rd

 | F.35. AISSF.35. AISSF.3.1 abs&rdiF.3.1 abs&rdiIntervention
 | F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi | F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi
 | F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi | F.3. ANS F.3. ANS F.3.1 abs.udi F.3.1 abs.udi T.3.1 abs.udi | |
| F.35 AINS ALibrary (see) F.35.1 ab&Md F.35.1 ab&Md F.15.1 ab F.35.1 ab F.15.2 ab F.35.2 ab F.35.2 ab F.35.2 ab F.35.3 ab F.35.2 ab <td>F.3: ANS F.3: ANS F.3.1 abs&Md F.3.1 abs&Md F.3.1 abs&Md F.3.1 abs&Md There rect_orig_1(f, 1), free rect_orig_1(f, 1</td> <td>F.3. AISS F.3. AISS F.3.1 abs&viet </td> <td>F.3. ANS F.3. ANS F.3.1 Abs.viet F.3.1 Abs.viet The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are r</td> <td>F.35 AIS8 F.35 AIS8 F.3.1 abs/vid F.3.1 abs/vid Interverse Interverse Interverse <</td> <td>F.3. ANS F.3. ANS F.3.1 abs.rd </td> | F.3: ANS F.3: ANS F.3.1 abs&Md F.3.1 abs&Md F.3.1 abs&Md F.3.1 abs&Md There rect_orig_1(f, 1), free rect_orig_1(f, 1
 | F.3. AISS F.3. AISS F.3.1 abs&viet

 | F.3. ANS F.3. ANS F.3.1 Abs.viet F.3.1 Abs.viet The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are reference Event precting of abors is The reference are r
 | F.35 AIS8 F.35 AIS8 F.3.1 abs/vid F.3.1 abs/vid Interverse Interverse Interverse <
 | F.3. ANS F.3. ANS F.3.1 abs.rd | F.3. ANS F.3. ANS F.3.1 abs.rd
 | F.3. ANS F.3. ANS F.3.1 abs.rd | F.3. ANS F.3. ANS F.3.1 abs.rd |
 |
| F.35 AIS8 AISA F.35 AIS8 F.35.1 ab8.Md F.
 | F.3: ANS Also Ans F.3.1 abs.vid F.3.1 abs.vid F.3.1 abs.vid Tree iere: rel_orig_1(4, AL); There iere: rel_orig_1(2, 1); Tree iere: rel_orig_1(4, AL); The iere iere: rel_orig_1(2, 1); Tree iere: rel_orig_1(4, AL); The iere iere: rel_orig_1(2, 1); Tree iere: rel_orig_1(4, AL); The iere: rel_orig_1(2, 1); Tree iere: rel_orig_1(2, AL); The iere: rel_orig_1(2, 1); Tree iere: rel_orig_1(2, AL); The iere: rel_orig_1(2, 1); Tree iere: rel_orig_1(2, AL); The iere: rel_orig_1(2, AL);
 | F.3. ANS F.3. ANS F.3.1 abs8.nd F.3.1 abs8.nd F.3.1 abs8.nd F.3.1 abs8.nd There's fee: There's fee: There's fee: Ther's fee:
 | F.3. ANS F.3. ANS F.3.1 abs3.vhd F.3.1 abs3.vhd Instant end for the formation of
 | F.35 ANS F.35 ANS F.3.1 abs/vid F.3.1 abs/vid F.3.1 abs/vid F.3.1 abs/vid There is a fill of (114 A 1);
in the fill of (114 A
 | F.3. ANS F.3. ANS F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add There is end for the intervention of the interventing of the interventing of the intervention | F.3. ANS F.3. ANS F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add There is end for the intervention of the interventing of the interventing of the intervention
 | F.3. ANS F.3. ANS F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add There is end for the intervention of the interventing of the interventing of the intervention | F.3. ANS F.3. ANS F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add F.3.1 abs/add There is end for the intervention of the interventing of the interventing of the intervention | |
| F.35 AIS8 Library icse; F.3.1 ab8.Md F.3.1 ab8.Md F.3.1 ab8.Md F.3.1 ab8.Md Thrary icse; Library icse; Thrary icse; Component mot; Thrary icse; Componen
 | F.3: ANS Also and a second
 | F.3. ARS F.3. ARS F.3.1 abs/uld F.3.1 abs/uld F.3.1 abs/uld F.3.1 abs/uld There's term F.3.2 abs/uld There's term F.3.3 abs/uld There's term F.3.3 abs/uld
 | F.3. AIS8 F.3. AIS8 F.3.1 abs&rhd F.3.1 abs&rhd F.3.1 abs&rhd F.3.1 abs&rhd Hilterary leee: Interary neet, joid(
 | F.35. ANS F.3.1 abs/vid F.3.1 abs/vid
 | F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""><td>F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""><td>F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""><td>F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""></tr<></td></tr<></td></tr<></td></tr<>
 | F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""><td>F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""><td>F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""></tr<></td></tr<></td></tr<> | F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""><td>F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""></tr<></td></tr<> | F.3. ANS F.3. ANS F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd F.3.1 abs&nd There'see F.3.1 abs&nd The set of Losic Lifs(1,11) F.3.1 abs The set of Losic Lifs(1,11) F.3.2 abs The set of F.1.1 F.3.1 abs <tr< td=""></tr<> | |
| F.35 AIS8 F.35 AIS8 F.35 AIS8 F.3.1 abs./hd • * * * * * * * * * * * * * * * * * * *
 | F.35 ANS F.35 ANS F.35.1 abs&vdf - TTPS terms wat:
Library wat:
Li
 | F.3. ANS F.3. ANS F.3.1 abs/vd F.3.1 abs/vd F.3.1 abs/vd F.3.1 abs/vd There'see: Trep: procession: There'see: Trep: procession: There'see: Trep: procession: There'see: There'see: Ther'see: Ther'see: Ther'see: Ther'see: Ther'see: Ther'see: Ther'see
 | F.3. ANS F.3. ANS F.3.1 abs/vid
 | E.3. AIS8 F.3. AIS8 F.3.1 abs/vid F.3.1 abs/vid Ibrary iee: Ibrary iee; Ibrary iee; Ibrary iee;
 | F.35 AIS8 F.35 AIS8 F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Hilterary lee: F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Provestion F.3.2 abs& vedorstription Provestion F.3.2 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestor F.3.4 abs*formention
 | F.35 AIS8 F.35 AIS8 F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Hilterary lee: F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Provestion F.3.2 abs& vedorstription Provestion F.3.2 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestor F.3.4 abs*formention | F.35 AIS8 F.35 AIS8 F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Hilterary lee: F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Provestion F.3.2 abs& vedorstription Provestion F.3.2 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestor F.3.4 abs*formention | F.35 AIS8 F.35 AIS8 F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Hilterary lee: F.3.1 abs&vhd F.3.1 abs&vhd F.3.1 abs&vhd Provestion F.3.2 abs& vedorstription Provestion F.3.2 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.3 abs&vedorstription Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestion F.3.4 abs*formention Provestor F.3.4 abs*formention
 | |
| E.35 AIS8
 | F.35 ANS F.35 ANS F.3.1 abs8.vhd F.3.1 abs8.vhd F.3.1 abs8.vhd F. TYPE DECLATION use isses.ret_logic_1164.all; F. TYPE DECLAVITORS use isses.ret_logic_111111 F. TYPE DECLAVITORS use isses.ret_logic_111111 F. TYPE DECLAVITORS use isses.ret_logic_11111 F. TYPE DECLAVITORS use isses.ret_l
 | F.35 ANS F.35 ANS F.3.1 abs3/rdd - F.3.1 abs3/rdd - maintersection - maintersectin - <td>F.35 ABS F.35 ABS F.3.1 abs/rdd F.3.1 abs/rdd Hibrary tee: Hibrary tee: Hibrary terms of to 11111; Hibrary terms for terms of the set is Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for ter</td> <td>E.3. AIS8 F.3. AIS8 F.3.1 abs/vid F.3.1 abs/vid Ibrary iee: Ibrary iee; Ibrary iee: Ibrary iee; Ibrary iee: Ibrary iee; Ibrary iee: Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary</td> <td>F.35ABS8F.35. Abs8F.3.1 abs8/hdF.3.1 abs8/hdHibrary tee:</td>
 | F.35 ABS F.35 ABS F.3.1 abs/rdd F.3.1 abs/rdd Hibrary tee: Hibrary tee: Hibrary terms of to 11111; Hibrary terms for terms of the set is Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for terms here: Hibrary terms of the set is Hibrary terms for ter
 | E.3. AIS8 F.3. AIS8 F.3.1 abs/vid F.3.1 abs/vid Ibrary iee: Ibrary iee; Ibrary iee: Ibrary iee; Ibrary iee: Ibrary iee; Ibrary iee: Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary iee; Ibrary
 | F.35ABS8F.35. Abs8F.3.1 abs8/hdF.3.1 abs8/hdHibrary tee:
 | F.35ABS8F.35. Abs8F.3.1 abs8/hdF.3.1 abs8/hdHibrary tee: | F.35ABS8F.35. Abs8F.3.1 abs8/hdF.3.1 abs8/hdHibrary tee: | F.35ABS8F.35. Abs8F.3.1 abs8/hdF.3.1 abs8/hdHibrary tee:
 | |
| F.35 AIS8 R.35 AIS8 F.341 abs/vhd F.341 abs/vhd F.352 abs/vhd F.353 abs/vhd F.353 abs/vhd F.353 abs/vhd F.353 abs/vhd F.354 abs/vhd F.353 abs/vhd F.354 abs/vhd F.353 abs/vhd F.354 abs/vhd F.353 abs/vhd F.354 abs/vhd F.353 abs/vhd
 | F.35 ANS F.35 ANS F.35 ANS F.35 ANS F.35.1 abs.Mid F.35.2 abs.Mid F.35.3 abs.Mid F.35.3 abs.Mid F.35.3 abs.Mid F.45.1 abs.Mid <td< td=""><td>F.35 ABS F.35 ABS F.3.1 abs8/rbd - Hiltery ice: <t< td=""><td>F.35 ABS F.35 ABS F.3.1 abs/rid </td><td>F.35 ABS F.35 ABS F.3.1 abs/rdd F.3.1 abs/rdd Ibrary tee: Ibrary tee: Ibrary tee: Ibrary tee:</td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td></t<></td></td<>
 | F.35 ABS F.35 ABS F.3.1 abs8/rbd - Hiltery ice: - Hiltery ice: <t< td=""><td>F.35 ABS F.35 ABS F.3.1 abs/rid </td><td>F.35 ABS F.35 ABS F.3.1 abs/rdd F.3.1 abs/rdd Ibrary tee: Ibrary tee: Ibrary tee: Ibrary tee:</td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td><td>F.35 ABS F.35 ABS F.3.1 abs/rhd </td></t<>
 | F.35 ABS F.35 ABS F.3.1 abs/rid
 | F.35 ABS F.35 ABS F.3.1 abs/rdd F.3.1 abs/rdd Ibrary tee: Ibrary tee:
 | F.35 ABS F.35 ABS F.3.1 abs/rhd
 | F.35 ABS F.35 ABS F.3.1 abs/rhd | F.35 ABS F.35 ABS F.3.1 abs/rhd | F.35 ABS F.35 ABS F.3.1 abs/rhd
 | |
| F.35 ADS8 F.35 ADS8 F.3.1 abs/vhd F.3.1 abs/vhd F.3.1 abs/vhd F.3.1 abs/vhd The provise intervises F.3.2 abs/s/structure/structure/vhd The provise intervises F.3.2 abs/structure/structur
 | F.35 ANS F.35 ANS F.3.1 abs.Md F.3.1 abs.Md F.3.1 abs.Md F.3.1 abs.Md F.3.1 abs.Md F.3.1 abs.Md There refer ref. logic_1164, all; F.3.1 abs.Md The ref refer ref. logic_1164, all; F.3.1 abs.Md The ref refer ref. logic_1164, all; F.3.1 abs.Md The ref refer ref. logic_1164, all; F.3.1 abs.Matrices The ref refer ref. ref. logic_1164, all; F.3.1 abs.Matrices The ref. ref. ref. ref. ref. ref. ref. ref
 | F.35 ABS F.35 ABS F.3.1 abs8/rdf - ::::::::::::::::::::::::::::::::::::
 | F.35 ABS F.35 ABS F.3.1 abs8/rbd F.3.1 abs8/rbd F.3.1 abs8/rbd F.3.1 abs8/rbd unberst is intermed in the intermed intermed in the intermed
 | F.35 ABS F.35 ABS F.3.1 abs/rdd - Intersy tee: - Intersy tee: <td< td=""><td>F.3. ANS F.3. ANS F.3.1 abs3/rdl - ::::::::::::::::::::::::::::::::::::</td><td>F.3. ANS F.3. ANS F.3.1 abs3/rdl - ::::::::::::::::::::::::::::::::::::</td><td>F.3. ANS F.3. ANS F.3.1 abs3/rdl - ::::::::::::::::::::::::::::::::::::</td><td>F.3. ANS F.3. ANS F.3.1 abs3/rdl - ::::::::::::::::::::::::::::::::::::</td></td<>
 | F.3. ANS F.3. ANS F.3.1 abs3/rdl - ::::::::::::::::::::::::::::::::::::
 | F.3. ANS F.3. ANS F.3.1 abs3/rdl - :::::::::::::::::::::::::::::::::::: | F.3. ANS F.3. ANS F.3.1 abs3/rdl - ::::::::::::::::::::::::::::::::::::
 | F.3. ANS F.3. ANS F.3.1 abs3/rdl - :::::::::::::::::::::::::::::::::::: | |
| F.35 ABS Rivery isee: F.3.1 abs/vhd F.3.1 abs/vhd F.3.1 abs/vhd F.3.1 abs/vhd The ray view: This ray view: This ray view: This ray view: The ray view: This ray view: The ray view: This ray view: The ray view: This ray view:
 | F.35 ABS F.35 ABS F.3.1 abs8.wld F.3.1 abs8.wld F.3.1 abs8.wld F. Tyre prectation of abs8 is universities Erectification univ
 | F.35 ABS F.35 ABS F.3.1 absS/vd F.3.1 absS/vd F.3.1 absS/vd F.3.1 absS/vd use free reflocing (114, al)) F.3.2 absS/vd use free reflocing (1111) F.3.2 absS/vd uservere reflocing (11111) F.3.1 absS/re
 | F.35 ABS F.35 ABS F.3.1 abs/hd F.3.1 abs/hd IPProv (coll) F.3.2 abs/hd IPProv (coll) F.3.1 abs/hd
 | F.35 ABS F.35 ABS F.3.1 abs/rdi
 | F.35 ABS F.35 ABS F.3.1 abs3/rdd - F.3.1 abs3/rdd - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 112; -
 | F.35 ABS F.35 ABS F.3.1 abs3/rdd - F.3.1 abs3/rdd - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 112; - | F.35 ABS F.35 ABS F.3.1 abs3/rdd - F.3.1 abs3/rdd - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 112; -
 | F.35 ABS F.35 ABS F.3.1 abs3/rdd - F.3.1 abs3/rdd - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ice: ici_joit_Lit(i.i) - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 171 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 117; - use ici; in threase area; 172 to 112; - | |
| F.35 ABS F.35 ABS F.3.1 abs.vid F.3.1 abs.vid T.3.1 abs.vid Transview: Transview: Transview: Transview: Transview: T.3.1 abs.vid Transview: Transview:
 | F.35 ABS F.35 ABS F.3.1 abs/vid F. Stab below (16, 114, 41); Territerer structures Ether structures Territerer structure Ether structures Territerer structure Ether structure Territerer structure <td>F.35 ABS F.35 ABS F.3.1 abs8.vid - Type pactavary tee: Illbrary tee: - Type pactavary tee: point: - Type pactavary tea: point: - Type pactavary tea: point:</td> <td>F.35 ABS F.35 ABS F.35.1 abs/vid - Type procession of data is the set interest interest</td> <td>F.35 ABS F.35 ABS F.35.1 abs5.hd F.35.1 abs5.hd Hibrary tee: Hibrary tee: Hibrary term Hibrary tee: Hibrary term Hibrary tee: Hibrary term Hibrary term Hibrary term Hibrary term Hibrary term Hibrary term Hibrary term Hibrary term Hibrary term Hibrary term</td> <td>F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE</td> <td>F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE</td> <td>F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE</td> <td>F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE</td> | F.35 ABS F.35 ABS F.3.1 abs8.vid - Type pactavary tee: Illbrary tee: - Type pactavary tee: point: - Type pactavary tea: point: - Type pactavary tea: point:
 | F.35 ABS F.35 ABS F.35.1 abs/vid - Type procession of data is the set interest
 | F.35 ABS F.35 ABS F.35.1 abs5.hd F.35.1 abs5.hd Hibrary tee: Hibrary tee: Hibrary term Hibrary tee: Hibrary term Hibrary tee: Hibrary term Hibrary term
 | F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE
 | F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE | F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE | F.35 ABS F.35 ABS F.3.1 abs/rdd - TYPE Inclassive verticity level Hiberory ieee: - TYPE Inclassive verticity Hiberory ieee: - TYPE Inclassive Hiberory ieee: - TYPE
 | |
| F.35 ABS F.35 ABS F.3.1 abs/vhd F. S.1 abs/vhd F.3.1 abs/vhd F. S.1 abs/vhd Threavy stee: Threavy stee: Threavy stee:
 | F.35 ABS F.35 ABS F.3.1 abs8/vid F. Star base is Therry ise: Trans rest: field of close is Therry ise: Therry ise:
 | F.35 ABS F.35 ABS F.3.1 abs8.vid - Type perchantions Hibrary ice: - Type perchantions Party assists - Contronnet: Print - Contronnet:
 | F.35 ABS F.35 ABS F.35.1 abs/rdd
 | F.35 ABS F.35 ABS F.35.1 abs./rd F.35.1 abs./rd Ibrary iee: Intervy vect; Ibrary iee: Intervy iee: Ibrary iee: Intervy iee: Ibrary iee: Intervy iee: Ibrary iee: Intervo: Ibrary iee: Intervo: </td <td>F.3. NBS Illerary tee:</td> <td>F.3. NBS Illerary tee:</td> <td>F.3. NBS Illerary tee:</td> <td>F.3. NBS Illerary tee:</td>
 | F.3. NBS Illerary tee:
 | F.3. NBS Illerary tee: | F.3. NBS Illerary tee:
 | F.3. NBS Illerary tee: | |
| F.35 ABS F.35 ABS F.35 ABS F.55 ABS <td< td=""><td>F.35 ABS F.35 ABS F.3.1 abs/vid F.3.1 abs/vid Interry view: Interry view: Intery view:</td><td>F.35 ABS F.35 ABS F.35.1 abs8.vdf - TYPE Inclustry tee: 11Prery tee: - TYPE Inclustry 12Present and - CONTONNEL 12Present - CONTONNEL</td><td>F.35 ABS F.35 ABS F.35.1 abs5.vhd F.35.1 abs5.vhd Hibrary ieee: Hibrary ieee: Library ieee: Hibrary ieee: Hibrary iee: Hibrary ieee: Hibrary ieee: Hi</td><td>F.35 ABS F.35 ABS F.35.1 abs./nd F.35.1 abs./nd F.35.1 abs./nd F.35.1 abs./nd Ilbrary iee: Ilbrary iee: Ilbrary iee: Ilbrary iee:</td><td>F.35ABSF.35. ABSF.35. AbsF.31. abs/rd</td><td>F.35ABSF.35. ABSF.35. AbsF.31. abs/rd</td><td>F.35ABSF.35. ABSF.35. AbsF.31. abs/rd</td><td>F.35ABSF.35. ABSF.35. AbsF.31. abs/rd</td></td<> | F.35 ABS F.35 ABS F.3.1 abs/vid F.3.1 abs/vid Interry view: Interry view: Intery view:
 | F.35 ABS F.35 ABS F.35.1 abs8.vdf - TYPE Inclustry tee: 11Prery tee: - TYPE Inclustry 12Present and - CONTONNEL 12Present - CONTONNEL

 | F.35 ABS F.35 ABS F.35.1 abs5.vhd F.35.1 abs5.vhd Hibrary ieee: Hibrary ieee: Library ieee: Hibrary ieee: Hibrary iee: Hibrary ieee: Hibrary ieee: Hi
 | F.35 ABS F.35 ABS F.35.1 abs./nd F.35.1 abs./nd F.35.1 abs./nd F.35.1 abs./nd Ilbrary iee: Ilbrary iee:
 | F.35ABSF.35. ABSF.35. AbsF.31. abs/rd | F.35ABSF.35. ABSF.35. AbsF.31. abs/rd
 | F.35ABSF.35. ABSF.35. AbsF.31. abs/rd | F.35ABSF.35. ABSF.35. AbsF.31. abs/rd |
 |
| F.35 ABS8 F.35 ABS8 F.55 ABS8
 | F.35ABSF.35.1 abs/vdF.35.1 abs/vdF.35.1 abs/vdF.35.1 abs/vdF.35.1 abs/vdF.35.1 abs/vdHitters in integer and logic_lif4.411;Hitters in integer and logic_lif4.11;Hitters in integer and logic_lif4.11;Itter if in integer and logic_lif4.11;Itter if in integer and logic_lif4.11;Itter if in integer and logic_lif4.11;Prist press and logicPrist prist press and logicPrist prist press and logicPrist prist
 | F.35 ABS F.35 ABS F.35.1 abs./nd F.35.1 abs./nd F.35.1 abs./nd F.35.1 abs./nd Hibrary iee: E.15.1 abs./nd Hibrary isource: E.15.2 abs./nd </td <td>F.35 ABSF.35 ABSF.35.1 abs.MdF.35.1 abs.MdF.35.1 abs.MdF.35.1 abs.MdF.35.1 abs.MdF.35.1 abs.MdIbrary tee:Ibrary ter:Ibrary ter:<t< td=""><td>F.35 ABS F.35 ABS F.35 ABS F.31 abs.wd F.31 abs.wd F.31 abs.wd F.31 abs.wd F.31 abs.wd Ilbrary veck: Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tity pass is Component bit vectificture remark of to +127); Compon</td><td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:</td></td></td></td></t<></td> | F.35 ABSF.35 ABSF.35.1 abs.MdF.35.1 abs.MdF.35.1 abs.MdF.35.1
abs.MdF.35.1 abs.MdF.35.1 abs.MdIbrary tee:Ibrary ter:Ibrary ter: <t< td=""><td>F.35 ABS F.35 ABS F.35 ABS F.31 abs.wd F.31 abs.wd F.31 abs.wd F.31 abs.wd F.31 abs.wd Ilbrary veck: Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tity pass is Component bit vectificture remark of to +127); Compon</td><td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:</td></td></td></td></t<> | F.35 ABS F.35 ABS F.35 ABS F.31 abs.wd F.31 abs.wd F.31 abs.wd F.31 abs.wd F.31 abs.wd Ilbrary veck: Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tese std_loat_lifs(Al); Ilbrary veck: use tity pass is Component bit vectificture remark of to +127); Compon
 | F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck: <td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:</td></td></td>
 | F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck: <td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:<td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:</td></td> | F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck: <td>F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck:</td> | F.35 ABSF.35 ABSF.35.1 abs5.htdF.35.1 abs5.htdF.35.1 abs5.htdIllbrary veck:Illbrary veck: | |
| F.35 ABS8 Library use:
I.S.1 abs.Md F. S.13 abs.Md See std.log(c.164.All);
use isee std.log(c.1164.all);
use isee std.log(c.1164.all);
use isee std.log(c.1164.all);
use isee std.log(c.1164.all);
use isee std.log(c.1164.all);
use isee std.log(c.1164.all);
use ises is utures area very of abs is
component inv
component inv
componen | F.35 ABS F.3.1 abs/vid F.3.1 abs/vid F.3.1 abs/vid Second Filter All (4.41); F.3.1 abs/vid F.3.1 abs/vid F.3.1 abs/vid Second Filter All (4.41); Hibrary ise: Hibrary ise: Second Filter All (4.41); Second Filter All (4.41); Price and Josic_1164.all; Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Price and
Josic_1164.all; Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Price and Josic_1164.all; Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Price and Josic_1164.all; Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Price and Josic_1164.all; Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Price and Josic_1164.all; Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Partice Class (4.41); Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Partice Class (4.41); Second Filter All (4.41); Second Filter All (4.41); Second Filter All (4.41); Partice Class (4.41); Second Filter All (4.41); Second
 | F.35 ABS F.35 ABS F.35.1 abs3.vhd F.35.1 abs3.vhd F.35.1 abs3.vhd F.35.1 abs3.vhd Ilbrary iee: Ilbrary iee: Ilbrary iee: Ilbrary iee: <
 | F.35 ABS F.35 ABS F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md Threaty ise: Threaty ise:
 | F.3. ABS F.3. ABS F.3. bis.wid
 | F.35 ABS F.35 ABS F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isty ass; component and use isty ass; component istr use isty ass; component istr use isty ass; component istr use isty ass; component intr component intr component intr pedia compo
 | F.35 ABS F.35 ABS F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isty ass; component and use isty ass; component istr use isty ass; component istr use isty ass; component istr use isty ass; component intr component intr component intr pedia compo | F.35 ABS F.35 ABS F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isty ass; component and use isty ass; component istr use isty ass; component istr use isty ass; component istr use isty ass; component intr component intr component intr pedia compo | F.35 ABS F.35 ABS F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md F.35.1 abs.Md use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isee: std_loat_lif4.All; use isty ass; component and use isty ass; component istr use isty ass; component istr use isty ass; component istr use isty ass; component intr component intr component intr pedia compo
 | |
| F.35 ABS8 Library vects F.351 abs.Wnd F. Stal abs.Wnd F.351 abs.Wnd F. Stal abs.Wnd F.351 abs.Wnd F. Stal abs.Wnd use less std_lopic_list, all; F. Stal abs.Wnd use less std_lopic_list, all; F. Stal abs.Wnd entity Mess is Component and entity Mess is Component bit entity Mess is Component inv process(x) Process(x) process(x) Proces
 | F.35 ABSF.35 ABSF.35.1 abs.MdF.35.1 abs.MdF.35.1 abs.MdF.35.2 abs.ModeF.35.2 abs. Vectors.Component intComponent int <td>F.35ABSF.35. ABSF.35. ABSF.35.1 abs.widF.35.1 abs.widIlbrary tee:Ilbrary tee:use tees std_loptc_1164.alt;use tees std_loptc_111111use tees std_loptc_111111use tees std_loptc_1111111use tees std_loptc_1111111<!--</td--><td>F.35 ABSF.35 ABSF.35 ABSF.35 Abs.vidF.31 abs.vidF.31 abs.vidF.32 abs.vidIlibrary iee:use iee: std_log(c_11st Alt;)use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?porting and abs?use iee: std_log(c_11st Alt;)porting and abs?porting abs?</td><td>F.3. ABSF.3. AbsF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidIntrast tess:Intrast tess:<t< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""></td<></td></td<></td></td<></td></td<></td></t<></td></td> | F.35ABSF.35. ABSF.35. ABSF.35.1 abs.widF.35.1 abs.widIlbrary tee:Ilbrary tee:use tees std_loptc_1164.alt;use tees std_loptc_111111use tees std_loptc_111111use tees std_loptc_1111111use tees std_loptc_1111111 </td <td>F.35 ABSF.35 ABSF.35 ABSF.35 Abs.vidF.31 abs.vidF.31 abs.vidF.32 abs.vidIlibrary iee:use iee: std_log(c_11st Alt;)use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?porting and abs?use iee: std_log(c_11st Alt;)porting and abs?porting abs?</td> <td>F.3. ABSF.3. AbsF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidIntrast tess:Intrast tess:<t< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""></td<></td></td<></td></td<></td></td<></td></t<></td>
 | F.35 ABSF.35 ABSF.35 ABSF.35 Abs.vidF.31 abs.vidF.31 abs.vidF.32 abs.vidIlibrary iee:use iee: std_log(c_11st Alt;)use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?use iee: std_log(c_11st Alt;)porting and abs?porting and abs?use iee: std_log(c_11st Alt;)porting and abs?porting abs?
 | F.3. ABSF.3. AbsF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidF.3.1 abs.vidIntrast tess:Intrast tess: <t< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""></td<></td></td<></td></td<></td></td<></td></t<>
 | F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv <td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""></td<></td></td<></td></td<></td></td<> | F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv <td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""></td<></td></td<></td></td<> | F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv <td< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv<td< td=""></td<></td></td<>
 | F.35 ABSF.35 ABSF.35 ABSF.35 ABSF.35 abs.widF.35 abs.widIlbrary iee:use iee: std_logic_lif4.al;use iee: std_logic_lif4.al;pointcomponent invcomponent inv <td< td=""></td<> | |
| F.35 ABS8 F.35 ABS8 F.35 Ibs&whd F.35.1 abs&whd Interpreting factors factors Interpreting factors Interpreting factors Interpreting factors Interpreting Interpreting Interpreting I | F.35ABSF.35.1 abs/wdF.35.1 abs/wdF.35.1 abs/wdF.35.1 abs/wdF.35.1 abs/wdF.35.1 abs/wduse ice.ic.F.35.1 abs/wdand x83)F.35.2 abs/wdcomponent intoF.35.2 abs/wdcomponent intoF.35.4 abs/wdcomponent intoF.35.4 abs/wdcomponent intoF.35.4 abs/wd< | F.35ABSF.35.1 abs8.wldF.35.1 abs8.wldF.35.1 abs8.wldF.35.1 abs8.wldHibrary tee:Hibrary tee: <tr< td=""><td>F.35ABS8F.35. 1958.vhdF.31. abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdIhrary iee:Ihrary is is recent of class isIhrary iee:port (: in hrees range of to +127);Ihrary iee:<td>F.35ABS8F.35ABS6F.35. abs8.wfd-F.35.1 abs8.wfd-F.35.1 abs8.wfd-Three tree structures of abs8 isThree tree tree abs8Three tree tree tree abs8Three tree tree tree abs8Three tree abs8<t< td=""><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td></t<></td></td></tr<> | F.35ABS8F.35. 1958.vhdF.31. abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdIhrary iee:Ihrary is is recent of class isIhrary iee:port (: in hrees range of to +127);Ihrary iee: <td>F.35ABS8F.35ABS6F.35. abs8.wfd-F.35.1 abs8.wfd-F.35.1 abs8.wfd-Three tree structures of abs8 isThree tree tree abs8Three tree tree tree abs8Three tree tree tree abs8Three tree abs8<t< td=""><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td></t<></td> | F.35ABS8F.35ABS6F.35. abs8.wfd-F.35.1 abs8.wfd-F.35.1 abs8.wfd-Three tree structures of abs8 isThree tree tree abs8Three tree tree tree abs8Three tree tree tree abs8Three tree abs8 <t< td=""><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td><td>F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres:</td></t<> | F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres: | F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres: | F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres: | F.35 ABS
F.35 abs. with
intervery teres:
Intervery teres: | |
| F.35 ABS8 F.35 ABS8 F.35.1 abs./rd ************************************
 | F.35 ABS F.35 ABS F.35.1 abs.whd - Hibrary ise: - Party ise: - Port(1: in INFEGER range -127 to +127); Port(2: in INFEGER range -127 to +127); Port(2: in INFEGER range -127 to +127); Port(2: in INFEGER range -127 to +127); Port(3: in INFEGER range -128 (ORFEGER ranger -128 (ORFEGER
 | F.35ABS8F.35.1 abs.vtdF.35.1 abs.vtdF.35.1 abs.vtdF.11kray iee:
utterary work;
the rest and the rest
 | F.35 ABS8 F.35 ABS8 F.35.1 abs.vhd F.35.1 abs.vhd F.35.1 abs.vhd F.35.1 abs.vhd Ihrary iee: Ihrary vect; Ihrary iee: Ihrary vect; Ihrary iee: Ihrary vect; Ihrary iee: Ihrary vect; Ihrary iee: Ihrary iee; Ihrary iee: Ihrary vect; Ihrary iee: Ihrary iee; Ihrary iee: Ihrary iee; Ihrary iee; Ihrary vect; entity Mas; Ihrary iee; entity Mas; Ihrary iee; entity Mas; Interstructure indic performent inv Interstructure indic perforemat inv Interstructure indic
 | F.35 ABS F.35 ABS F.35 ABS F.35 ABS F.35 Ibbs.vid F.35 Ibbs.vid F.35 Ibbs.vid F.35 Ibbs.vid F.35 Ibbs.vid F.35 Ibbs.vid Iltrary ieee: Iltrary ieee; Iltrary ieee: Iltrary ieee; Iltrary ieee: Iltrary ieee; Iltrary ieee; Iltrary ieee; Iltrary ieee; Interpretation accounter and accounter and accounter and accounter that accounter and accounter accou
 | F.35 ABS8 F.35 ABS8 F.35.1 abs.vhd
 | F.35 ABS8 F.35 ABS8 F.35.1 abs.vhd | F.35 ABS8 F.35 ABS8 F.35.1 abs.vhd | F.35 ABS8 F.35 ABS8 F.35.1 abs.vhd
 | |
| F.35 ABS8 F.35 ABS8 F.351 abs.Md F. Stabs is Ibstary work; Ibstary work; Ibstary lee: Ibstary lee: Ibstary lee: Ibstary lee: <t< td=""><td>F.35ABSF.35.1 abs.MdF.31.1 bbs.MdF.35.1 abs.MdF.35.1 abs.MdHibrary iee:Hibrary iee:HoldingHibrary iee:Hibrary iee:Hibrary</td><td>F.35ABS8F.35. ABS8F.35. ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdIhrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary iee:<t< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35ABS8F.35.1 b68.vhd</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td></t<></td></t<> | F.35ABSF.35.1 abs.MdF.31.1 bbs.MdF.35.1 abs.MdF.35.1 abs.MdHibrary iee:Hibrary iee:HoldingHibrary iee:Hibrary

 | F.35ABS8F.35. ABS8F.35. ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdIhrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary ieee:Ihrary iee:Ihrary iee: <t< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35ABS8F.35.1 b68.vhd</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd </td></t<>
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd
 | F.35ABS8F.35.1 b68.vhd
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd | |
| F.35 ABS8 F.35 ABS8 F.35.1 abs8.wdd see itee std_logic_1164.ALL; Ibrary work; Ibrary work; iste itee std_logic_1164.ALL; see itee std_logic_1164.ALL; Ibrary work; Ibrary work; stde itee std_logic_1164.ALL; stde itee std_logic_1164.ALL; Ibrary work; Ibrary work; stde itee std_logic_1164.ALL; stde itee std_logic_1164.ALL; Ibrary ABS8 stde iter
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd F.35.1 abs8.vhd F.35.1 abs8.vhd F.35.1 abs8.vhd Hibrary iee: Hibrary iee: Hibrary is: Hibrary iee: Hotick Hibrary iee: <t< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vnd </td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.wld </td><td>F.35 ABS8 R.35.1 abs8. Librery leee;
Librery work;
Librery work;
Librery work;
Librery work;
Librery work;
Librery work;
Librery meet extructure structures:
Librery ABS8 is
port(x: in INTEGER range -127 to +127);
Librery ABS8 is
port(x: in INTEGER range 0 to +127);
Librery ABS8 is
port(x:</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""></td<></td></td<></td></td<></td></td<></td></t<> | F.35 ABS8 F.35 ABS8 F.35.1 abs.vnd
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.wld

 | F.35 ABS8 R.35.1 abs8. Librery leee;
Librery work;
Librery work;
Librery work;
Librery work;
Librery work;
Librery work;
Librery meet extructure structures:
Librery ABS8 is
port(x: in INTEGER range -127 to +127);
Librery ABS8 is
port(x: in INTEGER range 0 to +127);
Librery ABS8 is
port(x:
 | F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""></td<></td></td<></td></td<></td></td<> | F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""></td<></td></td<></td></td<> | F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""></td<></td></td<> | F.35 ABS8 F.35 ABS8 F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid use iece std.logic_1164, all; use iece std.logic_1164, all; use iece std.logic_1166, all; <td< td=""></td<> | |
| F.35 ABS8 F.35 ABS8 Ilibrary vort; Ilibrary vort;<
 | F.35 ABS F.35 ABS F.35.1 abs.vnd F.35.1 abs.vnd F.35.1 abs.vnd F.35.1 abs.vnd Ibrary iee: Ibrary iee: Ibrary ise: Ibrary iee: Ibrary ise: Ibrary ise:
 | F.35 ABS8 F.35 ABS8 F.35. Jabs.vhd F.35. Jabs.vhd F.35. Jabs.vhd F.35. Jabs.vhd Ilbrary iee: Ilbrary iee: Ilbrary iee: Ilbrary iee: <tr< td=""><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Thibrary ices: <td< td=""><td>F.35 ABS F.35 ABS F.31 abs.vhd - TYPE DECLATION:
- TYPE DECLA</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--></td></td></td></td></td<></td></tr<> | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Thibrary ices: - Thibrary ices: <td< td=""><td>F.35 ABS F.35 ABS F.31 abs.vhd - TYPE DECLATION:
- TYPE DECLA</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse:
use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--></td></td></td></td></td<> | F.35 ABS F.35 ABS F.31 abs.vhd - TYPE DECLATION:
- TYPE DECLA
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras </td <td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--></td></td></td> | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras </td <td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--></td></td>
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras </td <td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras <!--</td--></td> | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd F.35.1 abs8.vbd Ibrary iese: F.11 brary isse: use iese std_logic_1164.all; F.11 brary isse: use isse: F.12 brary isse: use isse: F.21 brary isse: use isse: F.22 bras use isse: F.23 bras use isse: F.23 bras use isse: F.35.1 abras use: F.35.1 abras </td | |
| F.35 ABS8 F.35 ABS8 Ilibrary wort; Ilibrary wort;<
 | F.35 ABSF.35 ABSF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdIhrary iee:Ihrary iee: <t< td=""><td>F.35 ABS8F.35 ABS8F.35 ABS8F.35 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdIbrary ieee:Ibrary iee:Ibrary iee:Ibrary ieeIbrary iee</td><td>F.35ABS8F.35ABS8F.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.32abs8.hdF.33abs8.hdF.33abs8.hdF.35abs8.hd<!--</td--><td>F.35 ABS8 Ilbrary ieee; F.31 abs0.vhd - TYPE DECLANTIONS F.3.1 abs0.vhd - TYPE DECLANTIONS F.3.1 abs0.vhd - TYPE DECLANTIONS entry Asss is - CONFORMENTIONS entry Asss is - CONFORMENTIAN end Asss - CONFORMENTIAN acritic tecture BENAUE of hold - CONFORMENTIAN acritic tecture BENAUE of constraint - CONFORMENTIAN perform - CONFORMENTIAN acritic tecture BENAUE - CONCURRENT RISTANTIANTIANS acrit processi</td><td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv</td></td></t<> <td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv<td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv<td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv</td></td></td> | F.35 ABS8F.35 ABS8F.35 ABS8F.35 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdF.31 abs8.vhdIbrary ieee:Ibrary iee:Ibrary iee:Ibrary ieeIbrary iee

 | F.35ABS8F.35ABS8F.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.31abs8.hdF.32abs8.hdF.33abs8.hdF.33abs8.hdF.35abs8.hd </td <td>F.35 ABS8 Ilbrary ieee; F.31 abs0.vhd - TYPE DECLANTIONS F.3.1 abs0.vhd - TYPE DECLANTIONS F.3.1 abs0.vhd - TYPE DECLANTIONS entry Asss is - CONFORMENTIONS entry Asss is - CONFORMENTIAN end Asss - CONFORMENTIAN acritic tecture BENAUE of hold - CONFORMENTIAN acritic tecture BENAUE of constraint - CONFORMENTIAN perform - CONFORMENTIAN acritic tecture BENAUE - CONCURRENT RISTANTIANTIANS acrit processi</td> <td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv</td>
 | F.35 ABS8 Ilbrary ieee; F.31 abs0.vhd - TYPE DECLANTIONS F.3.1 abs0.vhd - TYPE DECLANTIONS F.3.1 abs0.vhd - TYPE DECLANTIONS entry Asss is - CONFORMENTIONS entry Asss is - CONFORMENTIAN end Asss - CONFORMENTIAN acritic tecture BENAUE of hold - CONFORMENTIAN acritic tecture BENAUE of constraint - CONFORMENTIAN perform - CONFORMENTIAN acritic tecture BENAUE - CONCURRENT RISTANTIANTIANS acrit processi
 | F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv | F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv <td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv<td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv</td></td> | F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv <td>F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv</td>
 | F.35ABS8F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.isF.35.2 abs8.iscomponent andComponent andcomponent invComponent inv | |
| F.35 ABS8 F.35 ABS8 Ilbrary vers;
 | F.35ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdIhrary iee:Ihrary iee;Ihrary iedIhrary iee;Ihrary iedIhrary iee;Ihrary iedIhrary iee;Ihrary iedIhrary ied;Ihrary iedIhrary ied; <td>F.35 ABS8F.35 ABS8F.35 ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.redF.35.2 abs8.redF.35.3 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.2 abs8.redF.35.2 abs8.redF.35.3 abs8.redF.35.3 abs8.redF.35.3 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.5 abs8.redF.35.5 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.</td> <td>F.35 ABS F.35 ABS F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd Ihrary iee: F.352 abs.vhd F.352 abs.vhd F.352 abs.vhd F.352 abs.vhd F.354 abs.store F.352 abs.vectors.tr F.354 abs.store F.352 abs.vectors.tr F.354 abs.store F.352 abs.vectors.tr F.354 abs.store F.352 abs.store F.354 abs.store F.352 abs.store F.354 abs.store F.352 abs.store F.354 abs.store F.352 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store</td> <td>R.35 ABS8 F.35.1 abs8.vhd F.35.1 abs8.vhd F.35.1 abs8.vhd Threaty icee; Ihrary icee; Threaty ice; Ihrary icee; Threaty ice; Ihrary icee; Threaty ice; Ihrary ice; Threaty ice; Introposed; Threaty ice; Intro</td> <td>F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro-</td> <td>F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro-</td> <td>F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro-</td> <td>F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro-</td>
 | F.35 ABS8F.35 ABS8F.35 ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8.redF.35.2 abs8.redF.35.3 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.2 abs8.redF.35.2 abs8.redF.35.3 abs8.redF.35.3 abs8.redF.35.3 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.5 abs8.redF.35.5 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.35.4 abs8.redF.
 | F.35 ABS F.35 ABS F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd F.351 abs.vhd Ihrary iee: F.352 abs.vhd F.352 abs.vhd F.352 abs.vhd F.352 abs.vhd F.354 abs.store F.352 abs.vectors.tr F.354 abs.store F.352 abs.vectors.tr F.354 abs.store F.352 abs.vectors.tr F.354 abs.store F.352 abs.store F.354 abs.store F.352 abs.store F.354 abs.store F.352 abs.store F.354 abs.store F.352 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store F.354 abs.store
 | R.35 ABS8 F.35.1 abs8.vhd F.35.1 abs8.vhd F.35.1 abs8.vhd Threaty icee; Ihrary icee; Threaty ice; Ihrary icee; Threaty ice; Ihrary icee; Threaty ice; Ihrary ice; Threaty ice; Introposed; Threaty ice; Intro
 | F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro-
 | F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro- | F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro- | F.35ABS8F.35ABS8F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-F.35.1 abs8.vhd-Ihrary ieee:-Ihrary ieee:-F.35.1 abs8.vhd-F.35.2 abs8.reme-Intro-
 | |
| F.35 ABS8 F.35 ABS8 Ilbrary vert;
 | F.35 ABS8F.35 ABS8F.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.2 abs8.htdF.35.3 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.1 abs8.htdF.35.2 abs8.htsF.35.3 abs8.htsF.35.3 abs8.htsF.35.4 abs8.htsF.35.3 abs8.htsF.35.2 abs8.htsF.35.3 abs8.htsF.35.4 abs8.structure1F.35.4 abs8.structure1F.35.4 abs8.structure1F.35.4 abs8.structure1F.35.4 abs8.structure1F.35.4 abs8.structure1F.35.4 abs8.structure1F.35.2 abs8.structure1F.35.2 abs8.structure1F.35.3 abs8.structure1F.35.4 abs8.structure
 | F.35 ABS8 F.35 ABS8 F.351 abs8.vhd
 | F.35 ABS F.35 ABS F.35.1 abs.vhd
 | E.35 ABS E.35 ABS F.35.1 abs8.vhd E.35.1 abs8.vhd F.35.1 abs8.vhd E.11 thrary icee; 11 hrary icee; use ices etcl.logic_1164.AL; 11 hrary icee; use ices etcl.logic_1164.AL; 11 hrary icee; use ice icel_logic_1164.AL; 11 hrary icee; use ice icel_logic_1164.AL; 11 hrary icee; Type psc.AAAATTONS entity Ass8 is conconsent and? component inv component inv postin conconsent nocip postin
 | F.35 ABS F.35 ABS F.351 abs.vid
 | F.35 ABS F.35 ABS F.351 abs.vid | F.35 ABS F.35 ABS F.351 abs.vid
 | F.35ABS8F.35ABS8F.351abs8.vidF.351abs8.vidF.351abs8.vidF.351abs8.vidF.351abs8.vidIlbrary ieee;architecture structured of abs8 isIlbrary ieee;architecture structured of abs8 isIlbrary ieee;architecture structured of abs8 isarchitecture structured of abs8 isporticy is in informationarchitecture structured of abs8 iscomponent informationscomponent informationscomponent informationsarchitecture structure structuredarchitecture structure structure structuredarchitecture structure structure structuredarchitecture structure structure structuredarchitecture structure structurearchitecture structure structuredbread< | |
| F.35 ABS Ilbrary vect,
Ilbrary vect,
Ilb | F.35 ABS F.35 ABS F.35.1 abs/vhd - Ibrary iee: - Ibrary is: - port(z): - Ibrary is: - port(z): - Ibrary is: - Port(z): - Ibrary is:

 | F.35 ABS8 F.35 ABS8 F.35.1 abs.Md F. Statawort; F.35.1 abs.Md F. Statawort; F.35.1 abs.Vd F. Statawort; Ibrary vec; Use ieee std.logic_lif4.all; use ieee std.logic_lif4.all; F. Statawort; port(x: in impress range -117 to +127); Component unt port(x: in impress range -117 to +127); Component unt port(x: in impress range -117 to +127); Component unt component unt Component unt component
 | E.35 ABS F.35 ABS F.35.1 abs.Md - F.35.1 abs.Nd - F.35.1 abs.Nd - F.35.1 abs.Nd - Thrave work - Thrave structures - To thrave of hass is - To thase of hass is -
 | E.3. ABS E.3. ABS F.3.1 abs3.vhd F.3.1 abs3.vhd F.3.1 abs3.vhd F.3.1 abs3.vhd Ilbrary ieee; E.1.114.AlL; use ieee.std_logic_1164.AlL; estitecture structures use ieee.std_logic_1164.AlL; estitecture poil: estitecture poil: estitecture poil: estitecture poil: estitecture poil: estitecture poil: estite poil: estite poil: estite poil: estite
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ibrary work: - Ibrary work: - Ibrary leee; - Ibrary lee; <
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ibrary work: - Ibrary work: - Ibrary leee; - Ibrary lee; < | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ibrary work: - Ibrary work: - Ibrary leee; - Ibrary lee; < | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ibrary work: - Ibrary work: - Ibrary leee; - Ibrary lee; < |
 |
| F.35 ABS Library tees; F.35.1 abs.whd - TYPE PECLANTTONS Library tees; - TYPE DECLANATTONS Library tees; - CONFORMET DECLANATTONS Library tees; - ILL Library tees; - CONFORMET DECLANATIONS Library tees; - ILL Library tees; - ILL Library tees; - ILL </td <td>F.35 ABSF.35 ABSF.35.1 abs/vidF.35.1 abs/vidF.35.1 abs/vidIbrary see:Ibrary see:use see: std_logic_1164.ALGuse see: std_logic_1111111use see: std_logic_1111111use see: std_logic_1111111use see: std_logic_1264.ALGuse see: std_logic_1264.ALG</td> <td>F.35 ABS
F.35 ABS
F.35 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.2 abs.vdf
F.35.4 abs.sf
F.35.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf</td> <td>F.35 ABS Ilbrary vect F.35.1 abs8.whd - F.35.1 abs8.whd - F.35.1 abs8.whd - F.35.1 abs8.whd - Ilbrary vect - ilbrary vect - ilbrary vect - ilbrary vect - ilbrary lee: - use leee std_logic_list.all; - use leee std_logic_list.all; - component inv - 2: out INFEGER range -177 to +127/; - 2: out INFEGER range -171 to +127/; - 2: out INFEGER range -171 to +127/; - 2: out INFEGER range -171 to +127/; - 2: out INFEGER range -127 to +127/; - 2: out INFEGER range -127 to +127/; - 2: out INFEGER range -127 to +127/; - 2: ou</td> <td>F.35 ABS Ilbrary icee; F.351 abs8.whd - F.351 abs8.whd - F.351 abs8.whd - Ilbrary icee; - Poilt - Ilbrary icee; - Poilt - Ilbrary icee; - Ilbrar - Ilbrary icee;</td> <td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary veck; - use isee :std_logic_li64.all; use isee :std_lo</td> <td>F.35 ABS F.35 ABS F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary work; use isee std.logic_li64.AL; use isee std.logic_li64.al; - component inv - z; out INTEGER range -127 - red at to construct and at to construct and at to construct and at to construct and at to consto construct and at to</td> <td>F.35 ABS F.35 ABS F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary work; use isee std.logic_li64.AL; use isee std.logic_li64.al; - component inv - z; out INTEGER range -127 - red at to construct and at to construct and at to construct and at to construct and at to consto construct and at to</td> <td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary veck; - use isee :std_logic_li64.all; use isee :std_lo</td> | F.35 ABSF.35 ABSF.35.1 abs/vidF.35.1 abs/vidF.35.1 abs/vidIbrary see:Ibrary see:use see: std_logic_1164.ALGuse see: std_logic_1111111use see: std_logic_1111111use see: std_logic_1111111use see: std_logic_1264.ALGuse see: std_logic_1264.ALG
 | F.35 ABS
F.35 ABS
F.35 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.1 abs.vdf
F.35.2 abs.vdf
F.35.4 abs.sf
F.35.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
F.45.4 abs.sf
 | F.35 ABS Ilbrary vect F.35.1 abs8.whd - F.35.1 abs8.whd - F.35.1 abs8.whd - F.35.1 abs8.whd - Ilbrary vect - ilbrary vect - ilbrary vect - ilbrary vect - ilbrary lee: - use leee std_logic_list.all; - use leee std_logic_list.all; - component inv - 2: out INFEGER range -177 to +127/; - 2: out INFEGER range -171 to +127/; - 2: out INFEGER range -171 to +127/; - 2: out INFEGER range -171 to +127/; - 2: out INFEGER range -127 to +127/; - 2: out INFEGER range -127 to +127/; - 2: out INFEGER range -127 to +127/; - 2: ou
 | F.35 ABS Ilbrary icee; F.351 abs8.whd - F.351 abs8.whd - F.351 abs8.whd - Ilbrary icee; - Poilt - Ilbrary icee; - Poilt - Ilbrary icee; - Ilbrar - Ilbrary icee;
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary veck; - use isee :std_logic_li64.all; use isee :std_lo
 | F.35 ABS F.35 ABS F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary work; use isee std.logic_li64.AL; use isee std.logic_li64.al; - component inv - z; out INTEGER range -127 - red at to construct and at to construct and at to construct and at to construct and at to consto construct and at to | F.35 ABS F.35 ABS F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary work; use isee std.logic_li64.AL; use isee std.logic_li64.al; - component inv - z; out INTEGER range -127 - red at to construct and at to construct and at to construct and at to construct and at to consto construct and at to | F.35 ABS8
 F.35 ABS8 F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Ilbrary veck; - use isee :std_logic_li64.all; use isee :std_lo | |
| F.35 ABS library ice; lib
 | F.35 ABSF.35 ABSF.35.1 abs.vidF.35.1 abs.vidF.35.1 abs.vidIlbrary veck;use ieee.std_logic_l164.all;use ieee.st
 | F.35 ABSF.35 ABSF.35. Jabs.vidF.35.1 abs.vidF.35.1 abs.vidF.35.2 abs.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vidF.35.4 abs.futcure2.vid
 | F.35 ABS Ilibrary vicee:
 | R.35 ABS 11ibrary isee: 11ibrary isee: 11ibrary wise: 11ibrary wi
 | F.35 ABS Ilibrary vice:
 | F.35 ABS Ilibrary vice: | F.35 ABS Ilibrary vice:
 | F.35 ABS Ilibrary vice: | |
| F.35ABS8F.35ABS8F.35.1abs8.vhdF.35.1abs8.vhdIbrary ieee:interative structure structuresIbrary ieee:interative structuresIbrary ieee:interative structuresIbrary ieee:interative structuresIbrary ieee:interative structuresinterative structuresinterativesinterative structuresinterativesintity in interast range of the interationsinterativesintity in interast range of the interativeinterativeintity in interast range of the interativeinterativeintity interast rangeinterativeintity interast rangeinterativeintity interast rangeinterativeintity interast rangeinterativeintity interast rangeintityintity intity intityintityintity intity intityintityintity intityintityintity intityintityintity intityintityintity intityintityintity intityintityintity intityintityintity intityintityintity inti
 | F.35 ABSF.35 ABSF.35.1 abs.vndF.35.1 abs.vndF.35.1 abs.vndF.35.1 abs.vndF.35.1 abs.vndF.35.1 abs.vndF.35.1 abs.vndIbrary ise:Ibrary ise: <td>F.35 ABSF.35 ABSF.35 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdIhrary ice:Ihrary ice:<t< td=""><td>F.35 ABS Ilibrary vice:
use ices: studiesty vice:
is architecture structures of abes is
- TYPE pECLANTIONS F.3.1 abs.vhd - TYPE pECLANTIONS I.1. TYPE DECLANTIONS - STORM DECLANATIONS use ices: studiesty ice;
use ice; ice; ice;
use ice; ice; ice; ice; ice; ice; ice; ice</td><td>E.3. ABS E.3. ABS F.3.1 abs8.vhd </td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td></t<></td> | F.35 ABSF.35 ABSF.35 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdF.351 abs/wdIhrary ice:Ihrary ice: <t< td=""><td>F.35 ABS Ilibrary vice:
use ices: studiesty vice:
is architecture structures of abes is
- TYPE pECLANTIONS F.3.1 abs.vhd - TYPE pECLANTIONS I.1. TYPE DECLANTIONS - STORM DECLANATIONS use ices: studiesty ice;
use ice; ice; ice;
use ice; ice; ice; ice; ice; ice; ice; ice</td><td>E.3. ABS E.3. ABS F.3.1 abs8.vhd </td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td><td>F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
 TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
 TYPE pECIARATIONS Std. Abs 8 is
 Structures:
 Struc</td></t<> | F.35 ABS Ilibrary vice:
use ices: studiesty vice:
is architecture structures of abes is
- TYPE pECLANTIONS F.3.1 abs.vhd - TYPE pECLANTIONS I.1. TYPE DECLANTIONS - STORM DECLANATIONS use ices: studiesty ice;
use ice; ice; ice;
use ice; ice; ice; ice; ice; ice; ice; ice
 | E.3. ABS E.3. ABS F.3.1 abs8.vhd
 | F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
TYPE pECIARATIONS Std. Abs 8 is
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Struc
 | F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
TYPE pECIARATIONS Std. Abs 8 is
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Struc | F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
TYPE pECIARATIONS Std. Abs 8 is
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Struc
 | F.35 ABS Ilibrary vice:
use ices: std.logic_1164.ALV;
is reinterture structures:
TYPE pECIARATIONS Ilibrary vice:
use ices: std.logic_1164.ALV;
architecture structures:
TYPE pECIARATIONS Std. Abs 8 is
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Structures:
Struc | |
| F.35ABS8Ilibrary vec:
library vec:
library vec:
library tee:
use iee-std_logic_lif4.all,
use isee.std_logic_lif4.all,
use isee.std_logic_lif4.all,
used isea.Stdatustry wse
absta
component bit
component bit
compone | F.35 ABSF.35 ABSF.351 abs.vidF.351 abs.vidF.351 abs.vidF.351 abs.vidF.351 abs.vidF.351 abs.vidF.351 abs.vidF.351 abs.vidF.351 abs.vidF.352 abs.vidF.352 abs8.vidF.352 abs8.vidF.354
abs8.vidF.35
 | E.35 ABS8F.35 ABS8F.351 abs8.wldF.351 abs8.wldF.352 abs8.wldF.352 abs8.vectors.klF.353 abs8.vectors.klF.353 abs8.vectors.klF.354 abs8.stF.354 ab
 | F.35 ABS8 F.35 ABS8 F.35 abs.vdd F.351 abs.vdd F.351 abs.vdd F.351 abs.vdd F.351 abs.vdd Tibrary ice:
 | F.35ABS8F.35ABS8F.35.1 abs8.wld
 | F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: Tibrary isee: <td< td=""><td>F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: <td< td=""><td>F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: <td< td=""><td>F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: <td< td=""></td<></td></td<></td></td<></td></td<>
 | F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: Tibrary isee: <td< td=""><td>F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: <td< td=""><td>F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: <td< td=""></td<></td></td<></td></td<> | F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: Tibrary isee: <td< td=""><td>F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: <td< td=""></td<></td></td<> | F.35 ABS8 F.35 ABS8 F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld F.351 abs8.wld Tibrary isee: Tibrary isee: <td< td=""></td<> |
 |
| F.35 ABSIlibrary vec:
library vec:
interary vec: <br< td=""><td>F.35ABSF.35.1 abS/MdF.35.1 abS/MdF.35.1 abS.MdF.35.1 abS.MdF.35.1 abS.MdIbrary iee:Ibrary iee:<t< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.2 abs.whdF.35.2 abs.whF.35.2 abs.wh</td><td>F.35ABS8F.35ABS8F.35.1abs.widF.35.1abs.widF.35.1abs.widIlbrary ice:use ieee stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;use ieee stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;use ieee stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;port(z: ain INTEGRE range 0 to +127);component bufareactineecure stel.logic_lid4.AU;component bufport(z: ain INTEGRE range 0 to +127);component bufareactineecure stel.logic_lid4.AU;component bufbeginareactineecure at an areactineecure at a areactineecure</td><td>F.35 ABSAbssF.35 ABSF.35 ABSF.35.1 abs8.whdIllbrary isee:Illbrary ieee:Illbrary isee:Illbrary ieee:Illbrary wer:use isee std_logic_1164.ALiIllbrary isee:use isee std_logic_1164.ALiIllbrary isee:port(y ABS8 isIllbrary isee:port(y ABS8 isIllbrary isee:port(s: in INTEGRE range 0 to +127);Illbrary isee:is in INTEGRE range 0 to +127);Illbrary isee:is intricerure BENUE of ABS8 isIllbrary isee:port(s: in INTEGRE range 0 to +127);Illbrary isee:is intricerure BENUE of ABS8 isIllbrary isee:is intricerure BENUEIllbrary isee:is interverIllbrary isee:is interver</td></t<><td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td><td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td><td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td><td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td></td></br<> | F.35ABSF.35.1 abS/MdF.35.1 abS/MdF.35.1 abS.MdF.35.1 abS.MdF.35.1 abS.MdIbrary iee:Ibrary iee: <t< td=""><td>F.35 ABSF.35 ABSF.35 ABSF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.2 abs.whdF.35.2 abs.whF.35.2 abs.wh</td><td>F.35ABS8F.35ABS8F.35.1abs.widF.35.1abs.widF.35.1abs.widIlbrary ice:use ieee stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;use ieee stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;use ieee stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;port(z: ain INTEGRE range 0 to +127);component bufareactineecure stel.logic_lid4.AU;component bufport(z: ain INTEGRE range 0 to +127);component bufareactineecure stel.logic_lid4.AU;component bufbeginareactineecure at an areactineecure at a areactineecure</td><td>F.35 ABSAbssF.35 ABSF.35 ABSF.35.1 abs8.whdIllbrary isee:Illbrary ieee:Illbrary isee:Illbrary ieee:Illbrary wer:use isee std_logic_1164.ALiIllbrary isee:use isee std_logic_1164.ALiIllbrary isee:port(y ABS8 isIllbrary isee:port(y ABS8 isIllbrary isee:port(s: in INTEGRE range 0 to +127);Illbrary isee:is in INTEGRE range 0 to +127);Illbrary isee:is intricerure BENUE of ABS8 isIllbrary isee:port(s: in INTEGRE range 0 to +127);Illbrary isee:is intricerure BENUE of ABS8 isIllbrary isee:is intricerure BENUEIllbrary isee:is interverIllbrary isee:is interver</td></t<> <td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td> <td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td> <td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td> <td>F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat</td>
 | F.35 ABSF.35 ABSF.35 ABSF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.1 abs.whdF.35.2 abs.whdF.35.2 abs.whF.35.2 abs.wh
 | F.35ABS8F.35ABS8F.35.1abs.widF.35.1abs.widF.35.1abs.widIlbrary ice:use ieee stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;Ilbrary ice:areactineecure stel.logic_lid4.AU;use ieee stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;use ieee stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;areactineecure stel.logic_lid4.AU;port(z: ain INTEGRE range 0 to +127);component bufareactineecure stel.logic_lid4.AU;component bufport(z: ain INTEGRE range 0 to +127);component bufareactineecure stel.logic_lid4.AU;component bufbeginareactineecure at an areactineecure at a areactineecure

 | F.35 ABSAbssF.35 ABSF.35 ABSF.35.1 abs8.whdIllbrary isee:Illbrary ieee:Illbrary isee:Illbrary ieee:Illbrary wer:use isee std_logic_1164.ALiIllbrary isee:use isee std_logic_1164.ALiIllbrary isee:port(y ABS8 isIllbrary isee:port(y ABS8 isIllbrary isee:port(s: in INTEGRE range 0 to +127);Illbrary isee:is in INTEGRE range 0 to +127);Illbrary isee:is intricerure BENUE of ABS8 isIllbrary isee:port(s: in INTEGRE range 0 to +127);Illbrary isee:is intricerure BENUE of ABS8 isIllbrary isee:is intricerure BENUEIllbrary isee:is interverIllbrary isee:is interver
 | F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat | F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat
 | F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat | F.35 ABSR.35 ABSF.35. abs3.wldF.35. abs3.wldF.35. abs3.wldThray iee:Thray areaThray areaThray areaThreatThreatThreatThray areaThreat | |
| F.35 ABSIllbrary isee:F.35.1 abs8.whd-F.35.1 abs8.whd-F.35.1 abs8.whd-Ilbrary iee:-Ilbrary iee:-use ieee std_logic_lif4.all;use ieee std_logic_lif4
 | F.35 ABSF.35 ABSF.35.1 abs.wdF.35.1 abs.wdF.35.2 abs.wdF.35.2 abs.wetors.xtF.35.2 abs.stF.35.4 abs.stF.3
 | F.35 ABSF.35 ABSF.35 ABSF.351 abs8.whdF.351 abs8.whdF.351 abs8.whdIlbrary ieee;Ilbrary ieee;Ibbrary iee;Ilbrary iee;Ilbrary iee;Ibbrary ie
 | F.35 ABS Ilibrary ieee:
Ilibrary ieee:
Ilibrary ieee:
transference studiodic_1164.AU:
use ieee.std_looic_1164.AU:
use ieee.std_looic_127.
use ieee.std_looic_127.
use ieee.std_looic_127.
use ieee.std_looic_127
 | F.35 ABS F.35 ABS F.35.1 abs8.whd Ilibrary ieee: F.35.1 abs8.whd Ilibrary ieee: using the set of constructions Ilibrary ieee: not to the set of constructions Ilibrary is component invo not to the set of constructions Ilibrary is component invo not down be set of to the set of constructions Ilibrary is component invo not down be set of to the set of to
 | F.35ABS8F.35.1 abs8.whdlibrary ieee;F.35.1 abs8.whd- rore: vort:Intrary ieee;library ieee;Intrary ieee;rore: current vort:Intrary ieee;- rore: prostructionsIntrary iee;- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andpor
 | F.35ABS8F.35.1 abs8.whdlibrary ieee;F.35.1 abs8.whd- rore: vort:Intrary ieee;library ieee;Intrary ieee;rore: current vort:Intrary ieee;- rore: prostructionsIntrary iee;- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andpor | F.35ABS8F.35.1 abs8.whdlibrary ieee;F.35.1 abs8.whd- rore: vort:Intrary ieee;library ieee;Intrary ieee;rore: current vort:Intrary ieee;- rore: prostructionsIntrary iee;- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andpor | F.35ABS8F.35.1 abs8.whdlibrary ieee;F.35.1 abs8.whd- rore: vort:Intrary ieee;library ieee;Intrary ieee;rore: current vort:Intrary ieee;- rore: prostructionsIntrary iee;- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range - 127 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andport (x: in INTEGER range 0 to +127);- component andpor
 | |
| F.35 ABS8 F.35 ABS8 Ilbrery veck:
Ilbrery veck:
Ilbrery veck:
Ilbrery veck:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery veck:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery ieee:
Ilbrery veck:
Ilbrery ieee:
Ilbrery ieee:
Ilbre | F.35ABS8F.35.1 abs8.vid- Tyre break in this are interventioned of abs8 isF.35.1 abs8.vid- Tyre break around the set all optic_1164.ALF;use iese set_logic_1164.all;use iese iese iese i

 | F.35 ABS8 F.35 ABS8 F.351 abs8.whd Ilibrary ieee; F.351 abs8.whd - TYPE ieee: eff. logic_1164.ALC; Ilibrary ieee; use ieee.std.logic_1164.ALC; Ilibrary ieee; - TYPE RELANATIONS entity ABS is - CONFORMENT DECLANATIONS port (2: in INTEGER range -127 to +127); - CONFORMENT INT component inv - CONFORMENT INT end ABS; out INTEGER range -127 to +127); end ABS; out INTEGER range -127 to +127); end ABS; - Component inv component inv - Component inv component inv - Component inv process(x) - Construct inv process(x) - Construct inv end BRAUS: - Construct inv end BRAUS: - CONSTRUMENT SCIENT ASSTONMENTINGS end BRAUS: - CONSTRUMENT SCIENT ASSTONMENT end BRAUS: - CONSTRUMENT SCIENT ASSTONMENT end BRAUS: - CONSTRUMENT SCIENT ASSTONMENT
 | F.35 ABS Ilibrary ieee; F.35.1 abs8.whd Ilibrary ieee; F.35.1 abs8.whd Ilibrary ieee; Instray ieee; interversed logic_lif4.AL; use ieee.std_logic_lif4.all; Interversed is architecture structured of abs8 is use ieee.std_logic_lif4.all; Interversed of abs8 is use ieee.std_logic_lif4.all; Interversed of abs8 is use ieee.std_logic_lif4.all; Interversed of abs8 is ort (2: in INTEGER range -127 to +127); Interversed of abs8 is ort (2: in INTEGER range 0 to +127); Interversed of abs8 is ort (2: in INTEGER range 0 to +127); Interversed of abs8 is ort (2: in INTEGER range 0 to +127); Interversed of abs8 is ort (3: in INTEGER range 0 to +127); Interver abs8 end ABS3; untimeGR range 0 to +127); end ABS4; Interver abs8 protein: Interver abs8 protein: Interver abs8 rend ABS4; Interver abs8 architecture BEAVE Interver abs8 protein: Interver abs8 protein: Interver abs8 protein: Interver abs8 architecture BEAVE Interver abs8
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.whd Tibrary ieee; Ilibrary work; Ilibrary work; user leve.std_logic_l164.all; TYPE protecuted of dass is Ilibrary leve; TYPE protecutions Ilibrary leve; Topic_l164.all; Ilibrary leve; Topic_l164.all; </td <td>F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com</td> <td>F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com</td> <td>F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com</td> <td>F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee:
 Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com</td> | F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com
 | F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com | F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com | F.35 ABS Ilibrary ieee: F.351 abs8.whd Ilibrary ieee: Istrary ieee: use ieee.std logic_lidALL; Istrary ieee: use istrary ieee: Istrary ieee: use istrary ieee: Istrary ieee: use istrange of to +127); port (x: in INFEGR range -127 to +127); use intrace entity ABS is use intrace port (x: in INFEGR range 0 to +127); use intrace end ABS; use intrace port (x: in INFEGR range 0 to +127); use intrace begin use intrace begin use intrace component inv use intrace component inv use intrace component inv use intrace begin use intrace architecture BEAVE; component inv architecture BEAVE; component inv architecture BEAVE; com
 | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8F.35 ABS8F.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.351 abs.vhdF.352 abs.vectors.thF.352 abs.vectors.thF.351 abs.vector
 | F.35 ABS F.35 ABS F.35.1 abs.Nid - Type becarrentered of abss is F.35.1 abs.Nid - Type becanarrows use isee std_logic_lif4.nLi, - Type becanarrows or type becanarrows - component buf port(1; in TYPEGR range 0 to +127); - component buf port(2; in TYPEGR range 0 to +127); - component buf port(2; in TYPEGR range 0 to +127); - component buf port(2; in TYPEGR range 0 to +127); - component to rado port(2; in TYPEGR range 0 to +127); - component to rado port(2; in TYPEGR range 0 to +127); - component to rado port(2; in TYPEGR range 0 to +127); - component to rado port(2; in TYPEGR range 0 to +127); - component to rado port(2; in TYPEGR range 0 to +127); - component to rado port(2; in TYPEGR range 0 to +127); - component to rado port(3; in TYPEGR range 0 to +127); - component to rado port(3; in TYPEGR range 0 to +127); - compone
 | F.35 ABS8F.35 ABS8F.35.1 abs8.whdF.35.1 abs8.whdF.35.1 abs8.whdThreary work:
use structured of abs1 is
threary stee:
i bot(2): in INTEGER range 0 to +127);
end ABS1Threary is component inv
component inv
structured of abs1 is
component inv
component inv
component inv
structured inState<
 | F.35 ABS8 F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd F.35.1 abs8.vhd - ithrary work;
uthrary work;
uthrary work;
uthrary work;
uthrary work;
uthrary work;
uthrary work;
uthrary work;
uthrary work;
uthrack 11 hurary work;
uthrary work;
uthrary work;
uthrack 15 hurary work;
uthrack
 | F.3. ABS8 Ilibrary ieee; F.3.1 abs8.wdd - TYPE breakmarrow work; F.3.5.1 abs8.wdd - TYPE breakmarrows Intrary ieee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrave iee; - CORPORED intrave Introver - CORPORED introve Port(1; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port
 | F.3. ABS8 Ilibrary ieee; F.3.1 abs8.wdd - TYPE breakmarrow work; F.3.5.1 abs8.wdd - TYPE breakmarrows Intrary ieee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrave iee; - CORPORED intrave Introver - CORPORED introve Port(1; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port | F.3. ABS8 Ilibrary ieee; F.3.1 abs8.wdd - TYPE breakmarrow work; F.3.5.1 abs8.wdd - TYPE breakmarrows Intrary ieee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrave iee; - CORPORED intrave Introver - CORPORED introve Port(1; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port
 | F.3. ABS8 Ilibrary ieee; F.3.1 abs8.wdd - TYPE breakmarrow work; F.3.5.1 abs8.wdd - TYPE breakmarrows Intrary ieee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrary iee; - CORPORED intrave Intrave iee; - CORPORED intrave Introver - CORPORED introve Port(1; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port(2; in INTEGER range 0 to +127); - CORPORED introve Port | |
| F.35. ABS8 F.35. ABS8 F.35.1 abs8.whd - Ihbrary ieee: Ihbrary over: Ihbrary ieee: - Ibrary is -
 | F.35 ABS8F.35 ABS8F.35.1 abs8.vhdF.35.1 abs8.vhdF.35.2 abs8 vectors:F.35.2 abs8 vectors: <t< td=""><td>F.35 ABS F.35 ABS F.35. Jabs.whd - F.35.1 abs.whd - F.35.2 abs.whd - F.35.2 abs8.whd - F.35.2 abs8.who - F.35.2 abs8.who -</td><td>F.35 ABS F.35 ABS Ilibrary ieee; Ilibrary iee; I</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.whd Ilibrary ieee; F.35.1 abs8.whd Ilibrary ieee; Ilibrary ieee; use ieee.stdlogic_lif4.all; Introduction component introductions entity ABS is component introductions port(; in introcsn range 0 to +127); component introductions int (component introductions) component introductions end ABS9; use introductions int (conses(x)) component introductions</td><td>F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; -</td><td>F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; -</td><td>F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; -</td><td>F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; -</td></t<>
 | F.35 ABS F.35 ABS F.35. Jabs.whd - F.35.1 abs.whd - F.35.2 abs.whd - F.35.2 abs8.whd - F.35.2 abs8.who - F.35.2 abs8.who -
 | F.35 ABS F.35 ABS Ilibrary ieee; Ilibrary iee; I
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.whd Ilibrary ieee; F.35.1 abs8.whd Ilibrary ieee; Ilibrary ieee; use ieee.stdlogic_lif4.all; Introduction component introductions entity ABS is component introductions port(; in introcsn range 0 to +127); component introductions int (component introductions) component introductions end ABS9; use introductions int (conses(x)) component introductions
 | F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; -
 | F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; - | F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; - | F.35 ABS F.35 ABS Ilibrary ieee; F.35.1 abs.wid - Ilibrary ieee; F.35.1 abs.wid - - represent extrantions Ilibrary ieee; - - represent extrantions Ilibrary ieee; - - - - Ilibrary ieee; - - - - - - Ilibrary ieee; -
 | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Interary veck;
Ibrary veck;
Ibrary ieee;
Ibrary ieee;
Ibrary ieee;
Ibrary ieee;
Ibrary ieee;
Ibrary veck;
Ibrary veck;
Ibra | F.35 ABS8
F.35 Jabs.Wid - Interary tee;
f.35.1 abs.Wid - Tritter structured of abs8 is
- Trytte DeclaARATIONS
use ieee std_logic_1164.ALF;
use ieee std_logic_1164.ALF;
use ieee std_logic_1164.ALF;
use ieee std_logic_1164.ALF;
is in INTEGER range -127 to 4127;
entity ABS8 is
porting for the interaction of abs8 is
- COMPONENT DECLARATIONS
- COMPONENT DECLARATIONS
- COMPONENT INTE
Declarize the BERAVE of ABS8 is
- INLINE CONFICTURE
Declarize the BERAVE of ABS8 is
- CONFORTER TAGE
- COMPONENT INTE
- COMPONENT I

 | F.35 ABS F.35 ABS F.35 ABS F.35.1 abs.vid F.35.1 abs.vid F.35.1 abs.vid Thrary ieee; Thrange; <td< td=""><td>F.35 ABS F.35. ABS6 Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; use ieee std.logic_lidtAlL; use iscl.usertere etructurestorestorestorestorestorestorestoresto</td><td>F.35 ABS8 F.35 ABS8 F.35.1 abs8.whd Ilibrary leee; F.35.1 abs8.whd Ilibrary leee; ulbrary leee; Ilibrary leee; port(1; il in inregen range -127); Incomponent intor couponent linv Incomponent inv end Ass3; Incomponent inv component linv Inv porters; Inverse; read broces; Inverse; component linv Inverse; co</td><td>F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth</td><td>F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth</td><td>F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth</td><td>F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth</td></td<> | F.35 ABS F.35. ABS6 Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; use ieee std.logic_lidtAlL; use iscl.usertere etructurestorestorestorestorestorestorestoresto
 | F.35 ABS8 F.35 ABS8 F.35.1 abs8.whd Ilibrary leee; F.35.1 abs8.whd Ilibrary leee; ulbrary leee; Ilibrary leee; port(1; il in inregen range -127); Incomponent intor couponent linv Incomponent inv end Ass3; Incomponent inv component linv Inv porters; Inverse; read broces; Inverse; component linv Inverse; co
 | F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth
 | F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth | F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth | F.35 ABS F.35. ABS F.35. abs. wid: Ilibrary ieee; Ilibrary isouthary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary ieee; Ilibrary isouthary isouth |
 |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intervery work;
Intervery tees:
Intervery tees:
Intervery tees:
Intervery tees:
Intervery tees:
Intervery tees:
Intervery tees:
Intervery tees:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Interverses:
Inter
 | F.35 ABS
F.35 ABS
F.35 abs.vhd - Interary view:
F.35.1 abs.vhd - F.35.1 architecture structured of abs8 is - F.35.1 architecture structured of abs8 is - F.35.1 architecture and - F.35.1 architecture
 | F.35 ABS R.35 ABS F.35.1 abs.whd F.35.1 abs.whd F.35.1 abs.whd F.35.1 abs.whd F.35.1 abs.whd F.35.1 abs.whd Intrary ieee; Intrary ieee; Intrary ieee std_logic_1164.AU; Intrary ieee; use ieee.std_logic_1164.AU; F.35.1 abs.whd use ieee.std_logic_1164.AU; F.35.1 abs.whrd nentity Abs8 is F.35.1 abs.whrd port(1; in INTEGER range -127 to +127); Component into' end Abs8; component inv component inv component inv component inv component inv broins f.2 abs.x; f.abs8 f.abs.x; end BPAUE; F.127; end Enverse; F.2 abs.x; end Enverse; F.2 abs.x; end Enverse; F.2 abs.x; end Enverse; F.2 concurrent inv end Enverse; F.2 abs.x; <
 | F.35 ABS
F.35 ABS
F.35. abs8.whd
F.35.1 abs8.wht
F.35.1 abs8.w
 | F.35 ABS
F.35 ABS
F.35 abs.vhd - Ibbary vert;
F.35.1 abs.vhd - Tribrary vert;
F.35.1 abs.vhd - Tribrary vert;
Ibbary vert;
 | F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work; </td <td>F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work;<!--</td--><td>F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work;<!--</td--><td>F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work;<!--</td--></td></td></td> | F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work; </td <td>F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work;<!--</td--><td>F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work;<!--</td--></td></td> | F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work; </td <td>F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work;<!--</td--></td> | F.35 ABS F.35 ABS F.35.1 abs8.whd F.1brary ieee; Stiture: work; Stiture: work; </td | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd TYPE PECLANATIONS
Intrary work;
use ieee.std_logic_lif4.AL;
use ieee.std_logic_lif4.AL;
use ieee.std_logic_lif4.AL;
use ieee.std_logic_lif4.al;
entity ABS8 is
port(r: in INTEGER range 0 to +127);
end ABS9; out INTEGER range 0 to +127);
end ABS9;
end ABS9;
end ABS9;
end ABS9;
end ABS9; | F.35 ABS
F.35 ABS
F.35 abs/whd - Interary verse;
F.351 abs/whd - F.351 abs/whole a for the format abs/minimum - F.351 abs/mini

 | F.35 ABS F.35 ABS F.35 ABS State of the set logic life ALU;
use isee stat logic life ALU;
use isee stat logic life ALU;
iser isee stat logic life ALU;
iser isee stat logic life ALU;
iser isee stat logic life ALU;
is component but
component but
component but
component inv
component inv
comp
 | F.35 ABS
F.35 ABS
F.35. Jabs.vhd - Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
use ieee std_logic_1164.ALL;
architecture structure1 of abs8 is
- TSTAML DECLANATIONS
- STGAML DECLANATIONS
- STGAML DECLANATIONS
- STGAML DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT INTEGRE range -127 to +127);
end ABS8 is
process(X)
process(X)
end process(X)
end process(X)
end process(X)
end process(X)
end process(X)
end process(X)
end process(X)
end structure1.
 | F.35 ABS8 F.35 ABS8 F.35. Jabs.vhd - ithizary work;
library work;
ulbrary w
 | F.35 ABS
F.35 ABS
F.35.1 absS.vhd
F.35.1 absS. | F.35 ABS
F.35 ABS
F.35.1 absS.vhd
F.35.1 absS. | F.35 ABS
F.35 ABS
F.35.1 absS.vhd
F.35.1 absS. | F.35 ABS
F.35 ABS
F.35.1 absS.vhd
F.35.1 absS. |
 |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS
F.35 ABS
F.35 abs/whd - Interverver estructures structures structures in a structures in a structure structures of abs8 is
Interver structures structures in a structures in a structures in a structures in a structure structures in a structure structure structures in a structure structure structures in a structure structure structure structures in a structure structure structure structure structures in a structure structure structure structure structures in a structure structure structure structure structure structures in a structure structure structure structure structure structures in a structure structure structure structure structures in a structure structures in a structure structures in a structure structure structure structure structure structure structures in a structure structure structure structure structure structure structure structure structures in a structure structure structure structure structure structure structures structurestructures st
 | F.35 ABS F.35 ABS F.35. ABS8 F.35.1 abs8.whd F.35.1 abs8.whd - TYPE DECLANATIONS Library view: - TYPE DECLANATIONS AND DECK - TYPE DECLANATIONS AND DECK - TYPE DECLANATIONS To out INTEGER range -127 to +127); - COMPONENT Info To out INTEGER range -127 to +127); - COMPONENT INTO To out INTEGER range -127 to +127); - COMPONENT INTO To out INTEGER range -127 to +127); - COMPONENT INTO To out INTEGER range -127 to +127); - COMPONENT INTO To out INTEGER range -127 to +127); - COMPONENT INTO To out INTEGER range -127 to +127); - COMPONENT INTO To o
 | F.35 ABS F.35 ABS Ilibrary ieee; Ilibrary vork; Ilibrary ieee; Ilibrary ieee; <td>F.35 ABS8 Ilibrary ieee; F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Thracy ieee; - Ibrary ieee; - entry Abs8 is - port(y in INTEGER range -127 to +127); - end Abs8; - T < abs7;</td> - T < abs8;
 | F.35 ABS8 Ilibrary ieee; F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - F.35.1 abs8.vhd - Thracy ieee; - Ibrary ieee; - entry Abs8 is - port(y in INTEGER range -127 to +127); - end Abs8; - T < abs7;
 | F.35 ABS Ilibrary vect;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary see;
Ilbrary
 | F.35 ABS Ilibrary vect;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary see;
Ilbrary | F.35 ABS Ilibrary vect;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary see;
Ilbrary | F.35 ABS Ilibrary vect;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary work;
Ilbrary see;
Ilbrary | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - TryPE bECLANATIONS
F.35.1 abs8.vhd - TryPE bECLANATIONS
Ilbrary ieee:
Ilbrary iee:
Ilbrary iee:
Il | F.35 ABS
F.35 abs/hd - Ilibrary ieee;
library work:
F.35.1 abs8.hd - F.35.1 abs8.hd - F.55.1 abs8.hd - F.

 | F.35 ABS F.35 ABS Ilbrary work; Ilbrary; Ilbrary
 | F.35 ABS
F.35 ABS
F.35. abs8.whd
F.35. abs8.s
F. CONFORMENT DECLANATIONS
F. CONFORME
 | F.35 ABS
F.35 ABS
F.35 abs
F.35. abs8.whd
F.35. abs8.whd
F.35. abs8.whd
F.35. abs8.whd
F.35. abs8.whd
F.35. abs8.whd
F.35. abs8.wh
F.35. abs8.
 | F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""><td>F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""><td>F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""><td>F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""></t<></td></t<></td></t<></td></t<> | F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""><td>F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""><td>F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""></t<></td></t<></td></t<>
 | F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""><td>F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""></t<></td></t<> | F.35 ABS F.35 ABS Ilbrary ice; F.35.1 abs8.vhd - Ilbrary work; F.35.1 abs8.vhd - - F.35.1 abs8.vhd - - Ibrary ice; Ilbrary work; - Ibrary ice; - - - - Ibrary ice; - - - - - - Ibrary ice; - <t< td=""></t<> | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS
F.35 abs8vhd - Ilibrary veck
F.351 abs8vhd - Ilibrary work
F.351 abs8vhd - Ilibrary work
Ibrary Ibrary
Ibrary Ibrary
Ibrary Ibrary Ibrary
Ibrary Ibrary
Ibrary
Ibrary Ibrary
Ibrar
 | F.35 ABS
F.35 ABS
F.35. abs3.vhd - Ilibrary work:
Ilibrary work:
use ieee.std_logic_1164.ALU;
use ieee.
 | F.35 ABS F.35 ABS F.35. absS.whd - F.35.1 absS.whd - F.35.1 absS.whd - T.35.1 absS.whd - T.35.1 absS.whd - T.35.1 absS.whd - Tibrary teee; - Total transformer and - Total transformer and - Total transformer transtormer transformer transtormer transformer t
 | F.35 ABS F.35 ABS F.35 ABS Ilibrary verk; F.35.1 abs8.whd - F.35.1 abs8.whd - F.35.1 abs8.whd - F.35.1 abs8.whd - Tibrary iee; use ieee.std_logic_l164.AU; aust ieee.std_logic_l164.AU; aust ieee.std_logic_l164.AU; aust is - aust is - component unit -
 | F.35 ABS Ilibrary ieee:
Ilibrary work;
use ieee: std_logic_l164.ALU;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use is scom | F.35 ABS Ilibrary ieee:
Ilibrary work;
use ieee: std_logic_l164.ALU;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use is scom
 | F.35 ABS Ilibrary ieee:
Ilibrary work;
use ieee: std_logic_l164.ALU;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use is scom | F.35 ABS Ilibrary ieee:
Ilibrary work;
use ieee: std_logic_l164.ALU;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use in NTPGER range 0 to +1271;
use is scomonent und?
use is scom | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Tyre backararious itees.std.logic_l164.ALC;
use itees.std. | F.35 ABS8
F.35 ABS8
F.35. abs8.vnd
architecture structured of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS

 | F.35 ABS
F.35 ABS
F.35. abs. work:
use ieee.std_logic_lid.ALL;
use ieee.std_logic_lid.ALL;
architecture structured of abs8 is
STGMAL DECLANATIONS
STGMAL DECLANAT
 | F.35 ABS
F.35 ABS
F.35.1 abs8.vhd - 11brary work:
Ibrary work:
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
. TYPE DECLARATIONS
- COMPONENT DECLAR
 | F.35 ABS
F.35 ABS
F.35. abs3.vhd - Ilibrary ieee;
F.35.1 abs3.vhd - Type BECLANTIONS
Ibrary ieee;
Ibrary ieee;
- TYPE DECLANTIONS
- TYPE DECLANTIO
 | F.35 ABS
F.35 ABS
F.35. abs/hd - Ilbrary work;
Ilbrary work;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
architecture structured of abs8 is
- StGMAN DECLANATIONS
- STGMAN | F.35 ABS
F.35 ABS
F.35. abs/hd - Ilbrary work;
Ilbrary work;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
architecture structured of abs8 is
- StGMAN DECLANATIONS
- STGMAN | F.35 ABS
F.35 ABS
F.35. abs/hd - Ilbrary work;
Ilbrary work;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
architecture structured of abs8 is
- StGMAN DECLANATIONS
- STGMAN | F.35 ABS
F.35 ABS
F.35. abs/hd - Ilbrary work;
Ilbrary work;
use std_logic_1164.ALL;
use std_logic_1164.ALL;
architecture structured of abs8 is
- STGMAN DECLANATIONS
- STGMAN D |
 |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd • Tyre Decreating the architecture structured of abs8 is
ibrary work;
use isee std_logic_1164.all;
use isee std_logic_1164.all;
use isee std_logic_1164.all;
use isee std_logic_1164.all;
architecture structured of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- COMPONENT DATS
- COMPONENT DATS
- COMPONENT DATS
- COMPONENT DATS
- COMPONENT DATS
- INLINE CONTROL ADD
- INLINE
- INL | F.35 ABS8
F.35 abs8
F.35.1 abs8.vnd
F.35.1 abs8.vnd
Ilbrary teee:
Ilbrary teee:
Ilbrary work:
use teee.std_logic_1164.ALU;
architecture structurel of abs8 is
STGMAL DECLARATIONS
STGMAL DECLARATIONS
STGM

 | F.35 ABS
F.35 ABS
F.35 abs/hd - Ilbrary work;
Ilbrary work;
use ieee:std_logic_l164.ALL;
use ieee.std_logic_l164.ALL;
architecture structured of Abs8 is
- STORP DECARATIONS
- STORP DECAR
 | F.35 ABS
F.35 ABS
F.35 abs8whd - Ilbrary work;
Ilbrary work;
Ilbrary ieee:
Ilbrary ieee:
Ilbrary ieee:
Ilbrary ieee:
Ilbrary ieee:
- 57000 Deficiled ALD;
architecture structured of abs8 is
- 57000 Deficiled ALD;
architecture structures structured of abs8 is
- 57000 Deficiled ALD;
architecture structures structures of abs8 is
- 570000 Deficiled ALD;
- 570000 Deficiled ALD;
- 570000 Deficiled ALD;
- 77000000 Deficiled ALD;
- 770000000 Deficiled ALD;
- 7700000000 Deficiled ALD;
- 77000000000 Deficiled ALD;
- 7700000000000000000000000000000000000
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
Ibrary ieee;
Ibrary iee;
Ibrary iee;
Ibra
 | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs8.vhd = 11breary work;
Ibreary work;
Ibreary work;
Ibreary work;
architecture structurel of abs8 is
- 17WB DECLANATIONS
- 17WB DE | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs8.vhd = 11breary work;
Ibreary work;
Ibreary work;
Ibreary work;
architecture structurel of abs8 is
- 17WB DECLANATIONS
- 17WB DE | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs8.vhd = 11breary work;
Ibreary work;
Ibreary work;
Ibreary work;
architecture structurel of abs8 is
- 17WB DECLANATIONS
- 17WB DE | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs8.vhd = 11breary work;
Ibreary work;
Ibreary work;
Ibreary work;
architecture structurel of abs8 is
- 17WB DECLANATIONS
- 17WB DE |
 |
| F.35 ABS8
Ilibrary viee;
F.35.1 abs8.vhd - TYPE DECLANATIONS
Ilibrary ieee; rd_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
architecture structurel of abs8 is
TYPE DECLANATIONS
TYPE DECLANATIONS
TWPE DECLANATI | F.35 ABS
F.35 abs/hd - Ibrary ieee;
F.35.1 abs8.vhd - Ibrary work;
use ieee std_logic_l164.ALU;
use ieee std_logic_l164.ALU;
architecture BEHAVE of Abs8 is
component
intv
component intv
component intv
component intv
component intv
component intv
component uc2
component uc2
com
 | F.35 ABS
F.35 ABS
F.351 abs8.vhd - Ilbrary work;
Ilbrary work;
use ieee.std_logic_1164.ALU;
architecture structurel of abs8 is
- STGNAL DECLANATIONS
- STG
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
Ibrary iee;
Ibrary iee;
I
 | F.35 ABS8
F.35 ABS8
F.35 abs8.vhd • [library veck;
library veck;
library ieee;
library iee;
library ise;
library ise;
libr
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
usteristeet structurel of abs8 is
- TYPE DECLARATIONS
- STGML | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
usteristeet structurel of abs8 is
- TYPE DECLARATIONS
- STGML | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
usteristeet structurel of abs8 is
- TYPE DECLARATIONS
- STGML | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
usteristeet structurel of abs8 is
- TYPE DECLARATIONS
- STGML | |
| F.35 ABS8
library work is tee;
library work is tee;
library work is tee;
use ieee.std_logic_l164.AL;
use ieee.std_logic_l164.AL;
component inv
component inv | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd TYPE DECLANATIONS
Ilbrary ieee:
Ilbrary iee:
Ilbrary iee:
Ilbrary iee:
Ilbrary iee:
Ilbrary iee:
Ilbrary iee:
Ilbrary ieee:
Ilbrary iee:
Ilbrary iee:
Ilb

 | F.35 ABS8
F.35 ABS8
F.351 abs8.vhd - Ibbrary work;
Ibbrary work;
use ieee strd_logic_1164.ALF;
architecture structurel of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DATA
- TYPE DECLANATIONS
- COMPONENT DATA
- COMPONENT DATA
- COMPONENT DATA
- COMPONENT DATA
- COMPONENT DATA
- TALE
- TALE
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd ~ [library work;
F.35.1 abs8.vhd ~ [library work;
remitecture structured. ALS;
library ieee;
use ieee.std_logic_1164.ALS;
use ieee.std_logic_1164.ALS;
use ieee.std_logic_1164.ALS;
use ieee.std_logic_1164.ALS;
use ieee.std_logic_1164.ALS;
use ieee.std_logic_1164.ALS;
component back
for ty ABS8 is
component back
component back
component back
component back
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
process(x)
pro
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd • Ilbrary veck:
Ilbrary veck:
use ieee.std_logic_1164.ALV;
are itee.std_logic_1164.ALV;
are iterested of abs8 is
• TYPE DECLANATIONS
• TYPE DECLANATIO
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · [11bhary work;
restitecture structured of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE - TABLE | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · [11bhary work;
restitecture structured of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE - TABLE | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · [11bhary work;
restitecture structured of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE - TABLE
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · [11bhary work;
restitecture structured of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- TABLE - TYPE - TABLE | |
| F.35 ABS8
library work
F.35.1 abs8.vhd - TYPE DECLARATIONS
Is in the event of abs8 is
use ieee.std_logic_l164.ALF;
use ieee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_logic_l164.ALF;
isee.std_log
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35 ABS8
F.351 abs8.vhd - Eiler and Eiler EILERATIONS
Ibrary ieee; std_logic_1164.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
architecture BEHAVE of Abs8 is
Trype DECLARATIONS
Trype EILERATIONS
Trype EILERATIONS
Component invr
Component invr
Component or 2
Component or 2

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ibtrary work;
Ibtrary ieee;
use ieee.std_logic_1164.ALF;
use ieee.std_logic_1164.ALF;
use ieee.std_logic_1164.ALF;
use ieee.std_logic_1164.ALF;
use ieee.std_logic_1164.ALF;
use ieee.std_logic_1164.ALF;
architecture structurel of abs8 is
- TYPE DECLARATIONS
- TYPE DECL
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ilbrary work;
Ilbrary ieee: std.logic_1164.ALL;
use ieee.std.logic_1164.ALL;
use ieee.std.logic_1164.ALL;
arrie Dec.RELARATIONS
- TYTED ENCLARATIONS
- TYTED ENC
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work:
Ilbrary ieee: retallogic_1164.ALb;
architecture a structured of abs8 is
- TYPE DECLARATIONS
- | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work:
Ilbrary ieee: retallogic_1164.ALb;
architecture a structured of abs8 is
- TYPE DECLARATIONS
-
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work:
Ilbrary ieee: retallogic_1164.ALb;
architecture a structured of abs8 is
- TYPE DECLARATIONS
- | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work:
Ilbrary ieee: retallogic_1164.ALb;
architecture a structured of abs8 is
- TYPE DECLARATIONS
- | |
| F.35 ABS8
F.35 ABS8
F.35. Ibbrary were;
Ilbrary were;
Ilbrary were;
Ilbrary were;
Ilbrary were;
were ieee.std_logic_1164.all;
were ieee.std_logic_1 | F.35 ABS
F.35 ABS
F.35.1 abS.vhd
F.35.1 abS

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - ===================================
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary verk;
Ilibrary ieee;
Ilibrary ieee;
Ilibrary ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
- TYPE DERARATIONS
- SIGNAL DECLARATIONS
- SIGNAL DECLARATION
- SIGNAL DECL
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary veck;
Ilibrary ieee;
Ilibrary iee;
Ilibrary iee;
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd |
 |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - F.35.1 a | F.35 ABS8
Ilibrary ieee;
Ilibrary vice:
Ilibrary vice:
Ilibrary vice:
Ilibrary vice:
Ilibrary ieee;
Ilibrary ieee;
use ieee std_logic_1164.all;
architecture structure1 of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATION
- TYPE DECLANATION
- TYPE DECL

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd -
library ieee;
library ieee;
library ieee;
library ieee;
library ieee;
library ieee;
-
ryre bechAATTONS
-
component buf
component buf
component inv
component
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ilibrary veck;
use ieee.std_logic_1164.ALL;
architecture structurel of abs8 is
- TYPE DECLANATIONS
- STGNAL DECLANATION
- STG | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ilibrary veck;
use ieee.std_logic_1164.ALL;
architecture structurel of abs8 is
- TYPE DECLANATIONS
- STGNAL DECLANATION
- STG
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ilibrary veck;
use ieee.std_logic_1164.ALL;
architecture structurel of abs8 is
- TYPE DECLANATIONS
- STGNAL DECLANATION
- STG | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ilibrary veck;
use ieee.std_logic_1164.ALL;
architecture structurel of abs8 is
- TYPE DECLANATIONS
- STGNAL DECLANATION
- STG | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - F.35.1 abs8.who - F.35.1 a | F.35 ABS
F.35 ABS
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.works
architecture structurel of abs8 is
TYE DECLANATIONS
TYE DECLANATIONS

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
Ilbrary ieee;
Ilbrary ieee;
use ieee.std_logic_l164.all;
use ieee.std_logic_l164.all;
use ieee.std_logic_l164.all;
use ieee.std_logic_l164.all;
= TYPE DECLARATIONS
= STORAL DECLARATIONS
= STORAL DECLARATIONS
= COMPONENT_DECLARATIONS
= COMPONENT_DECLARATIONS
= COMPONENT and 2
component and 2
component inv
component inv
component nor2p
component nor2p
component nor2p
component nor2p
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intrary work;
Intrary ieee;
Intrary ieee;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary vork;
Ilibrary ieee;
Ilibrary ieee;
Ilibrary ieee;
use ieee.std_logic_l164.ALL;
use ieee.std_logic_l164.ALL;
architecture structured of abs8 is
- TYPE DECLARATIONS
- Stock DECLARATIONS
- COMPONENT and2
- COMPONENT and2
- COMPONENT and2
- COMPONENT and2
- COMPONENT and2
- COMPONENT invr
- COMPONENT
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
Ilbrary ieee;
Ilbrary ieee;
use ieee.std_logic_1164.ALU;
arcchitecture structurel of abs8 is
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT and
- COMPONENT and
- COMPONENT inv
- COMPONENT INV | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
Ilbrary ieee;
Ilbrary ieee;
use ieee.std_logic_1164.ALU;
arcchitecture structurel of abs8 is
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT and
- COMPONENT and
- COMPONENT inv
- COMPONENT INV | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
Ilbrary ieee;
Ilbrary ieee;
use ieee.std_logic_1164.ALU;
arcchitecture structurel of abs8 is
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT and
- COMPONENT and
- COMPONENT inv
- COMPONENT INV
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
Ilbrary ieee;
Ilbrary ieee;
use ieee.std_logic_1164.ALU;
arcchitecture structurel of abs8 is
- TYPE DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT and
- COMPONENT and
- COMPONENT inv
- COMPONENT INVE
- COMPONENT INV
- COMPONENT INVE
- COMPONENT INV
- COMPONENT I | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - F.35.1 a | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs.whd
F.35.1 abs.whd

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Ilbrary work;
Ilbrary ieee;
Ilbrary ieee;
use ieee.std_logic_1164.ALU;
architecture structured of abs8 is
TYPE DECLARATIONS
TYPE DECL
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Type Escienter structured of abs8 is
- Type DECLANATIONS
- TYPE DECL
 | F.35 ABS8
F.35.1 abs8.whd - Type Sec.atd_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
. TYPE BECLARATIONS
- TYPE DECLARATIONS
- STORAL DECLARATION
- STORAL DECLARATION
- STORAL DECLARATION
- STORAL DECLARAT
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Trpe fact.atary work;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.all;
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Trpe fact.atary work;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.all;
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Trpe fact.atary work;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.all;
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL
DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Trpe fact.atary work;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.ALU;
use ieee.std_logic_1164.all;
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
STORNL DECLARATIONS
Trpe fact.ARATIONS
Trpe fact.ARATIONS
 | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
F.35.1 abs8.vhd - Studiet ALF,
is in the studiet of abs8 is
is component structure of abs8 is
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT INTEGER range -127 to +127/
port(X: in INTEGER range 0 to +127/);
end ABS8;
end ABS8;
end ABS8 is
component inv
component inv
c | F.35 ABSIlbrary iee;F.35 ABSIlbrary veck;F.35.1 abs.whd-F.35.1 abs.whd-rechitecture structurel of abs8 is

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.whd
F.35.1 abs8.is
F.35.1 a
 | F.35 ABS8library icee;F.35 ABS8library work;F.35.1 abs8.vhd-F.35.1 abs8.vhd-Tibrary icee;-library icee;-library icee;-library icee;-refice std_logic_l164.ALb;use icee std_l065.alb;<
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd · Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
restormer becomponent on and
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
becin to recomponent inv
component inv
c
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd · Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
restormer becomponent on and
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
becin to recomponent inv
component inv
c | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd · Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
restormer becomponent on and
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
becin to recomponent inv
component inv
c | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd · Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
use ieee std_logic_1164.ALL;
restormer becomponent on and
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
end ABS8 is
port (X: in INTEGER range 0 to +127);
becin to recomponent inv
component inv
c | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary stee;
F.35.1 abs8.vhd - Std_logic_1164.AL;
use iese.std_logic_1164.Al;
use iese.std_logic_127, use iese.std_logic_127, use iese.std_logic_127, use iese.std_logic_127, | F.35 ABS
F.35 ABS
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.whd
F.35.1 abs.worth
architecture structure1 of abs8 is
- TYPE DECLANTIONS
- COMPONENT DECLANTIONS
- COMPONENT ange -127 to +127;
pot (X ISB is
- COMPONENT and 2
- COMPONENT inv
- COMPONE

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd -
library work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture 1 f abs8 is
TYPE DECLANATIONS
TYPE DECLANATIONS
STGNAL DECLANATION
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35.1 abs8.whd = 11 ibrary work;
11 ibrary work;
11 ibrary work;
12.5.1 abs8.whd = 1000.1164.ALL;
12.5.1 abs8.whd = 1000.1164.ALL;
11 arst is received of abs8 is
11 ibrary ieee;
12.5 component of abs8 is
11 ibrary work;
12 ibrary ieee;
13.5.1 abs8.whd = 127 to +127;
13.5 ibrary ieee;
13.5 ibrary ieee;
14.5 ibrary ieee;
15.5 ibrary iee;
15.5 ibrary ier;
15.5 ibrary iee;
15.5 ibrary ier;
15.5
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd • Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture a structure of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- COPTONENT DECLARATIONS
- COPTONENT DECLARATIONS
- COMPONENT buf
component buf
component inv
component inv
component inv
component nor4p
component nor4p
component nor4p | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd • Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture a structure of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- COPTONENT DECLARATIONS
- COPTONENT DECLARATIONS
- COMPONENT buf
component buf
component inv
component inv
component inv
component nor4p
component nor4p
component nor4p
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd • Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture a structure of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- COPTONENT DECLARATIONS
- COPTONENT DECLARATIONS
- COMPONENT buf
component buf
component inv
component inv
component inv
component nor4p
component nor4p
component nor4p | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd • Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture a structure of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- COPTONENT DECLARATIONS
- COPTONENT DECLARATIONS
- COMPONENT buf
component buf
component inv
component inv
component inv
component nor4p
component nor4p
component nor4p |
 |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - F.35.1 a | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs.whd
F.35.1 ab
 |
F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Ilibrary work;
F.35.1 abs8.whd - F.35.1 abs8.w
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Type Declarations of abs8 is
- Type DECLANTIONS
- Type DECLANTION
- Type DECLA
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Type Sec.atd_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
. Type BECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- STONAL DECLANATIONS
- COMPONENT DECLANATIONS
- COMPONENT DAC
component buf
component inv
component inv
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Trpe Declatary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
Trpe DECLANATIONS
- Trpe DECLANATIONS
Trpe DECLANATIONS
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Trpe Declatary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
Trpe DECLANATIONS
- Trpe DECLANATIONS
Trpe DECLANATIONS
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Trpe Declatary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
Trpe DECLANATIONS
- Trpe DECLANATIONS
Trpe DECLANATIONS

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Trpe Declatary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
Trpe DECLANATIONS
- Trpe DECLANATIONS
Trpe DECLANATIONS
 | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Ilibrary work;
F.35.1 abs8.vhd - Extracture structures structures
interve ieee. std_logic_1164.ALb;
use ieee. std_logic_1164.ALb;
use ieee. std_logic_1164.ALb;
architecture structures
= TYPE DECLARATIONS
= TYPE DECLARATIONS
= STGNAL DECLARATIONS
= S | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs.whd
F.35.1 ab

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type Beclatarrows
F.35.1 abs8.whd - Type Beclatarrows
Ibbrary ieee:
- Type Beclatarrows
- Type Beclatarrows
- Type Beclatarrows
- STGMAL DECLATARTONS
- STGMAL D
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Type Science std_logic_1164 ALL;
use ieee std_logic_1164 ALL;
use ieee std_logic_1164.all;
- Type BecLANATIONS
- Type BecLANATIONS
- Type DecLANATIONS
- COMPONENT DESCIANATIONS
- COMPONENT buf
component and2
port(X: in INTEGER range 0 to +127);
end ABS8;
end ABS8;
component inv
component inv
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type becard logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture structured of abs8 is
TYPE PECLANATIONS
TYPE DECLANATIONS
TYPE DECLANATIONS
TY
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd |
 |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - F.35.1 abs8.who - F.35.1 a | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - F.35.1 a

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd -
library work;
use ieee.std_logic_l164.ALL;
use ieee.std_logic_l164.ALL;
architecture structurel of abs8 is
TYPE DECLANATIONS
TYPE DECLANATIONS
STGNAL DECLANAT
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35.1 abs8.whd = 11 ibrary work;
11 ibrary work;
12 ibrary work;
12 ibrary ieee;
13.5.1 abs8.whd = 124.logic_1164.aLL;
12 ibrary ieee;
13.5.1 abs8.whd = 124.all;
13.5.1 abs8.whd = 124.all;
13.5.1 abs8.whd = 127 to +127;
13.5 component inv
13.5 compo
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
architecture structure of abs8 is
TYPE PECLARATIONS
TYPE PECLARATIONS
TYPE PECLARATIONS
STGNAL DECLARATIONS
STGNAL DECLARATIONS | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
architecture structure of abs8 is
TYPE PECLARATIONS
TYPE PECLARATIONS
TYPE PECLARATIONS
STGNAL DECLARATIONS
STGNAL DECLARATIONS | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
architecture structure of abs8
is
TYPE PECLARATIONS
TYPE PECLARATIONS
TYPE PECLARATIONS
STGNAL DECLARATIONS
STGNAL DECLARATIONS | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
architecture structure of abs8 is
TYPE PECLARATIONS
TYPE PECLARATIONS
TYPE PECLARATIONS
STGNAL DECLARATIONS
STGNAL DECLARATIONS | |
| F.35 ABS8
11brary ieee;
F.35.1 abs8.vhd - Tyre BECLANATIONS
11brary ieee;
11brary iee;
11brary iee;
11br | F.35 ABS
F.35 ABS
F.35.1 abs8.whd
F.35.1 abs8.whd
F.3
 | F.35 ABS8
F.35
ABS8
F.35.1 abs8.vhd - Iibrary work;
restructures i logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
. TYPE DECLARATIONS
. TYPE DECLARATIONS
. TYPE DECLARATIONS
. TYPE DECLARATIONS
. TYPE DECLARATIONS
. TYPE DECLARATIONS
. CONFIRM DECLARATIONS
. CONFIRM DECLARATIONS
. COMPONENT DACE
component and?
. COMPONENT DACE
. COMPO
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intrary work;
Intrary ieee;
Intrary ieee;
Intrary ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
Intrary ieee;
use ieee.std_logic_1164.ALL;
Intrary ieee;
Intrary isee;
Intrary ise;
Intrary
 | F.35 ABS8
F.35.1 abs8.vhd = 11brary work;
11brary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
= 11brary ieee;
= = TYPE DECLARATIONS
= = TYPE DECLARAT
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intrary work;
restricted a logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
restricter structure1 of abs8 is
TYPE DECLARATIONS
TYPE DECLARATIONS
STOAL DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
Stout INTEGER range 0 to +127);
end ABS9;
end ABS9; | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intrary work;
restricted a logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
restricter structure1 of abs8 is
TYPE DECLARATIONS
TYPE DECLARATIONS
STOAL DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
COMPONENT DECLARATIONS
Stout INTEGER range 0 to +127);
end ABS9;
end ABS9;
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intrary work;
restricted a logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
restricter structure1 of abs8 is
restricter structure1 of abs8 is
restructure1 of abs8 is
restricter structure1 of abs8 is
restricter | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Intrary work;
restricted a logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
restricter structure1 of abs8 is
restricter structure1 of abs8 is
restructure1 of abs8 is
restricter structure1 of abs8 is
restricter | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8Iibrary ieee;F.35 ABS8Iibrary work;F.35.1 abs8.whd-F.35.1 abs8.whd-rechitecture structurel of abs8 is
 | F.35 ABS8Iibrary icee;F.35 ABS8Iibrary work;F.35.1 abs8.vhd-F.35.1 abs8.vhd-TYPE DECLARATIONSIbrary icee;-TYPE DECLARATIONS
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ibrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
. TYPE DECLANATIONS
- TYPE DECLANATIONS
- STONAL DECLANATIONATIONATIONATIONATIONATIONATIONATI
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type beclatarrows
F.35.1 abs8.whd - Type beclatarrows
= Type beclata
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type beclatarrows
F.35.1 abs8.whd - Type beclatarrows
= Type beclata | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type beclatarrows
F.35.1 abs8.whd - Type beclatarrows
= Type beclata | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type beclatarrows
F.35.1 abs8.whd - Type beclatarrows
= Type beclata | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - F.35.1 a | F.35 ABS
F.35 ABS
F.35 ABS
F.35.1 abs. Wid
F.35.1 abs. Wid
F.3

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd · Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
- TYPE PECLARATIONS
- TYPE PECLARATIONS
- TYPE PECLARATIONS
- COMPONENT DECLARATIONS
- COMPONENT below
entity ABS8 is
- TYPE DECLARATIONS
- COMPONENT DECLARATIONS
- COMPONENT below
- 2: out INTEGER range 0 to +127);
- 2: out ONTEGER range 0 to +127);
- 2: out ON
 | F.35 ABS8library ieee;F.35. ABS8library work;F.35.1 abs8.vhduse ieee.std_logic_l164.AL;use ieee.std_logic_l164.AL;use isee.std_logic_l164.AL;library ieee;
 | F.35 ABS8
F.35. ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - F.35.1 a | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - F.35.1 a

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35.1 abs8.whd • Ilbrary work;
Ilbrary work;
use ieee.std_logic_l164.ALL;
use ieee.std_logic_l164.ALL;
architecture stuctured of abs8 is
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- CONDUMENT DECLARATIONS
- CONDUMENT DECLARATIONS
- CONDUMENT ange -127 to +127;
port (X is) INTEGER range 0 to +127;
2: out INTEGER range 0 to +127;
3: out INTEG
 | F.35 ABS8
F.35.1 abs8.vhd = Iibrary work;
Isbrary work;
use ieee.std_logic_1164.ALb;
use ieee.std_logic_1164.ALb;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
= TYPE DECLARATIONS
= TYPE DECLARATIONS
= CONFORMER_DECLARATIONS
= CONFORMER_DECLARATIONS
= CONFORMER_TANGE = 127 to +127;
= OUT INTEGER range 0 to +127;
= OUT INTEGER r
 | F.35 ABS8 library work; 11brary work; use ieee.std_logic_l164.ALL; 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - SIGNAL DECLARATIONS 11brary iee; - SIGNAL DECLARATIONS | F.35 ABS8 library work; 11brary work; use ieee.std_logic_l164.ALL; 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - SIGNAL DECLARATIONS 11brary iee; - SIGNAL DECLARATIONS
 | F.35 ABS8 library work; 11brary work; use ieee.std_logic_l164.ALL; 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - SIGNAL DECLARATIONS 11brary iee; - SIGNAL DECLARATIONS | F.35 ABS8 library work; 11brary work; use ieee.std_logic_l164.ALL; 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - TYPE DECLARATIONS 11brary ieee; - SIGNAL DECLARATIONS 11brary iee; - SIGNAL DECLARATIONS | |
| F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8 Iibrary icee; F.35. abs8.vhd -
 | F.35 ABS8
F.35. ABS8
F.35.1 abs8.vhd
F.35.1 abs8.vhd
inbrary ieee:
use ieee.std_logic_1164.ALb;
architecture structure1 of abs8 is
- TYPE DECLANATIONS
- TYPE DECLANATIONS
- STONAL DECLANATIONS
- COMPONENT DECLANATIONS
- C
 | F.35 ABS8
F.35.1 abs8.vhd - Type Sec.atd_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
restructures f abs8 is
restructures f abs8 is
restr
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
F.35.1 abs8.vhd
r = TYPE DECLANATIONS
= TYPE DECLANATIONS
= TYPE DECLANATIONS
= TYPE DECLANATIONS
= CONFORMER DECLANATIONS
= CONFORMER DECLANATIONS
= CONFORMER and 2
component buf
component buf
component buf
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
F.35.1 abs8.whd
r = TYPE DECLANATIONS
= TYPE DECLANATIONS
= TYPE DECLANATIONS
= STORAL DECLANATI | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
F.35.1 abs8.whd
r = TYPE DECLANATIONS
= TYPE DECLANATIONS
= TYPE DECLANATIONS
= STORAL DECLANATI | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd
F.35.1 abs8.whd
r = TYPE DECLANATIONS
= TYPE DECLANATIONS
= TYPE DECLANATIONS
= STORAL DECLANATI | |
| F.35 ABS8 library work;
11.15rary work;
F.35.1 abs8.vhd - restanceure structurel of abs8 is
11.15rary ieee;
11.15rary iee | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd

 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35.1 abs8.whd - Ibrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
use isee.std_logic_1164.all;
- TYPE PECLARATIONS
- TYPE PECLARATIONS
- STONAL DECLARATIONS
- STONAL DECLARATIONS
- COMPONENT and 2
component buf
component buf
 | F.35 ABS8
F.35.1 abs8.whd - Ibrary work;
use iee.std_logic_l164.ALL;
use iee.std_logic_l164.all;
use iee.std_logic_l164.all;
use iee.std_logic_l164.all;
= TYPE DECLANATIONS
= TYPE DECLANATIONS
= COMPONENT DECLANATIONS
= COMPONENT and 2
component buf
component buf
 | F.35 ABS8
F.35.1 abs8.whd - Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
use isee.std_logic_1164.all;
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- STGNAL DECLARATIONS
- STGNAL DECLARATIONS
- COMPONENT and 2
component buf
component buf
 | F.35 ABS8
F.35.1 abs8.whd - Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
use isee.std_logic_1164.all;
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- STGNAL DECLARATIONS
- STGNAL DECLARATIONS
- COMPONENT and 2
component buf
component buf | F.35 ABS8
F.35.1 abs8.whd - Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
use isee.std_logic_1164.all;
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- STGNAL DECLARATIONS
- STGNAL DECLARATIONS
- COMPONENT and 2
component buf
component buf
 | F.35 ABS8
F.35.1 abs8.whd - Ibbrary work;
use ieee.std_logic_1164.ALL;
use isee.std_logic_1164.ALL;
use isee.std_logic_1164.all;
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- STGNAL DECLARATIONS
- STGNAL DECLARATIONS
- COMPONENT and 2
component buf
component buf | |
| F.35 ABS8 library icee; library icee; library icee; garchitecture structurel of abs8 is architecture structurel of abs8 is enclose garchitecture structurel of abs8 is garchitecture structurel garchitecturel <
 | F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
use ieee.std_logic_l164.all;
use ieee.std_logic_l164.all;
use ieee.std_logic_l164.all;
entity ABS8 is
- TYPE DECLARATIONS
- CONFONERT_DECLARATIONS
- CONFONERT_DECLARATIONS
 | F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
- TYPE DECLARATIONS
- TYPE DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS
 | F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
architecture structurel of abs8 is
- TYPE DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS
entity ABS8 is
component and2
component and2
 | F.35 ABS8
F.35.1 abs8.vhd · Iibrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
· TYPE DECLARATIONS
· TYPE DECLARATIONS
· COMPONENT_DECLARATIONS
· COMPONENT_DECLARATIONS
 | F.35 ABS8
F.35.1 abs8.vhd · Iibrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
· TYPE DECLARATIONS
· TYPE DECLARATIONS
· COMPONENT_DECLARATIONS
· COMPONENT_DECLARATIONS | F.35 ABS8
F.35.1 abs8.vhd · Iibrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
· TYPE DECLARATIONS
· TYPE DECLARATIONS
· COMPONENT_DECLARATIONS
· COMPONENT_DECLARATIONS | F.35 ABS8
F.35.1 abs8.vhd · Iibrary work;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_1164.all;
· TYPE DECLARATIONS
· TYPE DECLARATIONS
· COMPONENT_DECLARATIONS
· COMPONENT_DECLARATIONS | |
| F.35 ABS8 library ieee; library work; B F.35.1 abs8.whd - TYPE DECLARATIONS B B 11brary ieee; - TYPE DECLARATIONS S B B 11brary ieee; - TYPE DECLARATIONS - - B B entity ABS8 is - - COMPONENT_DECLARATIONS - - COMPONENT_DECLARATIONS S
 | F.35 ABS8 F.35 ABS8 F.35 ABS8 Ilbrary work;
Ilbrary work;
work; Ilbrary work;
use else std_logic_1164.ALb;
work; Establic Establic Egg Egg <t< td=""><td>F.35 ABS8 library icee; library work; gee; gef gef</td><td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Fibrary work;
use ieee.std_logic_1164.ALb;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
entity ABS8 is
entity ABS8 is
entity ABS8 is</td><td>F.35 ABS8
F.35.1 abs8.vhd • Iibrary work;
Iibrary ieee;
use ieee.std_logic_1164.ALb;
efficture structure1 of abs8 is
- TYPE DECLANATIONS
• FICANATIONS
• CONFORMER_DECLANATIONS
• CONFORMER_DECLANATIONS
• CONFORMER_DECLANATIONS
• CONFORMER_DECLANATIONS</td><td>F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALb; TYPE DECLARATIONS contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is entity ABS8 is component and</td><td>F.35 ABS8 library ieee; F.35.1 abs8.vhd rese.std_logic_l164.ALb; F.35.1 abs8.vhd rese.std_logic_l164.ALb; use ieee.std_logic_l164.alb; component of abs8 is entity ABS8 is component and</td><td>F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALb; TYPE DECLARATIONS contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is entity ABS8 is component and</td><td>F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALb; TYPE DECLARATIONS contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is entity ABS8 is component and</td></t<>
 | F.35 ABS8 library icee; library work; gee; gef
 | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd - Fibrary work;
use ieee.std_logic_1164.ALb;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
entity ABS8 is
entity ABS8 is
entity ABS8 is
 | F.35 ABS8
F.35.1 abs8.vhd • Iibrary work;
Iibrary ieee;
use ieee.std_logic_1164.ALb;
efficture structure1 of abs8 is
- TYPE DECLANATIONS
• FICANATIONS
• CONFORMER_DECLANATIONS
• CONFORMER_DECLANATIONS
• CONFORMER_DECLANATIONS
• CONFORMER_DECLANATIONS
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALb; TYPE DECLARATIONS contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is entity ABS8 is component and
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd rese.std_logic_l164.ALb; F.35.1 abs8.vhd rese.std_logic_l164.ALb; use ieee.std_logic_l164.alb; component of abs8 is entity ABS8 is component and | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALb; TYPE DECLARATIONS contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is entity ABS8 is component and
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALb; TYPE DECLARATIONS contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is Library ieee; contineeture 1 of abs8 is entity ABS8 is component and | |
| F.35 ABS8 Thibrary ieee;
Thibrary work;
F.35.1 abs8.vhd - TYPE DECLARATIONS
Therewise ieee: std_logic_1164.all;
architecture std_logic_1164.all;
TYPE DECLARATIONS
- TYPE DECLARATIONS
- COFFORMER_DECLARATIONS
- COFFORME | F.35 ABS8 Iibrary ieee; F.35.1 abs8.vhd -
 -
 | F.35 ABS8 library icce; F.35.1 abs8.vhd rest studies in the studies of abs8 is F.35.1 abs8.vhd rest studies of abs8 is Ibrary icce; rest studies of abs8 is rest std_logic_li64.all; rest studies of abs8 is rest std_logic_li64.all; rest std_logic_li64.all; std
 | F.35 ABS8 library icee; F.35. ABS8 library work; F.35.1 abs8.vhd restructures for abs8 is F.35.1 abs8.vhd restructures for abs8 is Ibrary icee; restructures for abs8 is restructures for abs8 is restructures restructures restructures restructures <threstructures< td="" the<=""><td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type Standthet of abs8 is
 Type BCLANATIONS Type Standthet of abs8 is
 Type BCLANATIONS STONAL DECLANATIONS STONAL DECLANATIONS STONAL DECLANATIONS COMPONENT_DECLANATIONS COMPONENT_DECLANATION</td><td>F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
 TYPE DECLANATIONS
 STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
 STGNAL DECLANATIONS
 STGNAL DECLANATIONS
 COMPONENT DECLANATIONS
 STGNAL DECLANATIONS
 S</td><td>F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
 TYPE DECLANATIONS
 STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
 STGNAL DECLANATIONS
 STGNAL DECLANATIONS
 COMPONENT DECLANATIONS
 STGNAL DECLANATIONS
 S</td><td>F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
 TYPE DECLANATIONS
 STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
 STGNAL DECLANATIONS
 STGNAL DECLANATIONS
 COMPONENT DECLANATIONS
 STGNAL DECLANATIONS
 S</td><td>F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
 TYPE DECLANATIONS
 STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
 STGNAL DECLANATIONS
 STGNAL DECLANATIONS
 COMPONENT DECLANATIONS
 STGNAL DECLANATIONS
 S</td></threstructures<> | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type Standthet of abs8 is
Type BCLANATIONS Type Standthet of abs8 is
Type BCLANATIONS STONAL DECLANATIONS STONAL DECLANATIONS STONAL DECLANATIONS COMPONENT_DECLANATIONS COMPONENT_DECLANATION
 | F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
TYPE DECLANATIONS
STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
STGNAL DECLANATIONS
STGNAL DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
STGNAL DECLANATIONS
S
 | F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
TYPE DECLANATIONS
STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
STGNAL DECLANATIONS
STGNAL DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
STGNAL DECLANATIONS
S | F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
TYPE DECLANATIONS
STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
STGNAL DECLANATIONS
STGNAL DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
STGNAL DECLANATIONS
S | F.35 ABS8 library work;
library work;
were std_logic_l164.ALb;
use ieee.std_logic_l164.ALb;
architecture structurel of abs8 is
TYPE DECLANATIONS
STGNAL DECLANATIONS
were ieee.std_logic_l164.All;
STGNAL DECLANATIONS
STGNAL DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
COMPONENT DECLANATIONS
STGNAL DECLANATIONS
S | |
| F.35 ABS8 library icee; library icee; Bibicary work; < | F.35 ABS8 Iibrary ieee; F.35 ABS8 Iibrary work; Iibrary work; Iibrary work; use ieee.std_logic_l164.ML; Eccentration of abs8 is TYPE DECLARATIONS SS Ibrary ieee; Component and addition | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd
reference std_logic_1164.ALL;
architecture structure of abs8 is
TYPE DECLARATIONS
STGNAL DECLARATIONS
STGNAL DECLARATIONS
STGNAL DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS
COMPONENT_DECLARATIONS | F.35 ABS8 library work; library work; library work; gas gas <th gas<="" td="" th<=""><td>F.35 ABS8 library work;
library work; library work; state gas g</td><td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td><td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td><td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td><td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td></th> | <td>F.35 ABS8 library work;
library work; library work; state gas g</td> <td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td> <td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td> <td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td> <td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
-</td> | F.35 ABS8 library work;
library work; library work; state gas g | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
- | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
- | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
- | F.35 ABS8
F.35 ABS8
F.35.1 abs8.whd - Type DECLANATIONS
- Type DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT DECLANATIONS
- |
| F.35 ABS8 library ieee; library wide B F.35.1 abs8.whd - - TYPE DECLANTIONS B library ieee; - - TYPE DECLANTIONS B use ieee.std_logic_l164.all; - - COMPONENT_DECLANTIONS B
 | F.35 ABS8 Iibrary ieee; Iibrary work; Iibrary work; Iibrary work; Iibrary work; Iibrary iee; Iibrary ie
 | F.35 ABS8 Iibrary ieee; Iibrary work; Bibrary work; Bibrary work; Bibrary ieee; Bibrary iee; Bibrary iei; Bibrary
 | F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - Structure structure of abs8 is
- TYPE DECLANATIONS
- SIGNAL DECLANATIONS
- COMPONENT_DECLANATIONS
- COMPONENT_DECLANATIONS
 | F.35 ABS8 library work; library work; library work; BS F.35.1 abs8.vhd - ree.std_logic_li64.ALL; BS BS F.35.1 abs8.vhd - ree.std_logic_li64.ALL; BS BS Ibrary icee; - ree.std_logic_li64.all; BS BS </td <td>F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS</td> <td>F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS</td> <td>F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS</td> <td>F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS</td>
 | F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS
 | F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS | F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS
 | F.35 ABS8
F.35.1 abs8.vhd - Iibrary work;
r.35.1 abs8.vhd - architecture structurel of abs8 is
- TYPE DECLARATIONS
- SIGNL DECLARATIONS
- COMPONENT_DECLARATIONS
- COMPONENT_DECLARATIONS | |
| F.35 ABS8 library ieee; library work; B F.35.1 abs8.whd - rested_logic_l164.ALL; B B F.35.1 abs8.whd - rested_logic_l164.ALL; B B B Ibrary ieee; rtrpe DECLARATIONS - rtrpe DECLARATIONS S S S use ieee.std_logic_l164.all; - rtrpe DECLARATIONS - CONFIDERTARTIONS S S
 | F.35 ABS8 F.35 ABS8 F.35 ABS8 library work; F.35.1 abs8.vhd - rese.std_logic_l164.ALL; - use ieee.std_logic_l164.all; - use ieee.std_logic_l164.all; -
 | F.35 ABS8 Iibrary ieee; Iibrary work; BE BE <td>F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd Type Declarations Type Declarations
acchietecture structured of abs8 is
 Type Declarations
 STORAL DECLARATIONS
 COMPORED DECLARATIONS
 COMPORED DECLARATIONS
 COMPORED DECLARATIONS</td> <td>F.35 ABS8
F.35.1 abs8.vhd - Type Bechatariows
architecture studiet ALS
11brary ieee:
use ieee.std_logic_1164.ALS
= Type Bechatariows
= Type Bechatariows
= SIGMAL DECLANARTOWS
= CONFORT DECLANARTOWS</td> <td>F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra</td> <td>F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra</td> <td>F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra</td> <td>F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra</td> | F.35 ABS8
F.35 ABS8
F.35.1 abs8.vhd Type Declarations Type Declarations
acchietecture structured of abs8 is
Type Declarations
STORAL DECLARATIONS
COMPORED DECLARATIONS
COMPORED DECLARATIONS
COMPORED DECLARATIONS

 | F.35 ABS8
F.35.1 abs8.vhd - Type Bechatariows
architecture studiet ALS
11brary ieee:
use ieee.std_logic_1164.ALS
= Type Bechatariows
= Type Bechatariows
= SIGMAL DECLANARTOWS
= CONFORT DECLANARTOWS
 | F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra | F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra | F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee; library lee; library lee; library lee; library lee; library lee; library libra | F.35 ABS8 library work; library lee; std_logic_ll64.ALb; library lee;
library lee; library lee; library lee; library lee; library lee; library libra | |
| F.35 ABS8 [library work,
library work,
F.35.1 abs8.vhd - Type beckaratrouch of abs8 is
architecture sturctures of abs8 is
library lees; stdel opic_1164.all;
- Type beckaratrous
use isees; stdel opic_1164.all;
- STGMAL DECLARATIONS
 | F.35 ABS8 Iibrary icee; F.35.1 abs8.vhd use icee.std.logic_1164.ALL; TYPE DECLANATIONS cschictecture 1 of abs8 is
 | F.35 ABS8 library work;
library work;
r.35.1 abs8.vhd - TYPE bECLANTIONS - TYPE BECLANTIONS - TYPE BECLANTIONS - STONAL DECLANATIONS
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALL; BB F.35.1 abs8.vhd - TYPE Extructure of abs8 is SS library ieee; - TYPE Extructure of abs8 is SS SS use ieee.std_logic_l164.all; - SIMAL bECLANATIONS SS SS
 | F.35 ABS8 library work; library work; BE
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALL; BB F.35.1 abs8.vhd - TYPE psclatharlows SS Library ieee; - TYPE psclatharlows SS SS use ieee.std_logic_l164.all; - SS SS SS
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALL; BB F.35.1 abs8.vhd - TYPE psclatharlows SS Library ieee; - TYPE psclatharlows SS SS use ieee.std_logic_l164.all; - SS SS SS | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALL; BB F.35.1 abs8.vhd - TYPE psclatharlows SS Library ieee; - TYPE psclatharlows SS SS use ieee.std_logic_l164.all; - SS SS SS
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd use ieee.std_logic_l164.ALL; BB F.35.1 abs8.vhd - TYPE psclatharlows SS Library ieee; - TYPE psclatharlows SS SS use ieee.std_logic_l164.all; - SS SS SS | |
| F.35 ABS8 library see; B F.35.1 abs8.vhd z architecture structure10f abs8 architecture g
 | R.35 ABS8 library work; library work; B R.35.1 abs8.vhd - ree.std_logic_l164.AL; B R.35.1 abs8.vhd - ree.std_logic_l164.AL; B Ibrary ieee; - ree.std_logic_l164.AL; S S - TYPE DECLARATIONS - S S
 | F.35 ABS8 library ieee; F.35.1 abs8.vhd - rese.std_logic_l164.ALL; BE P.35.1 abs8.vhd - rese.std_logic_l164.ALL; SS SS Ibrary ieee; - rese.std_logic_l164.ALL; SS SS SS Ibrary ieee; - reservation fabs8 is - SS SS SS Ibrary ieee; - - reservation fabs8 is - SS
 | F.35 ABS8
F.35.1 abs8.vhd - Ilbrary work:
use ieee.std_logic_1164.ALL;
architecture structure1 of abs8 is
- TYPE DECLARATIONS
- SIGNAL DECLARATIONS
 | F.35 ABS8 Iibrary work; F.35.1 abs8.vhd Iibrary work; F.35.1 abs8.vhd Ecc.std_logic_l164.ALL; Iibrary work; Statuctures structures of abs8 is Iibrary structures of abs8 is Statustrues Iibrary structures structures of abs8 is Statustrues

 | F.35 ABS8
F.35.1 abs8.vhd | F.35 ABS8
F.35.1 abs8.vhd
 | F.35 ABS8
F.35.1 abs8.vhd | F.35 ABS8
F.35.1 abs8.vhd | |
| F.35 ABS8 library ieee; B F.35.1 abs8.whd compared to the structure of abs8 abs8.whd compared to the structure of abs8 abs8.wh 1 intervice intervice intervice intervice abs8.wh
 | F.35 ABS8 Iibrary work:
Ibrary work:
use std_lot_l164.ALL;
architecture structurel of abs8 is 88 F.35.1 abs8.whd - TYPE DECLARATIONS 85 Iibrary ieee; - TYPE DECLARATIONS 85
 | F.35 ABS8 library work;
1.1brary work;
wee ieee.std_logic_1164.ALL;
architecture structurel of abs8 is
TYEE DECLARATIONS STORM, DECLARATIONS STORM STORM, DECLARATIONS STORM STO
 | F.35 ABS8 Iibrary work;
Isbrary work;
F.35.1 abs8.vhd TYE Bee.std.logic_1164.ALL;
architecture structurel of abs8 is TYE BECLANTIONS STORM, DECLANTIONS
 | F.35 ABS8 Library ieee;
11brary work;
F.35.1 abs8.vhd Tyre Bee.std_logic_1164.ALL;
architecture structurel of abs8 is
TYRE BECLANTIONS STORM, DECLANTIONS
 | F.35 ABS8 Library work;
11brary work;
F.35.1 abs8.vhd TYEE BEE-std_logic_1164.ALL;
architecture structurel of abs8 is
TYEE BELARATIONS STORM, DECLARATIONS STORM, DECLARATIONS
 | F.35 ABS8 Library work;
11brary work;
F.35.1 abs8.vhd TYEE BEE-std_logic_1164.ALL;
architecture structurel of abs8 is
TYEE BELARATIONS STORM, DECLARATIONS STORM, DECLARATIONS | F.35 ABS8 Library work;
11brary work;
F.35.1 abs8.vhd TYEE BEE-std_logic_1164.ALL;
architecture structurel of abs8 is
TYEE BELARATIONS STORM, DECLARATIONS STORM, DECLARATIONS
 | F.35 ABS8 Library work;
11brary work;
F.35.1 abs8.vhd TYEE BEE-std_logic_1164.ALL;
architecture structurel of abs8 is
TYEE BELARATIONS STORM, DECLARATIONS STORM, DECLARATIONS | |
| F.35 ABS8 library ieee; library work; B F.35.1 abs8.vhd - reference structurel of abs8 is G Ibbrary ieee; TYPE DECLARATIONS S
 | F.35 ABS8 Iibrary ieee; F.35.1 abs8.vhd acchitecture structurel of abs8 is 1ibrary ieee;
 | F.35 ABS8 Iibrary ieee; Iibrary work; BB F.35.1 abs8.vhd - - - Iibrary ieee; - - -
 | F.35 ABS8 library ieee; library work; BE BE <td>F.35 ABS8 Iibrary ieee; Iibrary ieee; B F.35.1 abs8.vhd architecture structurel of abs8 is A Iibrary ieee; TYPE DECLARATIONS S</td> <td>F.35 ABS8 library ieee; library work; library work; B F.35.1 abs8.vhd restructures of abs8 is control of abs8 is control of abs8 is control of abs8 is 11brary ieee; TYPE DECLARATIONS TYPE DECLARATIONS control of abs8 is control of abs8 is</td> <td>F.35 ABS8 library ieee; library work; library work; BE F.35.1 abs8.vhd - acclitecture structure! of abs8 is ecclitecture structure! ecclitecture is ecclitecture ecclitecture</td> <td>F.35 ABS8 library ieee; library work; library work; BE F.35.1 abs8.vhd - acclitecture structure! of abs8 is ecclitecture structure! ecclitecture is ecclitecture ecclitecture</td> <td>F.35 ABS8 library ieee; library work; library work; BE F.35.1 abs8.vhd - acclitecture structure! of abs8 is ecclitecture structure! ecclitecture is ecclitecture ecclitecture</td>
 | F.35 ABS8 Iibrary ieee; Iibrary ieee; B F.35.1 abs8.vhd architecture structurel of abs8 is A Iibrary ieee; TYPE DECLARATIONS S
 | F.35 ABS8 library ieee; library work; library work; B F.35.1 abs8.vhd restructures of abs8 is control of abs8 is control of abs8 is control of abs8 is 11brary ieee; TYPE DECLARATIONS TYPE DECLARATIONS control of abs8 is control of abs8 is
 | F.35 ABS8 library ieee; library work; library work; BE F.35.1 abs8.vhd - acclitecture structure! of abs8 is ecclitecture structure! ecclitecture is ecclitecture | F.35 ABS8 library ieee; library work; library work; BE F.35.1 abs8.vhd - acclitecture structure! of abs8 is ecclitecture structure! ecclitecture is ecclitecture | F.35 ABS8 library ieee; library work; library work; BE F.35.1 abs8.vhd - acclitecture structure! of abs8 is ecclitecture structure! ecclitecture is ecclitecture |
 |
| F.35 ABS8 library ieee; library work; B F.35.1 abs8.vhd - refere.stc_lofc_l164.ALL; B P.35.1 abs8.vhd - referestc_lofc_l164.ALL; S P.35.1 abs8.vhd - - TYPE DECLARATIONS S
 | F.35 ABS8 Iibrary ieee; F.35.1 abs8.vhd Zeritecture structured of abs8 is Type DECLANTIONS SS
 | F.35 ABS8 Iibrary ieee,
Iibrary work; Iibrary work; Best of abs1 Best of abs8 Best of abs8 </td <td>F.35 ABS8 Library ieee; Library work; Babse; Babse;</td> <td>F.35 ABS8 Library work;
Library work;
F.35.1 abs8.vhd - Error Logic_1164.ALL;
architecture studieter architecture structure of abs8 is
 TYPE DECLANATIONS TYPE DECLANATIONS</td> <td>F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor</td> <td>F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor</td> <td>F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor</td> <td>F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor</td>
 | F.35 ABS8 Library ieee; Library work; Babse;
 | F.35 ABS8 Library work;
Library work;
F.35.1 abs8.vhd - Error Logic_1164.ALL;
architecture studieter architecture structure of abs8 is
TYPE DECLANATIONS TYPE DECLANATIONS
 | F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor | F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor
 | F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor | F.35 ABS8 Iibrary ieee; Iibrary work; Babswork; Babswor | |
| F.35 ABS8 Fibrary ieee; F.35 ABS8 F.35.1 abs8.vhd reference std_logic_1164.ALL; are ieee.std_logic_1164.ALL; are ieee.std_logic_1164.ALL; are ieee.std_logic_1164.ALL; are ieee.std_logic_10.000 reference.std_logic_10.000 reference.std_logic_10.0000 reference.std_lo | F.35 ABS8 library ieee; F.35 ABS8 library work; use ieee.std_logic_l164.ALL; generation of abs8 is F.35.1 abs8.vhd reference of abs8 is

 | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
architecture structure of abs8 is
c-1 TYPE DECLANTIONS SS
 | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
architecture structure1 of abs8 is
TYPE perLANTIONS abs8 is
 | F.35 ABS8 library work:
11brary work:
use ieee.std_logic_l164.ALL;
architecture studened of abs8 is
TYPE perchantrows
 | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
architecture structure1 of abs8 is
TYPE DECLANTIONS SS
 | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
architecture structure1 of abs8 is
TYPE DECLANTIONS SS | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
architecture structure1 of abs8 is
TYPE DECLANTIONS SS
 | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
architecture structure1 of abs8 is
TYPE DECLANTIONS SS | |
| F.35 ABS8 Fiber Start steel Fiber Start St | F.35 ABS8 library work;
use ieee.std_logic_l164.ALL;
architecture structure of abs8 is
architecture structures of abs8 is
architecture

 | F.35 ABS8 library work;
Use ieee.std_logic_l164.ALL;
architecture structure of abs8 is
architecture structures of abs8 is
architecture
 | F.35. ABS8 library work:
I.35.1 abs8.vhd reference structures structures structures of abs8 is
architecture structures of abs8 is
 | F.35 ABS8 Iibrary work;
Use feee.std_logic_1164.ALL;
F.35.1 abs8.vhd
 | F.35 ABS8 library work:
1.35.1 abs8.vhd - Alls architecture structure of abs8 is architecture structures of abs8 is architecture structure of abs8 is architecture structure structures of abs8 is architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures architectures | F.35 ABS8 library work:
1.35.1 abs8.vhd - Alls architecture structure of abs8 is architecture structures of abs8 is architecture structure of abs8 is architecture structure structures of abs8 is architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures architectures
 | F.35 ABS8 library work:
1.35.1 abs8.vhd - Alls architecture structure of abs8 is architecture structures of abs8 is architecture structure of abs8 is architecture structure structures of abs8 is architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures architectures | F.35 ABS8 library work:
1.35.1 abs8.vhd - Alls architecture structure of abs8 is architecture structures of abs8 is architecture structure of abs8 is architecture structure structures of abs8 is architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures of abs8 is architecture structure structures architecture structures architectures | |
| F.35 ABS8 [library ieee;
library work:
web ieee.stdl.logic_l164.ALb;
architecture_structure1 of abs8 is
 | F.35 ABS8 library work; Reserved library work; Use ieee.std_logic_l164.ALL; second condition F.35.1 abs8.vhd second condition
 | F.35 ABS8 library work;
11brary work;
use ieee.std_logic_1164.ALD;
architecture structure1 of abs8 is
 | F.35 ABS8 Eibrary work;
Iibrary work;
use ieee.std.logic_1164.ALL;
architecture structure1 of abs8 is

 | F.35 ABS8 Eibrary work;
F.35.1 abs8.vhd - Erese.std_logic_1164.ALL;
architecture_structure1 of abs8 is
 | F.35 ABS8 Eibrary work;
Iibrary work;
use ieee.std_logic_1164.ALL;
architecture structure1 of abs8 is | F.35 ABS8 Eibrary work;
Iibrary work;
use ieee.std_logic_1164.ALL;
architecture structure1 of abs8 is
 | F.35 ABS8 Eibrary work;
Iibrary work;
use ieee.std_logic_1164.ALL;
architecture structure1 of abs8 is | F.35 ABS8 Eibrary work;
Iibrary work;
use ieee.std_logic_1164.ALL;
architecture structure1 of abs8 is | |
| F.35 ABS8 [library veck;
1.1brary work;
F.35.1 abs8.vhd v [library work;
architecture i of abs8 is so
 | F.35 ABS8 11brary work:
use std_logic_1164.ALL;
architecture structure1 of abs8 is 88
 | F.35 ABS8 [library work;
see ieee.std_logic_1164.ALL;
r.35.1 abs8.vhd - architecture structurel of abs8 is
 | F.35 ABS8 1ibrary work; F.35.1 abs8.vhd 2

 | F.35 ABS8 library ieee; library ieee; B F.35.1 abs8.vhd - architecture structure! of abs8 is SS
 | F.35 ABS8 11brary work; F.35.1 abs8.vhd 2 | F.35 ABS8 1ibrary work; F.35.1 abs8.vhd *
 | F.35 ABS8 1ibrary work; F.35.1 abs8.vhd * | F.35 ABS8 1ibrary work; F.35.1 abs8.vhd * | |
| F.35 ABS8 [library ieee; B
library work;
F.35.1 abs8.vhd ~ Abs8.vhd ~ 8 architecture structured of abs8 is 8 8
 | F.35 ABS8 Fibrary work;
Ubrary work;
F.35.1 abs8.vhd
 | F.35 ABS8 [library work;
library work;
F.35.1 abs8.vhd ~ [storic_logic_l164.ALL;
architecture1 of abs8 is 85
 | F.35 ABS8 [library ieee;
library work;
F.35.1 abs8.vhd - storictione_lodli64.ALU;
architecture1 of abs8 is

 | F.35 ABS8 [library ieee, work,
use ieee. std. logic_1164. ALL;
architecture structured of abs8 is
 | F.35 ABS8 [library work;
library work;
F.35.1 abs8.vhd - structure1 of abs8 is 85
850 [library work] of abs8 is 850 [library w | F.35 ABS8 [library work;
library work;
F.35.1 abs8.vhd - structure1 of abs8 is 85
850 [library work] of abs8 is 850 [library w | F.35 ABS8 [library work;
library work;
F.35.1 abs8.vhd - structure1 of abs8 is 85
850 [library work] of abs8 is 850 [library w | F.35 ABS8 [library work;
library work;
F.35.1 abs8.vhd - structure1 of abs8 is 85
850 [library work] of abs8 is 850 [library w |
 |
| F.35 ABS8 F.35 ABS8 F.35.1 abs8.vhd C. Start abs | F.35 ABS8 [library work;
library work;
wse jeee.std_logic_l164.ALL;

 | F.35 ABS8 library work;
library work;
use ieee.std_logic_l164.ALL;
 | F.35 ABS8 library work;
I.15rary work;
use ieee.std_logic_1164.ALL;
 | F.35 ABS8 Iibrary work:
Isbrary work:
Use ieee.std_logic_l164.ALL;
 | F.35 ABS8 library ieee;
library work;
use ieee.std_logic_l164.ALL;
 | F.35 ABS8 library ieee;
library work;
use ieee.std_logic_l164.ALL; | F.35 ABS8 library ieee;
library work;
use ieee.std_logic_l164.ALL;
 | F.35 ABS8 library ieee;
library work;
use ieee.std_logic_l164.ALL; | |
| F.35 ABS8 Fibrary ieee; 11brary work; E.35 ABS8 Fibrary work; E.35 ABS8 State of the std_logic_l164.ALL; State of the std_logic_l164 | F.35 ABS8 [library work;
R.35, ahs8, whd

 | F.35 ABS8 library ieee;
1.1brary work;
wse ieee.std_logic_1164.ALb;
SS
 | F.35 ABS8 library verk:
I.1brary work:
wse ieee.std_logic_1164.ALb;
SS
 | F.35 ABS8 [library ieee;
library work;
use ieee.std_logic_l164.ALb;
SS
 | F.35 ABS8 library verk:
I.1brary work:
wse ieee.std_logic_1164.ALb;
 | F.35 ABS8 library verk:
I.1brary work:
wse ieee.std_logic_1164.ALb; | F.35 ABS8 library verk:
I.1brary work:
wse ieee.std_logic_1164.ALb;
 | F.35 ABS8 library verk:
I.1brary work:
wse ieee.std_logic_1164.ALb; | |
| F.35 ABS8 [library ieee; B
library work; owe why construction [library work; construction] [library wor | F.35 ABS8 [library ieee;
Ilibrary work;
wse ieee.std_logic_l164.ALL;
SS

 | F.35 ABS8 library ieee;
I library work;
Wase ieee.std.logic_l164.ALL;
SS
 | F.35 ABS8 [library ieee;
Ilibrary work;
wse ieee.std.logic_l164.ALL;
SS
 | F.35 ABS8 [library work;
Library work;
were ieee.std.logic_l164.ALL;
SG
 | F.35 ABS8 [library ieee;
Ilibrary work;
wse ieee.std.logic_l164.ALL;
S
 | F.35 ABS8 [library ieee;
Ilibrary work;
wse ieee.std.logic_l164.ALL;
S | F.35 ABS8 [library ieee;
Ilibrary work;
wse ieee.std.logic_l164.ALL;
S
 | F.35 ABS8 [library ieee;
Ilibrary work;
wse ieee.std.logic_l164.ALL;
S | |
| F.35 ABS8 [library work]
library wo | F.35 ABS8 [library ieee] [library work] [library wo

 | F.35 ABS8 Fibrary work:
Itibrary work:
The foot of f
 | F.35 ABS8 Fibrary work:
 | F.35 ABS8 [library work]
Ibrary work] [starty work] [star
 | F.35 ABS8 Library work:
Library work: | F.35 ABS8 Library work:
Library work:
 | F.35 ABS8 Library work:
Library work: | F.35 ABS8 Library work:
Library work: |
 |
| F.35 ABS8 [library ieee;] library work: [library work:]]
 | F.35 ABS8 [library ieee/
library work;
 | F.35 ABS8 [library ieee]
 | F.35 ABS8 [library ieee, work:

 | F.35 ABS8 library ieee;
 | F.35 ABS8 [library ieee;] [library work;] [library work | F.35 ABS8 [library ieee;] [library work;] [library work | F.35 ABS8 [library ieee;] [library work;] [library work | F.35 ABS8 [library ieee;] [library work;] [library work |
 |
| F.35 ABS8 library ieee; B
 | F.35 ABS8 library ieee; B
 | F.35 ABS8 library ieee; B
 | F.35 ABS8 library ieee; B

 | F.35 ABS8 library ieee; B
 | F.35 ABS8 library ieee; B | F.35 ABS8 library ieee; B
 | F.35 ABS8 library ieee; B | F.35 ABS8 library ieee; B | |
| F.35 ABS8 [library ieee;] library work:
 | F.35 ABS8 [] library ieee; []] library work:
 | F.35 ABS8 [library ieee;] library work:
 | F.35 ABS8

 | F.35 ABS8
 | F.35 ABS8 | F.35 ABS8
 | F.35 ABS8 | F.35 ABS8 | |
| F.35 ABS8 library ieee, E
 | F.35 ABS8 [library ieee, P
 | F.35 ABS8 [library ieee,]
 | F.35 ABS8 library ieee; E

 | F.35 ABS8 [library ieee,]
 | F.35 ABS8 [library ieee,] | F.35 ABS8 [library ieee,]
 | F.35 ABS8 [library ieee,] | F.35 ABS8 [library ieee,] | |
| R 35 ARS8 library ieee
 | F 35 ARS8 library ieee/
 | R 35 ARC8 library ieee; D
 | F 35 ARS8 library ieee; D

 | R 35 ARS8 library ieee; E
 | F 35 ARCS library ieee; D | F 35 ARCS library ieee; D
 | F 35 ARCS library ieee; D | F 35 ARCS library ieee; D | |
| D 2 A BCO
 | T T ADCO
 | D 25 A DCO
 | T ABCO ABCO

 | C 1 jhrarv jees
 | E 26 ABCO Dibbary jees: | E 26 ABCO Dibbary jees:
 | E 26 ABCO Dibbary jees: | E 26 ABCO Dibbary jees: | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |
|
 |
 |
 |

 |
 | |
 | | | |

NEG4			274
component nor2p component xnor2p component xnor2 component xnor2 component xor2p component xor2p component xnor2 component xno	 VHDL object: Architecture "structure" (schemetic "/home/rjh6v/csis/mentor/B4/eddm/work/leg4/neg_gen_low_s") of entity "neg4" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/neg4") Generated by: rjh6v Generated by: rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Generated by: rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Frogram: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Store from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Store from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/neg_gen_low_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work Source from: /home/rjh6v/csis/mentor/B4/eddm/s Source from/s Source from/s Source from/s Source fro	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure2;	
F.36 NEG4 F.36.1 neg4.vhd Iibrary ieee; use iee.sci_logic_1164.all; use iees.sci_logic_1164.all; port(X: in INTEGER range -7 to +7; port(X: in INTEGER range -7 to +7); end NEG4; end NEG4; architecture BEHAVE of NEG4 is architecture BEHAVE of NEG4 is	begin pecess (x) pecess (x) begin a 2 = -x; end BEHAVE; F.36.2 neg4_vectors.txt * x 2 : time ; 0000 0000 : 500 ns; 1011 1001 : 500 ns; 1010 0110 : 500 ns; 1110 0100 : 500 ns; 1110 010 0 000 000 000 000 000 000 000	<pre> VHDL object: Entity 'neg4" (component interface 'home/rjh6v/csis/mentor/B4/eddm/work/ neg4.neg4') Ganerated on: Wed May 13 15:04:38 1998 Ganerated by: rjh6v/csis/mentor/B4/eddm/work/neg4/part Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg4/part Program: VHDLWITLE V8.5_2.4 Wed Feb 5 16:56:22 PST 1997 ELTBARY ieee: USE ieee std_logic_1164.ALL; entity neg4 is PORT LIST port(x : in std_logic_vector(3 downto 0); x : in std_logic_vector(3 downto 0); x : out std_logic_vector(3 downto 0); y; end neg4;</pre>	 WHDL Object: Architecture 'structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ neg4/neg_low_s') of entity "neg4" (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/ neg4.neg10w_s') of entity "neg4" (component interface '/home/rjh6v/csis/mentor/B4/eddm/work/ enerated on: Wed May 13 15:04:38 1998 Generated on: Wed May 13 15:04:38 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/ enerated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/ Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program: /HDWrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1

F.37 NEG8	architecture structure1 of neg8 is
F.37.1 neg8.vhd	
library ieee; use ieee.std_logic_l164.all;	component or 2 component xor2 component xor2
entity MEC8 is port(X: in INTEGER range -127 to +127; Z: out INTEGER range -127 to +127);	Degin Dongtows Construes COMPONENT SIGNAL ASSIGNMENTS COMPONENT INSTANTATIONS
end NBG8; architecture BEHAVE of NBC8 is	end structure1;
begin process (X)	F.37.4 neg8_structure2.vhd
Degin 2 <= -X; 2 <= -X; end process; end BEHAVE;	
F.37.2 neg8_vectors.txt	Generated on: Fri Jul 24 13:35:16 1998 - Generated by: rjh6v/csis/mentor/B4/eddm/work/neg8/neg8_xi72a_s Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg8/neg8_xi72a_s
<pre>% X Z : time ; 00000000 00000000 : 500 ns; 0111111 10000001 : 500 ns;</pre>	Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997
10000001 01111111 : 500 ms; 10000010 01111110 : 500 ms; 10001000 01111100 : 500 ms;	use iee.std_logic_l164.ALL; use iee.std_logic_l164.ALL; architecture structure2 of meg8 is SIGNAL DECLARATIONS
10010000 01110000 : 500 ms; 10100000 01100000 : 500 ms; 11000000 01000000 : 500 ms;	COMPONENT_DECLARATIONS component and2 component buf
F.37.3 neg8_structure1.vhd	Component Lity component Lity InLINE CONFIGURATIONS
neg8:neg8') Generated on: Wed May 13 16:50:45 1998 Generated by: rjh6v Source from: //nome/rjh6v/csis/mentor/B4/eddm/work/neg8/part	end structure2;
Program: VHULWTICE V8.5_2.4 Wed FED 5 16:30:22 FST 1997	
LTBRARY ieee, USE ieee.std_logic_l164.ALL; entityrneg8 is PORT LIST	
<pre>port(</pre>	
VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ ned8/ned8 den low.s") ofentity"ned8" (componentinterface"/home/rjh6v/csis/mentor/B4/eddm/	
work/neg8:neg8*) Generated on: Wed May 13 16:50:45 1998 Generated by: rjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/neg8/neg8_gen_low_s Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997	
library ieee; library work; use ieee.std_logic_1164.ALL;	. 275

S	
2	
Ξ	5
Ξ	5
2	ć
	_
2	2
r.,	
-	-

AITTHMETIC_Process: process(A, B, CIN, OP) Variable Operand1, operand2: std.logic_vector(4 downto 0); variable a_ext, istd.logic_vector(4 downto 0); variable not_a_ext, not_b_ext : std_logic_vector(4 downto 0); variable arry_ext : std_logic_vector(1 downto 0); variable logic_out : std_logic_vector(3 downto 0); variable arith_out : std_logic_vector(3 downto 0); library IEEE, ARTHMETIC; use IEEE.std_logic_l164.all; use ARTHMETIC std_logic_arith.all; use ARTHMETIC std_logic_arith.all; -use ARTHMETIC std_logic_arith.all; - Parameterized Generator Specification to VHDL Code - LogicLib generator called: ARITHMETIC - Parameters are: - type = ALU2901 w = 4 - carryin = YES - carryin = YES -- zero extend inputs to include carry bit a_ext := '0' & A; b_ext := '0' & B; b_ext := '0' & B; not_a_ext := '0' & not B; not_b_ext := '0' & CIN; carry_ext := '0' & CIN; OP: in std_logic_vector(2 downto 0); A: in std_logic_vector(3 downto 0); B: in std_logic_vector(3 downto 0); CIN: in std_logic; COUT: out std_logic; D); out std_logic_vector(3 downto 0)); when '101' => '...' and B; logic_out := (not A) and B; when '110' => A xor B; when '111' => when others => '.'); when others => '.'); when '011' =>
logic_out := A or B;
when '100' =>
when '100' =>
when '101' =>
when '101' => -- alu4wc Architecture Description architecture behave of alu4wc is begin ł -- alu4wc Entity Description entity alu4wc is -- Logical Functions F.38.1 alu4wc.vhd case OP is -- ALU2901 end case; end alu4wc begin port

-- assign output if (OP(2) = '1' or (OP(1) = '1' and OP(0) = '1')) then D <= logic_out; COTT <= 'X'; else D <= arith_out(3 downto 0); COTT <= arith_out(4); end if; arith_out := operand1 + operand2 + carry_ext; operand1 := (OTHERS => 'X');
operand2 := (OTHERS => 'X'); end process ARITHMETIC_Process; end behave; 500 ns; 500 ns; 500 ns; -- Arithmetic operations when "000" => : su su : su % OP A B CIN COUT D : time; 000 0000 0000 0 0 0000 : 500 ns; 000 1111 1111 1 1 1111 : 500 ns; 000 0101 0101 0 0 1010 : 500 ns; 000 1010 1010 1 1 0101 : 500 ns; 001 1111 0000 0 0 0000 : 500 ms; 001 0000 0000 0 0 1111 : 500 ms; 001 1010 0111 0 0 1010 : 500 ms; 001 1010 1101 1 0101 : 500 ms; susu :su su su su ł -----500 I 500 I 500 I : 500 : 500 : 500 011 0000 0000 0 X 0000 : 500 011 0000 1111 0 X 1111 : 500 011 1111 0000 0 X 1111 : 500 -- Arithmetic Functions F.38.2 alu4wc_vectors.txt
 101
 1111
 1111
 1111
 101

 101
 0000
 000
 000
 101
 101

 101
 0100
 101
 0101
 101
 101

 101
 0101
 1
 101
 101
 101

 %

 010
 0000
 1111
 0
 0000
 1

 010
 0000
 0000
 0
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 case OP is end case;

110 0000 0000 0 X 0000 : 500 ms; 110 0000 1111 0 X 1111 : 500 ms; 110 1111 0000 X 1111 : 500 ms; 110 1111 1111 X X 0000 : 500 ms;	CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure1; F.38.4 alu4wc_structure2.vhd
111 0000 0000 X X 1111 : 500 ns; 111 0000 1111 X X 0000 : 500 ns; 111 1111 0000 X X 0000 : 500 ns; 111 1111 X X 1111 : 500 ns;	A VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ alu4wc/alu4wc_high_s")offantity"alu4wc"(componentinterface"/home/rjh6v/csis/mentor/B4/eddm/
F.38.3 alu4wc_structure1.vhd	work/aludwc:aludwc') - Generated on: Tue Jun 16 11:18:16 1998 Generated by: rjh6v Source from: /home/rjh6v/css/mentor/84/eddm/work/aludwc/aludwc/high_s
VHDL object: Entity "aludwo" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work/ aludwo:aludwo")	
Generated on: Thu Jun 11 16:01:32 1998 Generated by: rjh6v Source from: /howe/rjh6v/csis/mentor/B4/eddm/work/alu4wc/part	LIDEARY WORK; use ieee.std_logic_l164.ALL; architecture structure2 of alu4wc is TYPE DECLARATIONS
LIBRARY STATEMENT	SIGNAL DECLARATIONS COMPONENT DECLARATIONS
LIBRAKT IEEEF FRCKAGE STATEMENT Eieee.std logic 1164.ALL;	component and component and component and
entity alu4wc is PORT LIST	component inv component invr
<pre>port(</pre>	component nand2p component nand3p
a : in std_logic_vector(3 downto 0); b : in std_logic_vector(3 downto 0); cir : in std_logic_vector(3 downto 0);	component nor2p commonent nor2p
cin : in starturgic; cout : out stallogic; d : out stallogic domin ()	component or 2 component xror2 component xror2
	INLINE CONFIGURATIONS
end alu4wc;	
VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ aludwc/aludwcgen.http:s")ofentity"aludwc"(componentinterface"/home/rjh6v/csis/mentor/B4/ 	end structure2; F.38.5 alu4wc_structure3.vhd
eculiwork.stutwec.stutwec / Generated On: Tute Jun 16 14:55:51 1998 Concerted Mu: viher.	
Generated by: /home/rih6v/csis/mentor/B4/eddm/work/alu4wc/alu4wc_gen_high_s Source from: /home/rihe v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Program:	VHDLobject: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ alu4wc/sdlu4wc.rj2a_gen_s")offntity"alu4wc"(componentinterface"/home/rjh6v/csis/mentor/B4/ eddm/work/alu4wc:alu4wc:)
library ieee;	Generated on: Tue Jun 23 15:41:03 1998 Generated by: rjh6v
use ieee.std_logic_l164.ALL;	Source from: /howe/rjh6v/csis/mentor/B4/eddm/work/alu4wc/alu4wc_xi72a_gen_s Program: VHDLwrite v8.5_2.4 Med Feb 5 16:56:22 PST 1997
architescute structured of aluque is Type Declarations	library ieee;
SIGMAL DECLARATIONS COMPONENT_DECLARATIONS COMPONENT_DAMPONENT	library work; use ieee.std_logic_1164.ALL;
component and3	architecture structure3 of alu4wc is TYPE DECLARATIONS
component inv component mand2 commonant ward3	SIGNAL DECLARATIONS COMPONENT_DECLARATIONS
component mande	component and2 component inv
component nors component nor3 commonent or2	component or2 INLINE CONFIGURATIONS
component or4 component xor2	CONCURRENT SIGNAL ASSIGNMENTS
- INLINE CONFIGURATIONS begin	end structure3;

-- assign output if (PP(2) = '1' or (OP(1) = '1' and OP(0) = '1')) then D <= 10gic_out; COUT <= 'X'; arith_out := operand1 + operand2 + carry_ext. : 500 ns; : 500 ns; : 500 ns; ; su ; su ; su ns; ns; ns; ns; su su su su : su su su case OF is - Arithmetic operations when '000' => ext; operand1 == a_ext; when '001' => b_ext; when '010' => b_ext; when '010' => b_ext; when '010' => cext; operand1 == a_ext; operand1 == a_ext; operand1 == a_ext; operand1 == (OTHERS => 'X'); end case; - 200 : 500 : 500 : 500 500 logic_out := not (A xor B); when others => logic_out := (OTHERS => 'X'); 500 else estimation of the second se 011 0000000 0000000 0 × 0000000 011 00000000 1111111 0 × 1111111 011 1111111 0 0000000 0 × 11111111 100 0000000 1111111 0 X 0000000 100 111111 000000 X 0000000 100 0101010 10101010 0 X 0101010 100 0101010 01010101 0 X 10101010 100 10101010 01010101 1 X 10101010 end process ARITHMETIC_Process ł -- Arithmetic Functions F.39.2 alu8wc_vectors.txt end case; end behave;

 Written by LL_to_VHBL at Mon Jum 8 12:23:29 1998
 Parameterized Generator Specification to VHDL Code
 Logictib generator called: ARITHMETIC
 Passed Parameters are:

 Linst name = alu4wc
 tinst nameters are:
 trinst nameters are:
 library IEEE, ARITHMETIC; use IEEE.std_logic_1164.all; use ARITHMETIC.std_logic_arith.all;

F.39.1 alu8wc.vhd

ALU8wc

F.39

-- alu4wc Entity Description entity alu8wc is port(

ARITHMETIC_Process: process(A,B,CIN,OP) variable operand1, operand2: std_logic_vector(8 downto 0); variable a_ext, bext: std_logic_vector(8 downto 0); variable not_a_ext, not_b_ext: std_logic_vector(8 downto 0); variable carry_ext: std_logic_vector(1 downto 0); variable logic_out: std_logic_vector(1 downto 0); variable arith_out: std_logic_vector(7 downto 0); variable arith_out: std_logic_vector(8 downto 0); ____zero extend inputs to include carry bit a_ext := '0' & A; b_ext := '0' & B; not_a_ext := '0' & not A; not_b_ext := '0' & not B; carry_ext := '0' & CIN; OF: in std_logic_vector(2 downto 0); A: in std_logic_vector(7 downto 0); B: in std_logic_vector(7 downto 0); COUT: in std_logic; COUT: out std_logic; COUT: out std_logic; D: out std_logic_vector(7 downto 0) minimize a or B; when '100' => and B; logic_out := A and B; when '101' => and B; when '110' => (not A) and B; when '110' => wor B; when '111' => -- alu8wc Architecture Description architecture behave of alu8wc is begin ł -- Logical Functions case OP is when "011" => -- ALU2901 end alu8wc; begin :

							279	
	INLINE CONFIGURATIONS begin CONCURRENT SIGNAL ASSIGNMENTS CONFONENT INSTANTIATIONS end structurel; F.394 alu8wc structure2.vhd	 VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ VHDL object: Architecture "structure" (schematic "/home/rjh6v/csis/mentor/B4/eddm/work/ alu8wc/alu8wc.xi72a_s') of entity alu8wc "(component interface "/home/rjh6v/csis/mentor/B4/ eddm/work/alu8wc:alu8wc:) eddm/work/alu8wc.alu8wc: 12.06:27 1998 Generated by: Tjh6v Source from: /home/rjh6v/csis/mentor/B4/eddm/work/alu8wc/alu8wc_xi72a_s Prooram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 Iibrary ieee; 	<pre>library vork; use ieee.std_logic_l164.ALL; use ieee.std_logic_latAArTONS TYER DECLANATIONS SIGNAL DECLANATIONS COMPONENT_DECLANATIONS COMPONENT_DECLANATIONS COMPONENT_AND' component inv</pre>	INLINE CONFIGURATIONS INLINE CONFIGURATIONS CONCURRENT SIGNAL ASSIGNMENTS COMPONENT INSTANTIATIONS end structure2;			219	
:	8 101 1111111 1111111 0 X 00000000 : 500 ms; 101 00000000 00000000 X 000000000 : 500 ms; 101 10101010 01010101 0 X 01010101 : 500 ms; 101 01010101 10101010 1 X 10101010 : 500 ms; 110 00000000 00000000 0 X 00000000 : 500 ms;	110 0000000 11111111 0 x 1111111 : 500 ms; 110 1111111 1111111 x x 0000000 : 500 ms; 111 0000000 0000000 x x 1111111 : 500 ms; 111 10000000 01111111 x x 0000000 : 500 ms; 111 11111111 00000000 x x 0000000 : 500 ms; 111 11111111 1111111 x x 1111111 : 500 ms; 111 11111111 1111111 x x 1111111 : 500 ms; 111 11111111 1111111 x x 1111111 : 500 ms;	 VHDL object: Bntity "alu8wc" (component interface "/home/rjh6v/csis/mentor/B4/eddm/work alu8wc") Generated on: Tue sep 8 11:23:40 1998 Generated by: rjh6v/csis/mentor/B4/eddm/work/alu8wc/part Source from: VHDLwrite v8.5_2.4 Wed Peb 5 16:56:22 PST 1997 	LIBRARY STATEMENT LIBRARY dee: PACRAGE STATEMENT USE iere-stallogic_1164.ALL; entity alu&uc is PORT LIST	<pre>port(a : in std_logic_vector(7 downto 0); b : in std_logic_vector(7 downto 0); cin : in std_logic_vector(2 downto 0); op : in std_logic_vector(2 downto 0); cout : out std_logic_vector(7 downto 0) d : out std_logic_vector(7 downto 0) ; end alu8wc;</pre>	 VHDLObject: Architecture 'structure' (schematic '/home/rjh6v/csis/mentor/B4/eddm/wor) alu8wc.gen_hidh_s')ofentity'alu8wc'(componentinterface'/home/rjh6v/csis/mentor/B4 eddm/work/alu8wc:alu8wc') edenzrated on: Tue Sep 8 11:23:40 1998 enerzated by: rjh6v/csis/mentor/B4 Source from: /home/rjh6v/csis/mentor/B4 Program: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 	<pre>library ieee; library vork; use ieee.std_logic_l164.ML; use ieee.std_logic_l164.ML; architecture fructurel of alu8wc is TYPE DECLARATIONS COMPONENT_DECLARATIONS COMPONENT_DECLARATIONS COMPONENT and2 component and3 component inv component nor2 component vor2 component vor2</pre>	

;

F.40 HAMMING4		
<pre>K401 hmmlingk/hd interfirmed: interfirme: int</pre>	<pre>eb 5 16:56:22 PST 1997 </pre>	HAMMING4

HAMMING4

.

component xnor2 component xnor3 -- INLINE CONFIGURATIONS begin -- CONTURENT SIGNAL ASSIGNMENTS -- COMPONENT INSTANTIATIONS end structure2;

F.40.5 hamming4_structure3.vhd

. .

-- VHDL Object: Architecture structure' (schematic 'home/rjh6v/csis/mentor/B4/eddm/work/ hamming4/hamming4.z's')ofentity'hamming4"(componentinterface'home/rjh6v/csis/mentor/ B4/eddm/work/hamming4:hamming4:) - Generated on: The Jun 23 12:09:05 1998 - Generated on: The Jun 23 12:09:05 1998 - Generated on: The Jun 23 12:09:05 1998 - Generated by: Tjh6v - Brource from: /home/rjh6v/csis/mentor/B4/eddm/work/hamming4/hamming4_xi72a_s - Brource from: /home/rjh6v/csis/mentor/B4/eddm/work/hamming4_xi72a_s - Propram: VHDLwrite v8.5_2.4 Wed Feb 5 16:56:22 PST 1997 - Program: VHDLwrite v8.5_2.4 Med Feb 5 16:56:22 PST 1997 - Program: Declanations - TyPE DECLANATIONS - Component and2 - Component inv - Component inv - CONFORMATIONS - COMPONENT INSTANTIONS - COMPONENT INSTANTIONS - CONFORMENT INSTANTIATIONS - CONFORMENT

HAMMING8	5	26	282
•	tdm/wor	a6v/csi w_s w_s	
	c / B4 / eð	ome/rji Gen_lo .gen_lo	
	/mentol	/mento ace -/ha mming8_ /	
	v/csis 198/par 51 1997	interfé interfé ng8/han ST 1997	
	 ne/rjh6 6:22 PS 	onent onent (/hammi: 6:22 P.	
	ce /hourk 5 16:5	1. (countries (countries)) (countries) (co	
	nterfa 8 8 /B4/edd 1 Feb	chemat anming 8 8 / B4/ed d Feb 	
	55 199 55 199 .4 We .4 We .4 Ve 	a tity "h tity" ag8") 55 199 //mentor 2.4 We 2.4 We a s a s b is	ENTS
ns; ns; ns; ns; ns; ns; ns; ns; ns; ns;	8" (com 10:35: v/csis/ v8:5.2 v8:5.2 continue	*struct) of en) of en) of en is hammin v Csis vV.csis vV.csis vV.csis is 10:35 vv.sis is 10:35 vv.sis vv.sis vv.sis sv.csis sv.sv.c	ASSIGNMATIONS
 5000 <li< td=""><td>aumuing v Jun 26 v vrite write sarite</td><td>ecture low_s muming8 o'un 26 o'un 26 b'un 26 herrite bwrite stors Rariows Rariows</td><td>IGNAL / STANTI/</td></li<>	aumuing v Jun 26 v vrite write sarite	ecture low_s muming8 o'un 26 o'un 26 b'un 26 herrite bwrite stors Rariows Rariows	IGNAL / STANTI/
0101101 110010101 110010101 0100100 010010	anticyte anticy	Archit gg8_gen, work/h. work/h. hog /hog /hog /hog /hog /hog /hog /hog	RENT S VENT IN rel;
0 1001 0 1010 0 1010 0 1010 0 1000 0 1010 0 1011 0 1011 0 1011 0 1111 0	increase interview intervi	/harmin /harmin /harmin ated by ated by ated by ated by ated by iter iter iter iter aten iter iter aten iter iter iter iter iter iter iter iter	CONCUT
	- vublication of the second of	- VHDL.o amming8 entor/B entor/B entor/B entor/B entor/B entor/B ibrary ibrary ibrary ibrary ibrary ibrary - Try coorce c	begir end s
¥		;;,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	

F.41 HAMMING8

F.41.1 hamming8.vhd

library IEEE; use IEEE.std_logic_1164.all;

); end hamming8;

architecture behave of hamming8 is begin

process(X,P)
variable S : std_logic_vector(1 to 4);
begin

						•••	•••						
••	••			-	-	-	-		-	^		-	
-	2				2	a	2		-	-		<u>-</u>	
-	2			- 2	2	÷	2	•••	C	÷	••	-	
ρ,	р,			Ū	0	S	S	^	S	S	^	s	
								-	~	~	-	~	
<u> </u>	£				ς.	Υ.	*	Ξ	片	片	Σ	片	
S.	S.			- 2	4	ä	H	ñ	2	2	õ	Z	
	.,	••	•••	•		Ξ.	-						
^	~	1	-	- 6	2	<u>e</u>	6	ň	-	-	6	-	
5	5	<u></u>	4	- 6	5	З	З.	g	<u> </u>	Ξ.	Э.	2	
¥	ž	Ā	Ā		4	~	~	н	ō	ū	~	ō	
		_	-		~	~	~	Δ			~		
ц,	ĸ	2	~		,	m	m	z	يد	÷	ŝ	e	
0	2	0	Ω.		-	~	~	4	2	2	~	3	
×	×	×	~	6	2	U1	U1	_	F	Ħ	01	4	
~	_	~	~		J	۵	۵	m	۵	Δ	υ,	_	
ŝ	6	80	80	ĩ	5	Ξ	Ξ	-	Ξ	Ξ	ō	N	
-	-	-	-	\$	1	4	4	S	~	4	F	~	
×	×	×	×					_			~	S	
~	~	~	~	E F	1	-	3	片	2	-	¥.	11	
5	r.	3	5	- 2	2	2	2	2	2	5	2	5	
×	×	×	×		٦	ŝ	S		ŝ	ŝ		ĕ	
•••	•••			-	-		_	-		_	-	-	
-	-	-	-		4	ñ	8	2	Ä	8	2	8	
Ξ	Ξ	Σ	5	- 2	2	2	9	20	2	2	70	2	
ž	2	50	÷.		-	H	~	•••	-	~	•••	-	
-				6	2	Ω	~	ρ	ρ	~	Р	~	
ĸ	Pi	Ř	ц,	- 2	4	z	-	z	z	ч	z	-	
<u>0</u>	<u>o</u>	2	Ω.		c	4	Ξ.	4	4	~	4	~	
24	25	24	~			_	•1	_	_	•1	_		
~	_	_	~		7	긑	ц	H	H	ų,	e i	ų,	
N	e	m	۰.		÷	-	ō	-	-	ō	-	ō	
-	-	~	~	۲	Q	S	p	S	S	c	s	R	
×	×	×	×		-	-	-	-	-	~	-	~	
~	~	e.	~		2	e.	e.	R.	e.	e.	e.	æ	
5	5	5	ö.	ā	5	5	5	Б	5	Б	Б	5	
×	×	×	×	- 5	ē.	×	×	×	×	×	×	×	
-	-	-	-		7	-	-	-	1	10	2	-	
- 2	Ξ	2	5	- 5	2	2	2	2	-	2	c	2	
	×	×	×		d	×	×	×	×	×	×	×	
			•••		1								
н	B	п	i(ļ	II.	H	1	Ŀ	J.	11	1	ll.	
••	••	••	••	,	۷	v	v	v	۷	v	v	v	
_	_	_	~		_	~	~	-	-	~	~	_	
e e	N	m	4		4	N	m	4	ŝ	v	5	œ	
-	-	-	÷		÷	÷	÷	÷	-	-	-	~	
S	S	S	S	- 6	٦	ρ	ρ	P	р	р	ρ	Р	

end process;

end behave;

F.41.2 hamming8_vectors.txt

	ime;	:su 009	:su 00	:00 ns;	(Su 00)	isu 009	:su 009	;su 00;	:su 009	:su 009	:su 009		:su 009	:00 ns;	:su 009	:su 009	:su 009	500 ns;					
																	•••						
,	0	00000000	011111110	01110000	10011101	0110110	11110000	00100100	00110111	01011010	11011010		11010010	10011010	11011011	11011010	01011010	11011100	11011000	11101010	11001010	11011010	00100101
	ρ.	0000	0000	00100	1100	1101	1110	0011	0011	0000	0011		0100	0100	1111	1100	0100	0100	0001	0100	0010	1011	1000
	8 X	00000000	01101111	01110010	10011101	11101110	11110010	00100101	001101100	01011011	01011011	æ	1101010	10011010	11011010	11011011	01011010	11011000	11011100	11001010	11101010	11011010	00100100

F.41.4 hamming8_structure2.vhd

-- VHDL object: Architecture Structure' [schematic 'home/rjh6v/csis/mentor/bareddm/work/ harming8/harming8:harming8:(componentinterface'home/rjh6v/csis/mentor/ BA/eddm/work/harming8:harming8:(componentinterface'home/rjh6v/csis/mentor/ BA/eddm/work/harming8:harming8: - Generated on: Fri Jun 26 10:59:01 1998 - Generated on: Fri Jun 26 10:59:01 1998 - Generated on: Fri Jun 26 10:59:01 1998 - Generated by: Tjh6v - Source from: home/rjh6v/csis/mentor/84/eddm/work/harming8/harming8.xi72a_s - Program: VHDLwrite v9.5_2.4 Wed Feb 5 16:56:22 FST 1997 - Frogram: VHDLwrite v9.5_2.4 Wed Feb 5 16:56:22 FST 1997 - Frogram: VHDLwrite v9.5_2.4 Wed Feb 5 16:56:22 FST 1997 - Frogram: Congramer vock; interry work; use ieee.std_logic_1164.ALD; architecture structure2 of harming8 is - TYPE DECLARATIONS - CONFORMENTONS - Frogram: Component and2 - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS - INLINE CONFIGURATIONS - COMPONENT INSTANTIONS - CONFORMENT SIGNAL ASSIGNMENTS - CONFORMENT INSTANTIONS - CONFORMENT INSTANTIANS - CONFORMENT INSTANTIANS - CONFORMENT INSTANTIANS - CONFORMENT INSTANTIANS - CONFORMENT INSTANTIANS

HAMMING8