WRDC-TR-90-5026

(2)

AD-A231 276

VHSIC HARDWARE DESCRIPTION LANGUAGE (VHDL) BENCHMARK SUITE

CAPT KAREN M. SERAFINO
DESIGN BRANCH
MICROELECTRONICS DIVISION

MICHAEL ALAN DUKES, M.S.E.E.
CAPTAIN, U.S. ARMY
AIR FORCE INSTITUTE OF TECHNOLOGY

October 1990

FINAL REPORT FOR PERIOD NOV 1989 TO OCT 1990

Approved for public release; distribution unlimited.

ELECTRONIC TECHNOLOGY LABORATORY
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

(3)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-5026	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION WRDC/ELED 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION WRDC/ELED
6c ADDRESS (City, State, and ZIP Code) WRDC/ELED Wright-Patterson AFB OH 45433-6543	7b ADDRESS (City, State, and ZIP Code) WRDC/ELED Wright-Patterson AFB OH 45433-6543
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER IN-HOUSE
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
62204F	6096	40	18

11 TITLE (Include Security Classification)

VHSIC HARDWARE DESCRIPTION LANGUGAGE (VHDL) BENCHMARK SUITE

12 PERSONAL AUTHOR(S)
Serafino, Karen Marie, Dukes, Michael Alan

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
FINAL	FROM NOV 89 TO OCT 90	1990 October	709

16 SUPPLEMENTARY NOTATION

The computer software contained herein are "harmless". Already in the public domain.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	05		VHDL - IEEE 1076 - DESIGN LANGUAGE-HARDWARE DESCRIPTION LANGUAGE.
12	05		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report documents changes and additions made to the VHDL Benchmark Suite released in October 1989 (WRDC-TR-89-5046). Each benchmark is designed to test one or more of a set of 71 VHDL language features in terms of the limitations of user's or vendor's system architecture, operating system, and VHDL toolset. These limitations could include CPU time and amount of memory required to simulate a test. Examples of language feature tests are the maximum number of signal declarations allowed in an architecture or the maximum size (number of characters) of a process label.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL CAPT KAREN M. SERAFINO	22b TELEPHONE (Include Area Code) 513-255-8635	22c OFFICE SYMBOL WRDC/ELED

DD Form 1473, JUN 86 *Previous editions are obsolete*

Table of Contents

I. Introduction

The VHDL Benchmark Suite is a set of VHDL models, each designed to test one or more specified features of the language in terms of system architecture/operating system/VHDL toolset limitations. These limitations include amount of memory and CPU time required for analyzing, model generating, building, and simulating a model. Most tests have one or more parameters that can be varied to observe their effect on timing and memory usage. Although the tests are referred to as a benchmark suite, they differ from the computer science definition of benchmarks in that they are designed to explore the robustness of a system's architecture, operating system, and VHDL environment, not to try to equalize these factors before comparing results between systems. They attempt to test limitations of VHDL features of interest to design engineers.

VHDL vendors each have their own set of validation tests to use on their product, but no standard suite exists for comparing capabilities and limitations between toolsets. This VHDL Benchmark Suite is an attempt to provide a standard set of tests and make it available to vendors and users. To date, the suite has 331 models, each tested on a VAX 8800 running the VMS operating system, a UNIX-based SUN 4, and the Intermetrics, Inc Standard VHDL 1076 Support Environment. This report covers the organization of the suite and how to use the benchmarks to test toolset/system limitations.

II. VHDL Benchmark Suite Organization

VHDL Benchmark Suite data resides in three directories, [.bench], [.coms], and [.readme] (VMS) or bench, coms, and readme (UNIX). Directory [.bench] contains the tests, [.coms] contains the Intermetrics, Inc (VMS and UNIX) command files, and [.readme] contains operating instructions and other descriptive documentation. Tests in the VHDL Benchmark Suite are classified and identified according to a "category matrix." There are 71 VHDL language categories, ranging from ALIASED SIGNALS to FOR-LOOPS to ACCESS OPERATIONS. The categories are the row headings of the matrix. The tests are numbered and the numbers are used as column headings in the matrix. A row/column cell contains an "X" if the benchmark corresponding to that column's test number uses the category corresponding to that row's language category. A listing of the matrix is included in this report as Appendix A, and in [.readme]matrices.edt (VMS) or readme/matrices.edt (UNIX) in the VHDL Benchmark Suite documentation.

Each language category represents a subdirectory name, and each test is placed in the appropriate subdirectory according to which language feature(s) it uses. All source code is under directory BENCH. For example, if a benchmark uses CONCURRENT (subdirectory a) ASSERT (subdirectory g2) statements in an ARCHITECTURE (subdirectory c), the test would reside in [.bench.a.c.g2] for VMS, or bench/a/c/g2 for UNIX.

In most cases, a [.bench] subdirectory will contain one or more test shells (".sh" filename extension). The test shell is not a VHDL source code file ready for analysis. It is input for a VHDL code generator named "gen.vhd" that resides in the directory [.bench]. This code generator is written in VHDL, and when simulated, takes the test shell and operator-input generic parameters, and produces syntactically correct VHDL source code. The generic parameters are what vary the resulting VHDL source file. For example, a generic parameter could specify how long a model is to be simulated, or how many signal assignment statements an architecture is to contain. The comments of each test shell include the author, the date of creation, an explanation of parameter meaning(s), and an example of generated code resulting from a simulation of gen.vhd with the test shell and specified inputs as generic parameters. There is also a version of the code generator for use with VHDL toolsets that do not allow generic parameters in a top-level entity : front_end.vhd and alternate_gen.vhd. A brief description of each test, along with its shell file appears in Appendix B. Test descriptions are also in file [.readme]test.edt (VMS) or readme/test.edt (UNIX) in the VHDL Benchmark Suite documentation. The VHDL source for the code generator appears in Appendix C and in directory [.bench] (VMS) or bench (UNIX).

The [.coms] directory has the same structure as [.bench]. For each subdirectory under [.bench] that has a test, there is the same subdirectory in [.coms] that has the VMS and UNIX command files to run that test. The command files to analyze, model generate, and build the code generator are under [.coms].

III. Corrections and Additions

This release of the **VHDL Benchmark** Suite includes corrected VHDL models and additional VHDL models. The corrections and additions made to the previous VHDL Benchmark Suite fell into one of several categories.

1. Corrections were made to the errant VHDL model.
2. Comments were included with the errant VHDL model stating why it was incorrect. The errant VHDL model was not removed. A corrected version of the VHDL model was added to the VHDL Benchmark Suite[1].
3. "Global" errors were corrected throughout all VHDL models[2].
4. VHDL models that include more VHDL language features were added to the VHDL Benchmark Suite.
5. "Realistic" VHDL models of simple hardware designs were added to the VHDL Benchmark Suite.

Some of the corrections made to VHDL models reflect recommendations made by the VASG. The large parity generator VHDL models, added as tests 328 through 331, were designed to be self-checking. Therefore, a dataflow-type Exclusive-OR equation is used to verify the correctness of the large collection of gates used to calculate the parity. This helps to ensure the correct operation of the target simulator when handling large structural descriptions.

[1] The errant VHDL model may be used to ensure that the target VHDL environment will flag the error indicated.
[2] e.g., `0ns` was changed to `0 ns`.

IV. Using the VHDL Benchmark Suite System

Two "help files" exist that should be read before running any of the benchmarks. They are [.readme]help.txt and [.readme]unixinfo.txt (VMS) or readme/help.txt and readme/unixinfo.tex (UNIX). These files are included in this report as Appendix D. Only UNIX users must read unix.info. It explains how to collect timing data using the UNIX "set time" command. File help.txt explains how to use the VHDL code generator, "gen.vhd," to generate VHDL source code files from test shells, then use the command files to analyze, model generate, build, and simulate the tests in batch (VMS) or background (UNIX) mode.

V. Recommendations

After running the benchmarks with varying parameters, users can get an idea of which VHDL language features are best supported/implemented by their system and VHDL toolset, and can compare their results with those obtained running the tests on different systems/VHDL toolsets.

Appendix A. *Category Matrix*

Author: Captain Michael A. Dukes
Date: 22 May 1989
Revisions:

	25May89	Capt Karen M. Serafino Added Block Asserts item G4
	29May89	Capt Michael A. Dukes Reconfigured For-loops (L) and While (O) to the category of loops (L). Under Loops exists "exits," "while," and "for." Added a category for Attributes, (O). Added another category for label size testing (T).
	31May89	Capt Michael A. Dukes Added a category (U) for consideration of the number of waveforms that may be in a signal assignment statement. Also added another category for consideration of inertial versus transport (V).
	6Jun89	Capt Karen M. Serafino Added Procedure Asserts and Function Asserts items G5 and G6.
	8Jun89	Capt Karen M. Serafino Added Operators "remainder," "absolute value," and "exponentiation," items P13, P14, and P15 respectively.
	14Feb90	Capt Karen M. Serafino Added another category for configuration specifications (W).

Benchmark Creation Environment:

Hardware Environment
- Processor - Digital Equipment VAX 8820
- Memory Size - 32M Bytes
- Disk Type - RA-81

Operating System - VAX/VMS 5.2

Vendor Toolset - Standard VHDL 1076 Support Environment
Intermetrics Inc., Bethesda, MD
Version 2.0 September 1, 1989

Account Parameters

CPU Limit	Infinit
Buffered I/O Byte Count Quota	7190
Timer Queue Entry Quota	50
Paging File Quota	123924
Default Page Fault Cluster	64
Enqueue Quota	200
Max Detached Processes	0

Direct I/O Limit	18
Buffered I/O Limit	18
Open File Quota	59
Subprocess Quota	6
AST Quota	78
Shared File Limit	0
Max Active Jobs	0

	\ Test CATEGORY	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
A	Concurrent	X	X	X	X			X			X	X		X	X	X		X	X	X	X			X	X
B	Sequential					X	X		X	X			X				X					X	X		
C	Architecture	X	X	X	X	X	X	X			X	X	X	X	X	X		X	X	X	X			X	X
D	Blocked			X	X																				
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics	X	X																						
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block				X																				
G5	Procedure																								
G6	Function									X															
H	Procedures																								
H1/R	Entities/Recursive																								
H2/R	Architecture/Recursive											X		X	X	R		X			X			X	
H3/R	Process/Recursive									X															
I	Functions																								
I1/R	Entities/Recursive	X																							
I2/R	Architecture/Recursive		X																X	X					X
I3/R	Process/Recursive									X															
J	Levels of Heirarchy																								
K	Arrayed					X	X		X																
L	Loops																								
L1	For					X						X													
L2	For exit																								
L3	While													X											
L4	While exit																								
M	If-then-else					X	X		X	X		X				X									
N	Case																								
O	Attributes																								
P	Operators																X	X	X						
P1	Addition																			X	X	X			
P2	Subtraction																						X	X	X
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals									X															
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write											X	X	X	X	X									
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms										X														
V	Inertial																								
W	Configuration Spec																								

	\ Test CATEGORY	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
A	Concurrent		X	X		X						X	X	X	X	X	X	X	X	X		X	X	X	X
B	Sequential	X			X		X	X	X	X	X										X				
C	Architecture		X	X	X	X	X					X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architectu e																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive																								
H2/R	Architecture/Recursive		X																				X		X
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive																								
I2/R	Architecture/Recursive			X																		X		X	
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed				X																				
L	Loops																								
L1	For				X																				
L2	For exit																								
L3	While																								
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition							X				X	X												
P2	Subtraction								X					X	X										
P3	Multiplication									X						X	X							X	X
P4	Division	X	X	X							X							X	X			X	X		
P5	Concatenation																								
P6	AND				X	X	X													X	X				
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec																								

	\ Test CATEGORY	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72
A	Concurrent	X	X	X	X			X	X				X	X				X	X				X	X	
B	Sequential					X	X			X	X	X			X	X	X			X	X	X			X
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive																								
H2/R	Architecture/Recursive	X			X																				
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive																								
I2/R	Architecture/Recursive		X	X																					
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed						X					X					X					X		
L	Loops																								
L1	For						X					X					X					X		
L2	For exit																								
L3	While																								
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition			X	X																				
P2	Subtraction	X	X																						
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR					X	X	X	X	X															
P8	NAND										X	X	X	X	X										
P9	NOR															X	X	X	X	X					
P10	XOR																				X	X	X	X	X
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec																								

	\ Test CATEGORY	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
A	Concurrent			X	X		X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
B	Sequential	X	X			X			X	X															
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/P	Entities/Recursive														X	X	X	R	X	X					X
H2/R	Architecture/Recursive										X	X													
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive																				X	X	X	X	
I2/R	Architecture/Recursive												X	X											
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed		X																						
L	Loops																								
L1	For		X												X										
L2	For exit																								
L3	While															X									
L4	While exit																								
M	If-then-else														X			X							
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																		X	X	X	X			
P2	Subtraction																						X	X	X
P3	Multiplication																								
P4	Division																								
P5	Concatenation						X	X	X	X	X	X	X	X											
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT	X	X	X	X	X																			
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write														X	X	X	X							
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec																								

	CATEGORY \ Test	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
A	Concurrent	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
B	Sequential																								
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Res Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive	X	X	X			X	X					X	X	X	X							X	X	
H2/R	Architecture/Recursive																X	X							X
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive				X	X			X	X	X	X							X	X					
I2/R	Architecture/Recursive																				X	X			
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed																								
L	Loops																								
L1	For																								
L2	For exit																								
L3	While																								
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																								
P2	Subtraction	X																							
P3	Multiplication		X	X	X	X																			
P4	Division						X	X	X	X															
P5	Concatenation										X	X	X	X											
P6	AND														X	X	X	X	X	X	X	X			
P7	OR																						X	X	X
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
A	Concurrent	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
B	Sequential																								
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
P2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive						X	X							X	X							X	X	
H2/R	Architecture/Recursive	X							X	X							X	X							X
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive		X	X							X	X							X	X					
I2/R	Architecture/Recursive				X	X							X	X							X	X			
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed																								
L	Loops																								
L1	For																								
L2	For exit																								
L3	While																								
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																								
P2	Subtraction																								
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR	X	X	X	X	X																			
P8	NAND						X	X	X	X	X	X	X	X											
P9	NOR														X	X	X	X	X	X	X	X			
P10	XOR																						X	X	X
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168
A	Concurrent	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
B	Sequential																								X
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive						X	X																	
H2/R	Architecture/Recursive	X							X	X															
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive		X	X							X	X													
I2/R	Architecture/Recursive				X	X							X	X											
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed																								X
L	Loops																								
L1	For																								X
L2	For exit																								
L3	While																								
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition														X										
P2	Subtraction															X									
P3	Multiplication																X								
P4	Division																	X							
P5	Concatenation																								
P6	AND																		X						X
P7	OR																			X					
P8	NAND																				X				
P9	NOR																					X			
P10	XOR	X	X	X	X	X																	X		
P11	NOT						X	X	X	X	X	X	X	X										X	
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
A	Concurrent																								
B	Sequential	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive																								
H2/R	Architecture/Recursive																								
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive																								
I2/R	Architecture/Recursive																								
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
L	Loops																								
L1	For	X	X	X	X	X													X	X	X	X	X	X	X
L2	For exit																								
L3	While						X	X	X	X	X	X	X	X	X	X	X	X							
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																		X				X		
P2	Subtraction																			X				X	
P3	Multiplication																				X				X
P4	Division																					X			
P5	Concatenation																								
P6	AND											X	X												
P7	OR	X									X			X											
P8	NAND		X							X					X									
P9	NOR			X					X							X								
P10	XOR				X			X									X							
P11	NOT					X	X											X						
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216
A	Concurrent																				X	X	X	X	X
B	Sequential	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	*Bus Resolved*																					•			
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive																				X				
H2/R	Architecture/Recursive																					X			
H3/R	Process/Recursive																						X		
I	Functions																								
I1/R	Entities/Recursive																							X	
I2/R	Architecture/Recursive																								X
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed	X	X	X	X	X	X	X	X	X															
L	Loops																								
L1	For	X																							
L2	For exit																								
L3	While		X	X	X	X	X	X	X	X															
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition		X				X																		
P2	Subtraction			X				X																	
P3	Multiplication				X				X																
P4	Division	X				X				X															
P5	Concatenation																								
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture														X										
T1B	Block															X									
T1C	Port																X								
T2	Variable																								
T2A	Process																	X							
T3	Constant											X													
T4	Types												X												
T5	Subtypes													X											
T6	Component Instant.																		X						
T7	Entity Label																			X					
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label										X														
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
A	Concurrent	X							X	X							X	X	X	X	X	X	X	X	X
B	Sequential		X	X	X	X	X	X			X	X	X	X	X	X									
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports			X	X	X	X																		
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive										R						X	X							
H2/R	Architecture/Recursive											R							X	X					
H3/R	Process/Recursive													R											
I	Functions																								
I1/R	Entities/Recursive									R					R							X	X		
I2/R	Architecture/Recursive								R							R								X	X
I3/R	Process/Recursive	X											R												
J	Levels of Heirarchy																								
K	Arrayed			X	X	X	X																		
L	Loops																								
L1	For			X																					
L2	For exit						X																		
L3	While				X																				
L4	While exit					X																			
M	If-then-else							X																	
N	Case		X																						
O	Attributes																								
P	Operators																								
P1	Addition																								
P2	Subtraction																								
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR			X	X	X	X																		
P11	NOT																								
P12	Modulo																X	X	X	X	X	X	X	X	X
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264
A	Concurrent	X							X	X	X	X	X	X	X	X	X	X							X
B	Sequential		X	X	X	X	X	X											X	X	X	X	X	X	
C	*Architecture*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	*Enumerated Types* (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive								X	X															X
H2/R	Architecture/Recursive										X	X													
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive												X	X											
I2/R	Architecture/Recursive														X	X									
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed		X	X	X	X													X	X	X	X			
L	Loops																								
L1	For		X	X															X	X					
L2	For exit																								
L3	While				X	X															X	X			
L4	While exit																								
M	*If-then-else*																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																								
P2	Subtraction																								
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo	X	X	X	X	X	X	X																	
P13	Remainder								X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
P14	Absolute Value																								X
P15	Exponentiation																								
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	*Inertial*																								
W	Configuration Spec.																								

	\ Test CATEGORY	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288
A	Concurrent	X	X	X	X	X	X	X	X	X													X	X	X
B	Sequential										X	X	X	X	X	X	X	X	X	X	X	X			
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	Entities																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive	X																					X	X	
H2/R	Architecture/Recursive		X	X																					X
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive				X	X																			
I2/R	Architecture/Recursive						X	X																	
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed										X	X	X	X	X	X	X	X							
L	Loops																								
L1	For										X	X	X	X											
L2	For exit																								
L3	While														X	X	X	X							
L4	While exit																								
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																								
P2	Subtraction																								
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
P15	Exponentiation																						X	X	X
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																								
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	\ Test CATEGORY	289	290	291	292	293	294	295	296	297	298	299	30	301	302	303	304	305	306	307	308	309	310	311	312
A	Concurrent	X	X	X	X	X	X	X																	X
B	Sequential								X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	A	X	X	X	X	X	X	X	X	X	X	X	X
D	Blocked																								
D1	Guards																								
D1A	Resolved																								X
D1B	Reg. Resolved																								
D1C	Bus Resolved																								
E	Enumerated Types (Large composite types)																								
F	Components																								
F1	w/ Ports																								
F2	w/o Ports																								
F3	w/ Generics																								
F4	w/o Generics																								
G	Asserts																								
G1	*Entities*																								
G2	Architecture																								
G3	Process																								
G4	Block																								
G5	Procedure																								
G6	Function																								
H	Procedures																								
H1/R	Entities/Recursive																								
H2/R	Architecture/Recursive	X																							
H3/R	Process/Recursive																								
I	Functions																								
I1/R	Entities/Recursive		X	X																					
I2/R	Architecture/Recursive				X	X																			
I3/R	Process/Recursive																								
J	Levels of Heirarchy																								
K	Arrayed								X	X	X	X	X	X	X	X									
L	Loops																								
L1	For								X	X	X	X									X				
L2	For exit																					X			
L3	While												X	X	X	X							X		
L4	While exit																							X	
M	If-then-else																								
N	Case																								
O	Attributes																								
P	Operators																								
P1	Addition																								
P2	Subtraction																								
P3	Multiplication																								
P4	Division																								
P5	Concatenation																								
P6	AND																								
P7	OR																								
P8	NAND																								
P9	NOR																								
P10	XOR																								
P11	NOT																								
P12	Modulo																								
P13	Remainder																								
P14	Absolute Value																								
P15	Exponentiation	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
Q	Aliased Signals																								
R	Access Operations (sec 3.3 & 7.3.6)																								
S	File I/O																								
S1	Read																				X	X	X	X	
S2	Write																								
T	Label Size																								
T1	Signal																								
T1A	Architecture																								
T1B	Block																								
T1C	Port																								
T2	Variable																								
T2A	Process																								
T3	Constant																								
T4	Types																								
T5	Subtypes																								
T6	Component Instant.																								
T7	Entity Label																								
T8	Architecture Label																								
T9	Block Label																								
T10	Process Label																								
U	Waveforms																								
V	Inertial																								
W	Configuration Spec.																								

	CATEGORY \ Test	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331
A	Concurrent	X	X	X	X				X	X	X	X					X	X	X	X
B	Sequential					X	X	X					X	X			X	X	X	X
C	Architecture	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X
D	Blocked																			
D1	Guards																			
D1A	Resolved														X					
D1B	Reg. Resolved		X																	
D1C	Bus Resolved	X																		
E	Enumerated Types (Large composite types)																			
F	Components														X					
F1	w/ Ports								X	X	X	X				X	X	X	X	X
F2	w/o Ports																			
F3	w/ Generics															X				
F4	w/o Generics																			
G	Asserts																			
G1	Entities																			
G2	Architecture			X																
G3	Process																			
G4	Block				X															
G5	Procedure						X													
G6	Function					X														
H	Procedures														X					
H1/R	Entities/Recursive						X													
H2/R	Architecture/Recursive																			
H3/R	Process/Recursive																			
I	Functions														X					
I1/R	Entities/Recursive																			
I2/R	Architecture/Recursive					X										X				
I3/R	Process/Recursive																			
J	Levels of Hairarchy									X						X				
K	Arrayed							X	X						X					
L	Loops																			
L1	For							X												
L2	For exit																			
L3	While																			
L4	While exit																			
M	If-then-else							X												
N	Case																			
O	Attributes							X												
P	Operators																			
P1	Addition																			
P2	Subtraction																			
P3	Multiplication																			
P4	Division																			
P5	Concatenation																			
P6	AND																			
P7	OR																			
P8	NAND																			
P9	NOR																			
P10	XOR					X	X													
P11	NOT																			
P12	Modulo																			
P13	Remainder																			
P14	Absolute Value																			
P15	Exponentiation																			
Q	Aliased Signals								X						X					
R	Access Operations (sec 3.3 & 7.3.6)																			
S	File I/O																			
S1	Read																			
S2	Write																			
T	Label Size																			
T1	Signal																			
T1A	Architecture																			
T1B	Block																			
T1C	Port																			
T2	Variable																			
T2A	Process																			
T3	Constant																			
T4	Types																			
T5	Subtypes																			
T6	Component Instant																			
T7	Entity Label																			
T8	Architecture Label																			
T9	Block Label																			
T10	Process Label																			
U	Waveforms																			
V	Inertial																			
W	Configuration Spec.																			

```
TEST NUMBER :  1

PATHNAME :  [.BENCH.A.C.F3.I1]shell0.sh (see readme.txt in this directory)
            (UNIX equivalent : bench/a/c/f3/i1/shell0.sh)

PURPOSE :  Determine the number of signals allowed in the generic clause of
           a component; determine the number of such components allowed in
           an architecture with a recursive function in its entity declaration.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Michael A. Dukes
--
--  DATE : 25 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signals (minus 1) per component
--        generic map
--        must match value of parameter number 1 in "shell1.sh"
--    2 : number of components
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test0.vhd",3,6
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test0.vhd\"\,3,6)
--      will generate a model in file "test0.vhd" with an architecture
--      in the form :
--        entity vbm0 is
--        end vbm0;
--        architecture vbm0 of vbm0 is
--            signal delay_time : integer := 1;
--            signal delay_time1 : integer := 1;
--            signal delay_time2 : integer := 1;
--            signal delay_time3 : integer := 1;
--            component vbm1
--              generic (delay  : integer
--                                ; delay1 : integer
--                                ; delay2 : integer
--                                ; delay3 : integer
--                       );
--              end component;
--            for all : vbm1 use entity work.vbm1(vbm1);
--            begin
--            comp1 : vbm1 generic map (delay_time
--                                      , delay_time1
--                                      , delay_time2
--                                      , delay_time3
```

```
--                                      );
--                          .
--                          .
--            comp6 : vbm1 generic map (delay_time
--                                                  , delay_time1
--                                                  , delay_time2
--                                                  , delay_time3
--                                      );
--            end vbm0;



entity vbm0 is
end vbm0;
architecture vbm0 of vbm0 is
    signal delay_time : integer := 1;
#1[    signal delay_time@ : integer := 1;]
    component vbm1
      generic (delay  : integer#1[; delay@ : integer]
              );
      end component;

    for all : vbm1 use entity work.vbm1(vbm1);

    begin
#2[    comp@ : vbm1 generic map (delay_time#1[, delay_time@]
                               );]
    end vbm0;


--  AUTHOR : Captain Michael A. Dukes
--
--  DATE : 25 May 1989
--
--  PARAMETER NUMBER MEANING
--    1 : number of signals (minus 1) in entity generic clause, number of
--        signal declarations/signal assignment statements in architecture
--        must match value of parameter number 1 in "shell0.sh"
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test1.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test1.vhd\"\,3)
--      will generate a model in file "test1.vhd" with an architecture
--      in the form :
--        entity vbm1 is
--          generic (delay  : integer
--                   ;delay1 : integer
--                   ;delay2 : integer
--                   ;delay3 : integer
--                  );
--                    .
```

```
--
--        architecture vbm1 of vbm1 is
--            signal go : bit;
--            signal clock1 : bit;
--            signal clock2 : bit;
--            signal clock3 : bit;
--            begin
--            go <= '0', '1' after delay_time(delay);
--            clock1 <= '1' nand go after delay_time(delay1);
--            clock2 <= '1' nand go after delay_time(delay2);
--            clock3 <= '1' nand go after delay_time(delay3);
--            end vbm1;


entity vbm1 is
  generic (delay  : integer
#1[           ;delay@ : integer]
         );
  function delay_time (in_delay : integer) return time is
    begin
    return(in_delay * 1 ns);
    end;
end vbm1;
architecture vbm1 of vbm1 is
    signal go : bit;
#1[    signal clock@ : bit;]
    begin
    go <= '0', '1' after delay_time(delay);
#1[    clock@ <= '1' nand go after delay_time(delay@);]
    end vbm1;
```

```
TEST NUMBER : 2

PATHNAME :  [.BENCH.A.C.F3.I2]shell0.sh (see readme.txt in this directory)
            (UNIX equivalent : bench/a/c/f3/i2/shell0.sh)

PURPOSE :  Determine the number of signals allowed in the generic clause of
           a component; determine the number of such components allowed in
           an architecture with a recursive function in its architecture body.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

--  AUTHOR : Captain Michael A. Dukes
--
--  DATE : 25 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signals (minus 1) per component
--        generic map
--        must match value of parameter number 1 in "shell1.sh"
--    2 : number of components
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test0.vhd",3,6
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test0.vhd\"\,3,6)
--      will generate a model in file "test0.vhd" with an architecture
--      in the form :
--        entity vbm0 is
--        end vbm0;
--        architecture vbm0 of vbm0 is
--            signal delay_time : integer := 1;
--            signal delay_time1 : integer := 1;
--            signal delay_time2 : integer := 1;
--            signal delay_time3 : integer := 1;
--            component vbm1
--              generic (delay  : integer
--                                ; delay1 : integer
--                                ; delay2 : integer
--                                ; delay3 : integer
--                       );
--              end component;
--            for all : vbm1 use entity work.vbm1(vbm1);
--            begin
--            comp1 : vbm1 generic map (delay_time
--                                          , delay_time1
--                                          , delay_time2
--                                          , delay_time3
--                                   );
--
```

```
--
--            comp6 : vbm1 generic map (delay_time
--                                                , delay_time1
--                                                , delay_time2
--                                                , delay_time3
--                                      );
--            end vbm0;



entity vbm0 is
end vbm0;
architecture vbm0 of vbm0 is
    signal delay_time : integer := 1;
#1[    signal delay_time@ : integer := 1;]
    component vbm1
      generic (delay  : integer#1[; delay@ : integer]
              );
      end component;

    for all : vbm1 use entity work.vbm1(vbm1);

    begin
#2[    comp@ : vbm1 generic map (delay_time#1[, delay_time@]
                               );]
    end vbm0;


--  AUTHOR : Captain Michael A. Dukes
--
--  DATE : 25 May 1989
--
--  PARAMETER NUMBER MEANING
--    1 : number of signals (minus 1) in entity generic clause, number of
--        signal declarations/signal assignment statements in architecture
--        must match value of parameter number 1 in "shell0.sh"
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test1.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test1.vhd\"\,3)
--      will generate a model in file "test1.vhd" with an architecture
--      in the form :
--        entity vbm1 is
--          generic (delay  : integer
--                   ;delay1 : integer
--                   ;delay2 : integer
--                   ;delay3 : integer
--                  );
--                    .
--                    .
--        architecture vbm1 of vbm1 is
```

```
--            function delay_time (in_delay : integer) return time is
--              begin
--              return(in_delay * 1 ns);
--              end;
--            signal go : bit;
--            signal clock1 : bit;
--            signal clock2 : bit;
--            signal clock3 : bit;
--            begin
--            go <= '0', '1' after delay_time(delay);
--            clock1 <= '1' nand go after delay_time(delay1);
--            clock2 <= '1' nand go after delay_time(delay2);
--            clock3 <= '1' nand go after delay_time(delay3);
--            end vbm1;



entity vbm1 is
  generic (delay  : integer
#1[           ;delay@ : integer]
         );
end vbm1;
architecture vbm1 of vbm1 is
    function delay_time (in_delay : integer) return time is
      begin
      return(in_delay * 1 ns);
      end;
    signal go : bit;
#1[   signal clock@ : bit;]
    begin
    go <= '0', '1' after delay_time(delay);
#1[   clock@ <= '1' nand go after delay_time(delay@);]
    end vbm1;
```

```
TEST NUMBER :  3

PATHNAME :  [.BENCH.A.C.D]shell.sh
            (UNIX equivalent : bench/a/c/d/shell.sh)

PURPOSE :  Determine the maximum number of blocks allowed in an architecture;
           determine the maximum number of signal declarations/signal
           assignment statements allowed per block; determine the CPU time
           required per block for analysis, model generation, build, and
           simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :  Seconds/block


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 26 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of blocks
--    2 : number of signal declarations/signal assignment statments per block
--    3 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",5,4,7
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,5,4,7)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity blocks is end;
--            .
--            .
--          go <= '0', '1' after 1 ns, '0' after 7 ns;
--          b1 : block
--            signal s1 : bit;
--            signal s2 : bit;
--            signal s3 : bit;
--            signal s4 : bit;
--          begin
--            s1 <= s1 nand go after 1 ns;
--            s2 <= s2 nand go after 1 ns;
--            s3 <= s3 nand go after 1 ns;
--            s4 <= s4 nand go after 1 ns;
--          end block b1;
--              .
--              .
--          b5 : block
--            signal s1 : bit;
--            signal s2 : bit;
--            signal s3 : bit;
```

```
--            signal s4 : bit;
--          begin
--            s1 <= s1 nand go after 1 ns;
--            s2 <= s2 nand go after 1 ns;
--            s3 <= s3 nand go after 1 ns;
--            s4 <= s4 nand go after 1 ns;
--          end block b5;
--        end blocks;



entity blocks is end;
architecture blocks of blocks is
  signal go : bit;
begin
  go <= '0', '1' after 1 ns, '0' after %3% ns;
#1[  b@ : block
#2[    signal s@ : bit;]
  begin
#2[    s@ <= s@ nand go after 1 ns;]
  end block b@;]
end blocks;


TEST NUMBER :  4

PATHNAME :  [.BENCH.A.C.D.G4]shell.sh
            (UNIX equivalent : bench/a/c/d/g4/shell.sh)

PURPOSE :  Determine the maximum number of blocks allowed in an architecture;
           determine the maximum number of signal declarations/signal
           assignment statements/assertions (one per signal) allowed per block;
           determine the CPU time required per block for analysis, model
           generation, build, and simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of blocks
--    2 : number of signal declarations/signal assignment statements/assertions
--        per block
--    3 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
```

```
--    $ sim gen/param="shell.sh","test.vhd",5,3,8
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,5,3,8)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity blocks is end;
--            .
--            .
--          go <= '0', '1' after 1 ns, '0' after 8 ns;
--          b1 : block
--            signal s1 : bit;
--            signal s2 : bit;
--            signal s3 : bit;
--          begin
--            s1 <= s1 nand go after 1 ns;
--            assert (s1= '0') or (s1= '1') severity error;
--            s2 <= s2 nand go after 1 ns;
--            assert (s2= '0') or (s2= '1') severity error;
--            s3 <= s3 nand go after 1 ns;
--            assert (s3= '0') or (s3= '1') severity error;
--          end block b1;
--                .
--                .
--          b5 : block
--            signal s1 : bit;
--            signal s2 : bit;
--            signal s3 : bit;
--          begin
--            s1 <= s1 nand go after 1 ns;
--            assert (s1= '0') or (s1= '1') severity error;
--            s2 <= s2 nand go after 1 ns;
--            assert (s2= '0') or (s2= '1') severity error;
--            s3 <= s3 nand go after 1 ns;
--            assert (s3= '0') or (s3= '1') severity error;
--          end block b5;
--        end blocks;


entity blocks is end;
architecture blocks of blocks is
  signal go : bit;
begin
  go <= '0', '1' after 1 ns, '0' after %3% ns;
#1[  b@ : block
#2[    signal s@ : bit;]
  begin
#2[    s@ <= s@ nand go after 1 ns;
    assert (s@= '0') or (s@= '1') severity error;]
  end block b@;]
end blocks;
```

```
TEST NUMBER :  5

PATHNAME :  [.BENCH.B.C.K.L1.M]shell.sh
            (UNIX equivalent : bench/b/c/k/l1/m/shell.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes and the same number of bit_vector signal declarations;
           each process consists of a variable declaration of a bit_vector the
           same size as the signals above, a signal assignment statement, and
           an if-then-else construct with for-loops in the 'if' and 'else'
           sections; the number of iterations of each for-loop is equal to the
           size of the bit_vectors, and a variable assignment is made at each
           iteration :  number of processes/number of bit_vector signal
           declarations, bit_vector size of the signals, and length of time
           (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Michael A. Dukes
--
--  DATE : 25 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal/process repetitions
--    2 : bit_vector size
--    3 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10,100,5
--   (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10,100,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--             .
--             .
--           signal big1 : bit_vector(100 downto 1);
--               .
--               .
--           signal big10 : bit_vector(100 downto 1);
--           begin
--           go <= '1' after 1 ns, '0' after 5 ns;
--           p1 : process(go,big1)
--             variable temp : bit_vector(big1'range);
--                 .
--                 .
--             big1 <= temp nand big1 after 1 ns;
--           end process p1;
```

```
--                  .
--                  .
--            p10 : process(go,big10)
--              variable temp : bit_vector(big10'range);
--                  .
--                  .
--              big10 <= temp nand big10 after 1 ns;
--            end process p10;
--          end test;


entity test is
end test;
architecture test of test is

    signal go : bit;
#1[    signal big@ : bit_vector(%2% downto 1);]
    begin
    go <= '1' after 1 ns, '0' after %3% ns;
#1[    p@ : process(go,big@)
      variable temp : bit_vector(big@'range);
      begin
      if go = '1' then
        for i in temp'range loop
          temp(i) := '1';
        end loop;
      else
        for i in temp'range loop
          temp(i) := '0';
        end loop;
      end if;
      big@ <= temp nand big@ after 1 ns;
    end process p@;]
    end test;
```

```
TEST NUMBER :  6

PATHNAME :  [.BENCH.B.C.K.M]shell.sh
            (UNIX equivalent : bench/b/c/k/m/shell.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes and the same number of bit_vector signal declarations;
           each process consists of a variable declaration of a bit_vector the
           same size as the signals above and an if-then-else construct with
           a signal assignment statement in the 'if' section and a for-loop in
           the 'else' section; the number of iterations of the for-loop is
           equal to the size of the bit_vectors, and a variable assignment is
           made at each iteration :  number of processes/number of bit_vector
           signal declarations, bit_vector size of the signals, and length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Michael A. Dukes
--
--  DATE : 26 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal/process repetitions
--    2 : bit_vector size
--    3 : length of time (ns) to simulate model (should be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10,100,5
--   (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10,100,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--                .
--                .
--            signal big1 : bit_vector(100 downto 1);
--                  .
--                  .
--            signal big10 : bit_vector(100 downto 1);
--            begin
--            go <= '1' after 1 ns;
--            stop <= '1' after 5 ns;
--            assert (stop = '0') severity error;
--            p1 : process(go,big1)
--              variable temp : bit_vector(big1'range);
--              begin
--              if go = '1' then
```

```
--                big1 <= temp nand big1 after 1 ns;
--                   .
--                   .
--             end process p1;
--                   .
--                   .
--             p10 : process(go,big10)
--               variable temp : bit_vector(big10'range);
--               begin
--               if go = '1' then
--                 big10 <= temp nand big10 after 1 ns;
--                   .
--                   .
--               end process p10;
--             end test;



entity test is end;
architecture test of test is
  signal go : bit;
  signal stop : bit;
#1[  signal big@ : bit_vector(%2% downto 1);]
    begin
    go <= '1' after 1 ns;
    stop <= '1' after %3% ns;
    assert (stop = '0') severity error;
#1[    p@ : process(go,big@)
      variable temp : bit_vector(big@'range);
      begin
      if go = '1' then
      big@ <= temp nand big@ after 1 ns;
      else
        for i in temp'range loop
          temp(i) := '1';
        end loop;
      end if;
    end process p@;]
end test;
```

```
TEST NUMBER :  7

PATHNAME :  [.BENCH.A.C]shell.sh
            (UNIX equivalent : bench/a/c/shell.sh)

PURPOSE :  Determine the maximum number of signal declarations/signal assignment
           statements allowed in an architecture;  determine the CPU time
           required for simulating the model, varying the time before reaching
           a quiescent state inside the description; determine the CPU time
           required per signal for analysis, model generation, build, and
           simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Michael A. Dukes
--
--  Date : 25 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal assignment statements
--    2 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",25,3
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,25,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--                  .
--                  .
--            signal clock1 : bit;
--                  .
--                  .
--            signal clock25 : bit;
--            begin
--            go <= '1' after 1 ns, '0' after 3 ns;
--            clock1 <= clock1 nand go after 1 ns;
--                  .
--                  .
--            clock25 <= clock25 nand go after 1 ns;
--            end testor;



entity test is
end test;
architecture testor of test is
```

```
    signal go : bit;
#1[    signal clock@ : bit;]

    begin

    go <= '1' after 1 ns, '0' after %2% ns;

#1[    clock@ <= clock@ nand go after 1 ns;]

    end testor;
```

```
TEST NUMBER :  8

PATHNAME :  [.BENCH.B.K.M]shell.sh
            (UNIX equivalent : bench/b/k/m/shell.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of one bit_vector
           signal declaration and a number of processes equal to the bit_vector
           size of the signal;  each process consists of an if-then-else
           construct, where the 'if' and 'else' sections each contain a signal
           assignment statement : bit_vector size/number of processes and length
           of time (in ns) to simulate model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Michael A. Dukes
--
--  Date : 31 May 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of processes/number of 'if' statements
--    2 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",15,3
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,15,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture testor of test is
--            signal go : bit;
--            signal clock : bit_vector(15 downto 1);
--            begin
--            go <= '1' after 1 ns, '0' after 3 ns;
--          pr1: process (go,clock(1))
--              constant zero : bit := '0';
--            begin
--            if(clock(1)=zero)then
--              clock(1) <= clock(1) nand go after 1 ns;
--            else
--              clock(1) <= clock(1) nand go after 1 ns;
--            end if;
--            end process pr1;
--                .
--                .
--          pr15: process (go,clock(15))
--              constant zero : bit := '0';
```

```
--            begin
--            if(clock(15)=zero)then
--              clock(15) <= clock(15) nand go after 1 ns;
--            else
--              clock(15) <= clock(15) nand go after 1 ns;
--            end if;
--            end process pr15;
--            end testor;



entity test is
end test;
architecture testor of test is

    signal go : bit;
    signal clock : bit_vector(%1% downto 1);

    begin

    go <= '1' after 1 ns, '0' after %2% ns;


#1[  pr@: process (go,clock(@))
      constant zero : bit := '0';
    begin
    if(clock(@)=zero)then
      clock(@) <= clock(@) nand go after 1 ns;
    else
      clock(@) <= clock(@) nand go after 1 ns;
    end if;
    end process pr@;]
    end testor;
```

TEST NUMBER : 9

PATHNAME : [.BENCH.B.G6.H3.I3.M.Q]shell.sh
(UNIX equivalent : bench/b/g6/h3/i3/m/q/shell.sh)

PURPOSE : Determine the effect on CPU time when analyzing, model generating, building, and simulating the following model : an architecture consisting of a number of unaliased signal declarations, a number of aliased signal declarations, and three types of processes. The first type of process has two signal assignment statements. The architecture contains one of this type, and a number of the other two types of processes. The second process type has three variable declarations, a procedure declaration, an if-then-else statement where the "if" and "else" sections each have two variable assignment statements, a procedure call, and an aliased signal assignment statement. The procedure consists of one variable declaration, a function declaration, a function call via a variable assignment statement, and an if-then-else statement, where the "if" and "else" sections each have a function call via a variable assignment statement. The function has one variable declaration, an if-then-else statement, where the "if" and "else" sections each have two variable assignment statements, a return statement, and an assert statement. The third process type has three variable declarations, a procedure declaration, an if-then-else statement, where the "if" and "else" sections each have a variable assignment statement, a procedure call, and an unaliased signal assignment statement. The procedure is identical to the one in the second type of process. The function is identical to the one in the second type of process, except it has one assert statement instead of two. The factors to be varied are the number of unaliased signal declarations/number of type two processes, the number of aliased signal declarations/number of type three processes, and the length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Michael Dukes
--
--  Date : 1 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/processes not using an aliased signal
--    2 : number of signal declarations/processes using an aliased signal
--    3 : length of time to simulate model (ns)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",11,8,5
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,11,8,5)
```

```
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        use work.mos_logic_package3.all;
--        entity nested is end;
--        architecture nested of nested is
--            signal go:mos_node;
--            alias g2 : mos_node is go;
--            signal s1 : mos_node;
--                    .
--                    .
--            signal s11 : mos_node;
--            signal sa1 : mos_node;
--                    .
--                    .
--            signal sa8 : mos_node;
--        begin
--            process
--                begin
--                go.L.S <= 'D' after 1 ns;
--                go.L.V <= '1'
--                      ,'0' after 1 ns
--                      ,'1' after 2 ns
--                      ,'0' after 3 ns
--                      ,'1' after 4 ns
--                      ,'0' after 5 ns
--                           ;
--                wait;
--                end process;
--
--          pra1 : process(g2)
--          variable v0,v1:integer:=0;
--          variable stmp : mos_node;
--          procedure f1(n1:integer;n4:out mos_node) is
--            variable n3:mos_node;
--            function f2(n2:integer) return mos_node is
--              variable temp : mos_node;
--            begin
--              if (n2 mod 3 = 0) then temp.L.S := 'D';temp.L.V := '0';
--              else temp.L.S := 'D';temp.L.V := '0';
--              end if;
--              return temp;
--              assert (n2 < 10000) severity error;
--              assert false severity error;
--            end f2;
--          begin
--            n3 := f2(v1);
--            if (n1 mod 2 = 0) then n4 := snot(snor(g2,n3));
--            else n4 := snot(snand(g2,n3));
--            end if;
--          end f1;
--          begin
```

```
--            if (g2.L.S = 'D') and (g2.L.V = '0') then v0:=v0+1;
--            else v1:=v1+1;
--            end if;
--            f1(v0,stmp);
--            sa1 <= stmp after 1 ns;
--            end process pra1 ;
--                  .
--                  .
--          pra8 : process(g2)
--          variable v0,v1:integer:=0;
--          variable stmp : mos_node;
--          procedure f1(n1:integer;n4:out mos_node) is
--            variable n3:mos_node;
--            function f2(n2:integer) return mos_node is
--              variable temp : mos_node;
--            begin
--              if (n2 mod 3 = 0) then temp.L.S := 'D';temp.L.V := '0';
--              else temp.L.S := 'D';temp.L.V := '0';
--              end if;
--              return temp;
--              assert (n2 < 10000) severity error;
--              assert false severity error;
--            end f2;
--          begin
--            n3 := f2(v1);
--            if (n1 mod 2 = 0) then n4 := snot(snor(g2,n3));
--            else n4 := snot(snand(g2,n3));
--            end if;
--          end f1;
--          begin
--            if (g2.L.S = 'D') and (g2.L.V = '0') then v0:=v0+1;
--            else v1:=v1+1;
--            end if;
--            f1(v0,stmp);
--            sa8 <= stmp after 1 ns;
--            end process pra8 ;
--                  .
--                  .
--          pr1 : process(go)
--          variable v0,v1:integer:=0;
--          variable stmp : mos_node;
--          procedure f1(n1:integer;n4:out mos_node) is
--            variable n3:mos_node;
--            function f2(n2:integer) return mos_node is
--              variable temp : mos_node;
--            begin
--              if (n2 mod 3 = 0) then temp.L.S := 'D';temp.L.V := '0';
--              else temp.L.S := 'D';temp.L.V := '0';
--              end if;
--              return temp;
--              assert false severity error;
```

```
--           end f2;
--         begin
--           n3 := f2(v1);
--           if (n1 mod 2 = 0) then n4 := snot(snor(go,n3));
--           else n4 := snot(snand(go,n3));
--           end if;
--         end f1;
--         begin
--           if (go.L.S = 'D') and (go.L.V = '0') then v0:=v0+1;
--           else v1:=v1+1;
--           end if;
--           f1(v0,stmp);
--           s1 <= stmp after 1 ns;
--           end process pr1 ;
--                .
--                .
--         pr11 : process(go)
--         variable v0,v1:integer:=0;
--         variable stmp : mos_node;
--         procedure f1(n1:integer;n4:out mos_node) is
--           variable n3:mos_node;
--           function f2(n2:integer) return mos_node is
--             variable temp : mos_node;
--           begin
--             if (n2 mod 3 = 0) then temp.L.S := 'D';temp.L.V := '0';
--             else temp.L.S := 'D';temp.L.V := '0';
--             end if;
--             return temp;
--             assert false severity error;
--           end f2;
--         begin
--           n3 := f2(v1);
--           if (n1 mod 2 = 0) then n4 := snot(snor(go,n3));
--           else n4 := snot(snand(go,n3));
--           end if;
--         end f1;
--         begin
--           if (go.L.S = 'D') and (go.L.V = '0') then v0:=v0+1;
--           else v1:=v1+1;
--           end if;
--           f1(v0,stmp);
--           s11 <= stmp after 1 ns;
--           end process pr11 ;
--       end nested;


use work.mos_logic_package3.all;
entity nested is end;
architecture nested of nested is
    signal go:mos_node;
```

```
    alias g2 : mos_node is go;
#1[    signal s@ : mos_node;]
#2[    signal sa@ : mos_node;]
begin
    process
        begin
        go.L.S <= 'D' after 1 ns;
        go.L.V <= '1'
#3[              ,'$2$0$1$' after @ ns]
                  ;
        wait;
        end process;

#2[  pra@ : process(g2)
  variable v0,v1:integer:=0;
  variable stmp : mos_node;
  procedure f1(n1:integer;n4:out mos_node) is
    variable n3:mos_node;
    function f2(n2:integer) return mos_node is
      variable temp : mos_node;
    begin
      if (n2 mod 3 = 0) then temp.L.S := 'D';temp.L.V := '0';
      else temp.L.S := 'D';temp.L.V := '0';
      end if;
      return temp;
      assert (n2 < 10000) severity error;
      assert false severity error;
    end f2;
  begin
    n3 := f2(v1);
    if (n1 mod 2 = 0) then n4 := snot(snor(g2,n3));
    else n4 := snot(snand(g2,n3));
    end if;
  end f1;
  begin
    if (g2.L.S = 'D') and (g2.L.V = '0') then v0:=v0+1;
    else v1:=v1+1;
    end if;
    f1(v0,stmp);
    sa@ <= stmp after 1 ns;
    end process pra@ ;]
#1[  pr@ : process(go)
  variable v0,v1:integer:=0;
  variable stmp : mos_node;
  procedure f1(n1:integer;n4:out mos_node) is
    variable n3:mos_node;
    function f2(n2:integer) return mos_node is
      variable temp : mos_node;
    begin
      if (n2 mod 3 = 0) then temp.L.S := 'D';temp.L.V := '0';
      else temp.L.S := 'D';temp.L.V := '0';
```

```
      end if;
      return temp;
      assert false severity error;
    end f2;
  begin
    n3 := f2(v1);
    if (n1 mod 2 = 0) then n4 := snot(snor(go,n3));
    else n4 := snot(snand(go,n3));
    end if;
  end f1;
  begin
    if (go.L.S = 'D') and (go.L.V = '0') then v0:=v0+1;
    else v1:=v1+1;
    end if;
    f1(v0,stmp);
    s@ <= stmp after 1 ns;
    end process pr@ ;]
end nested;
```

```
TEST NUMBER :  10

PATHNAME :  [.BENCH.A.C.U]shell.sh
            (UNIX equivalent : bench/a/c/u/shell.sh)

PURPOSE :  Determine the maximum number of values allowed in a waveform of a
           signal assignment statement; determine the CPU time required for
           analysis, model generation, build, and simulation of the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 2 June 1989
--
--  PARAMETER MEANING :
--    1 : length of time to simulate model/number (-1) of waveforms in signal
--        assignment statement
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit;
--        begin
--          sig <= '0'
--                ,'1' after 1 ns
--                ,'0' after 2 ns
--                .
--                .
--                ,'0' after 10 ns
--                   ;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit;
begin
  sig <= '0'
#1[        ,'$2$1$0$' after @ ns]
           ;
end test;
```

```
TEST NUMBER :  11

PATHNAME :  [.BENCH.A.C.H2.L1.M.S2]shell.sh
            (UNIX equivalent : bench/a/c/h2/l1/m/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required to write characters to
           an output file.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a for-loop containing a write statement
           (one character) and an if-then construct containing a write
           statement (one character).  The factor to be varied is the number
           of characters to write to the file during one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 2 June 1989
--
--  PARAMETER MEANING :
--    1 : number of characters to write to output file "data_file.dat";
--        after every 75 characters, a linefeed is written
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file(number_of_characters : in integer) is
--          begin
--            for i in 1 to number_of_characters loop
--              write(out_file,'*');
--              if (i mod 75) =  0 then
--                write(out_file,lf);
--              end if;
--            end loop;
--          end write_file;
--        begin
--          write_file(100);
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
--   ********************************************************************************
--   *************************
```

```
entity test is end;
architecture test of test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file(number_of_characters : in integer) is
  begin
    for i in 1 to number_of_characters loop
      write(out_file,'*');
      if (i mod 75) =  0 then
       · write(out_file,lf);
      end if;
    end loop;
  end write_file;
begin
  write_file(%1%);
end test;
```

```
TEST NUMBER :  12

PATHNAME :  [.BENCH.B.C.S2]shell.sh
            (UNIX equivalent : bench/b/c/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required for writing characters
           to an output file.  The model simulated is an architecture consisting
           of a process containing a number of write character/write linefeed
           combinations.  The factor to be varied is the number of write
           character/write linefeed pairs.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 5 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form (each write statement repeated 100 times) :
--        entity test is end;
--        architecture test of test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--        begin
--          process
--            begin
--              write(out_file,'*');
--              write(out_file,lf);
--              write(out_file,'*');
--              write(out_file,lf);
--                      .
--                      .
--              write(out_file,'*');
--              write(out_file,lf);
--              write(out_file,'*');
--              write(out_file,lf);
--            wait;
--         end process;
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
```

```
--      (100 lines)
--        *
--        *
--        .
--        .
--        *
--        *


entity test is end;
architecture test of test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
begin
  process
    begin
#1[      write(out_file,'*');
      write(out_file,lf);]
    wait;
    end process;
end test;
```

```
TEST NUMBER :  13

PATHNAME :  [.BENCH.A.C.H2.L3.S2]shell.sh
            (UNIX equivalent : bench/a/c/h2/l3/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required to write characters to
           an output file.  The model simulated is an architecture consisting of
           a procedure declaration and a procedure call.  The procedure consists
           of a while-loop construct, where a write character/write linefeed
           combination is executed at each iteration.  The factor to be varied
           is the number of iterations the while-loop will be executed.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 5 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file is
--            variable number_of_lines : integer := 100;
--          begin
--            while number_of_lines > 0 loop
--              write(out_file,'*');
--              write(out_file,lf);
--              number_of_lines := number_of_lines - 1;
--            end loop;
--          end write_file;
--        begin
--          write_file;
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
--      (100 lines)
--        *
--        *
```

```
--          .
--          .
--          *
--          *


entity test is end;
architecture test of test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file is
    variable number_of_lines : integer := %1%;
  begin
    while number_of_lines > 0 loop
      write(out_file,'*');
      write(out_file,lf);
      number_of_lines := number_of_lines - 1;
    end loop;
  end write_file;
begin
  write_file;
end test;
```

```
TEST NUMBER :  14

PATHNAME :  [.BENCH.A.C.H2.S2]shell.sh
            (UNIX equivalent : bench/a/c/h2/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required to write characters
           to an output file.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call.  The procedure
           consists of a number of write character/write linefeed combinations.
           The factor to be varied is the number of write character/write
           linefeed pairs in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 5 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE .
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file is
--          begin
--            write(out_file,'*');  (100
--            write(out_file,lf);     repetitions)
--                    .
--                    .
--            write(out_file,'*');
--            write(out_file,lf);
--          end write_file;
--        begin
--          write_file;
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
--      (100 lines)
--        *
--        *
```

```
--        .
--        .
--        *
--        *


entity test is end;
architecture test of test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file is
  begin
#1[    write(out_file,'*');
    write(out_file,lf);]
  end write_file;
begin
  write_file;
end test;
```

```
TEST NUMBER :  15

PATHNAME :  [.BENCH.A.C.H2R.M.S2]shell.sh
            (UNIX equivalent : bench/a/c/h2r/m/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required for writing characters
           to an output file.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call.  The procedure
           consists of a write character/write linefeed combination and a
           recursive call to itself.  The factor to be varied is the number of
           times the procedure will recursively call itself.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 5 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file(number_of_lines : in integer) is
--          begin
--            write(out_file,'*');
--            write(out_file,lf);
--            if number_of_lines > 1 then
--              write_file(number_of_lines - 1);
--            end if;
--          end write_file;
--        begin
--          write_file(20);
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
--      (20 lines)
--        *
--        *
--        .
```

```
--         .
--        *
--        *


entity test is end;
architecture test of test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file(number_of_lines : in integer) is
  begin
    write(out_file,'*');
    write(out_file,lf);
    if number_of_lines > 1 then
      write_file(number_of_lines - 1);
    end if;
  end write_file;
begin
  write_file(%i%);
end test;
```

```
TEST NUMBER :  16

PATHNAME :  [.BENCH.B.P1]shell0.sh
            (UNIX equivalent : bench/b/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition statements.  The model simulated is an architecture
           consisting of a process.  The process contains a number of
           integer variable declarations and an addition statement for each
           variable.  The factor to be varied is the number of variable
           declarations/addition statements in the process.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 5 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable declarations/variable addition statements in the
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var1 : integer := 0;
--                        .
--                        .
--            variable var20 : integer := 0;
--          begin
--            var1 := var1 + 1;
--                  .
--                  .
--            var20 := var20 + 1;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
begin
  process
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ + 1;]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  17

PATHNAME :  [.BENCH.A.C.H2.P1]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and an addition statement for each variable.  The factor to be varied
           is the number of variable declarations/addition statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino

--  Date : 6 June 1989

--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable addition statements in
--        procedure

--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure add is
--            variable var1 : integer := 0;
--                      .
--                      .
--            variable var20 : integer := 0;
--          begin
--            var1 := var1 +  1;
--                  .
--                  .
--            var20 := var20 +  1;
--          end add;
--        begin
--          add;
--        end test;



entity test is end;
```

```
architecture test of test is
  procedure add is
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ +  1;]
  end add;
begin
  add;
end test;
```

```
TEST NUMBER :  18

PATHNAME :  [.BENCH.A.C.I2.P1]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and an addition statement for each variable.  The factor to be varied
           is the number of variable declarations/addition statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 7 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable addition statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function add return boolean is
--            variable var1 : integer := 0;
--                      .
--                      .
--            variable var20 : integer := 0;
--          begin
--            var1 := var1 +  1;
--                  .
--                  .
--            var20 := var20 +  1;
--            return true;
--          end add;
--          signal done : boolean := false;
--        begin
--          done <= add;
--        end test;
```

```
entity test is end;
architecture test of test is
  function add return boolean is
#1[    variable var0 : integer := 0;]
  begin
#1[    var0 := var0 +  1;]
    return true;
  end add;
  signal done : boolean := false;
begin
  done <= add;
end test;
```

```
TEST NUMBER :  19

PATHNAME :  [.BENCH.A.C.I2.P2]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and a subtraction statement for each variable.  The factor to be
           varied is the number of variable declarations/subtraction statements
           in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 7 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable subtraction statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function subtract return boolean is
--            variable var1 : integer := 0;
--                          .
--                          .
--            variable var20 : integer := 0;
--          begin
--            var1 := var1 -  1;
--                    .
--                    .
--            var20 := var20 -  1;
--            return true;
--          end subtract;
--          signal done : boolean := false;
--        begin
--          done <= subtract;
--        end test;
```

```
entity test is end;
architecture test of test is
  function subtract return boolean is
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ -  1;]
    return true;
  end subtract;
  signal done : boolean := false;
begin
  done <= subtract;
end test;
```

```
TEST NUMBER :  20

PATHNAME :  [.BENCH.A.C.H2.P2]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and a subtraction statement for each variable.  The factor to be
           varied is the number of variable declarations/subtraction statements
           in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino

--  Date : 7 June 1989

--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable subtraction statements in
--        procedure

--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure subtract is
--            variable var1 : integer := 0;
--                        .
--                        .
--            variable var20 : integer := 0;
--          begin
--            var1 := var1 -  1;
--                    .
--                    .
--            var20 := var20 -  1;
--          end subtract;
--        begin
--          subtract;
--        end test;



entity test is end;
```

```
architecture test of test is
  procedure subtract is
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ -  1;]
  end subtract;
begin
  subtract;
end test;
```

```
TEST NUMBER :  21

PATHNAME :  [.BENCH.B.P2]shell0.sh
            (UNIX equivalent : bench/b/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction statements.  The model simulated is an architecture
           consisting of a process.  The process contains a number of
           integer variable declarations and a subtraction statement for each
           variable.  The factor to be varied is the number of variable
           declarations/subtraction statements in the process.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE . 7 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable declarations/variable subtraction statements in the
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var1 : integer := 0;
--                          .
--                          .
--            variable var20 : integer := 0;
--          begin
--            var1 := var1 - 1;
--                  .
--                  .
--            var20 := var20 - 1;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
begin
  process
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ - 1;]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  22

PATHNAME :  [.BENCH.B.P3]shell0.sh
            (UNIX equivalent : bench/b/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication statements.  The model simulated is an architecture
           consisting of a process.  The process contains a number of
           integer variable declarations and a multiplication statement for
           each variable.  The factor to be varied is the number of variable
           declarations/multiplication statements in the process.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 8 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable declarations/variable multiplication statements in
--        the process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var1 : integer := 5;
--                          .
--                          .
--            variable var20 : integer := 5;
--          begin
--            var1 := var1 * 2;
--                  .
--                  .
--            var20 := var20 * 2;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
begin
  process
#1[    variable var@ : integer := 5;]
  begin
#1[    var@ := var@ * 2;]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  23

PATHNAME :  [.BENCH.A.C.H2.P3]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and a multiplication statement for each variable.  The factor to be
           varied is the number of variable declarations/multiplication
           statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable multiplication statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure multiply is
--            variable var1 : integer := 5;
--                        .
--                        .
--            variable var20 : integer := 5;
--          begin
--            var1 := var1 *  2;
--                   .
--                   .
--            var20 := var20 *  2;
--          end multiply;
--        begin
--          multiply;
--        end test;



entity test is end;
```

```
architecture test of test is
  procedure multiply is
#1[    variable var@ : integer := 5;]
  begin
#1[    var@ := var@ *  2;]
  end multiply;
begin
  multiply;
end test;
```

```
TEST NUMBER :  24

PATHNAME :  [.BENCH.A.C.I2.P3]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and a multiplication statement for each variable.  The factor to be
           varied is the number of variable declarations/multiplication
           statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable multiplication statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function multiply return boolean is
--            variable var1 : integer := 5;
--                          .
--                          .
--            variable var20 : integer := 5;
--          begin
--            var1 := var1 *  2;
--                    .
--                    .
--            var20 := var20 *  2;
--            return true;
--          end multiply;
--          signal done : boolean := false;
--        begin
--          done <= multiply;
--        end test;
```

```
entity test is end;
architecture test of test is
  function multiply return boolean is
#1[    variable var@ : integer := 5;]
  begin
#1[    var@ := var@ *  2;]
    return true;
  end multiply;
  signal done : boolean := false;
begin
  done <= multiply;
end test;
```

```
TEST NUMBER :  25

PATHNAME :  [.BENCH.B.P4]shell0.sh
            (UNIX equivalent : bench/b/p4/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division statements.  The model simulated is an architecture
           consisting of a process.  The process contains a number of
           integer variable declarations and a division statement for
           each variable.  The factor to be varied is the number of variable
           declarations/division statements in the process.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 8 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable declarations/variable division statements in
--        the process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var1 : integer := 10;
--                          .
--                          .
--            variable var20 : integer := 10;
--          begin
--            var1 := var1 / 2;
--                  .
--                  .
--            var20 := var20 / 2;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
begin
  process
#1[    variable var0 : integer := 10;]
  begin
#1[    var0 := var0 / 2;]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  26

PATHNAME :  [.BENCH.A.C.H2.P4]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p4/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and a division statement for each variable.  The factor to be
           varied is the number of variable declarations/division
           statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable division statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure divide is
--            variable var1 : integer := 10;
--                        .
--                        .
--            variable var20 : integer := 10;
--          begin
--            var1 := var1 /  2;
--                  .
--                  .
--            var20 := var20 /  2;
--          end divide;
--        begin
--          divide;
--        end test;



entity test is end;
```

```
architecture test of test is
  procedure divide is
#1[    variable var@ : integer := 10;]
  begin
#1[    var@ := var@ /  2;]
  end divide;
begin
  divide;
end test;
```

```
TEST NUMBER :  27

PATHNAME :  [.BENCH.A.C.I2.P4]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p4/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and a division statement for each variable.  The factor to be
           varied is the number of variable declarations/division
           statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable division statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",20
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,20)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function divide return boolean is
--            variable var1 : integer := 10;
--            variable var20 : integer := 10;
--          begin
--            var1 := var1 /  2;
--            var20 := var20 /  2;
--            return true;
--          end divide;
--          signal done : boolean := false;
--        begin
--          done <= divide;
--        end test;


entity test is end;
architecture test of test is
  function divide return boolean is
#1[     variable var@ : integer := 10;]
```

```
  begin
#1[    var@ := var@ /  2;]
    return true;
  end divide;
  signal done : boolean := false;
begin
  done <= divide;
end test;
```

```
TEST NUMBER :  28

PATHNAME :  [.BENCH.B.C.K.L1.P6]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p6/shell0.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes; each process consists of a variable bit_vector
           declaration and a for-loop; the for-loop contains a logical AND
           statement, and the number of iterations of the loop is equal to
           the size of the variable bit_vector : number of processes, bit_vector
           size/number of iterations of for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of AND statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit_vector(1 to 10);
--          begin
--            for i in 1 to 10 loop
--              var(i) := var(i) AND var(i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit_vector(1 to 10);
--          begin
--            for i in 1 to 10 loop
--              var(i) := var(i) AND var(i);
--            end loop;
--            wait;
--          end process pr2;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit_vector(1 to %1%);
  begin
    for i in 1 to %1% loop
      var(i) := var(i) AND var(i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  29

PATHNAME :  [.BENCH.A.C.P6]shell0.sh
            (UNIX equivalent : bench/a/c/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical AND
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical AND signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of logical AND signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/AND statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : bit := '0';
--                      .
--                      .
--          signal sig10 : bit := '1';
--        begin
--          sig1 <= sig1 AND sig1;
--                      .
--                      .
--          sig10 <= sig10 AND sig10;
--        end test;



entity test is end;
architecture test of test is
#1[  signal sig@ : bit := '$2$0$1$';]
begin
#1[  sig@ <= sig@ AND sig@;]
end test;
```

```
TEST NUMBER :  30

PATHNAME :  [.BENCH.B.C.P6]shell0.sh
            (UNIX equivalent : bench/b/c/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing logical
           AND operations on variables.  The model simulated is an architecture
           consisting of a number of processes; each process consists of a
           number of variable declarations and one logical AND variable
           assignment statement for each variable.  The factors to be varied
           are the number of processes and the number of variable declarations/
           number of logical AND variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
-   Date : 8 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/AND statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : bit := '0';
--                        .
--                        .
--            variable var10 : bit := '1';
--          begin
--            var1 := var1 AND var1;
--                    .
--                    .
--            var10 := var10 AND var10;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : bit :=  '0';
--                        .
--                        .
--            variable var10 : bit := '1';
```

```
--          begin
--            var1 := var1 AND var1;
--                      .
--                      .
--            var10 := var10 AND var10;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin .
#2[  pr@ : process
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ AND var@;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  31

PATHNAME :  [.BENCH.B.P1]shell1.sh
            (UNIX equivalent : bench/b/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform addition
           operations on a variable; determine the number of addition
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a process.  The process
           consists of an integer variable declaration and a variable
           assignment statement containing a number of addition operations.
           The factor to be varied is the number of addition operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 9 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable additions in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var : integer := 0;
--          begin
--            var := var
--                      + var
--                      + var
--                      + var
--                      + var
--                      + var
--                      + var
--                      + var
--                      + var
--                      + var
--                      + var
--                    ;
--            wait;
--          end process;
```

```
--        end test;


entity test is end;
architecture test of test is
begin
  process
    variable var : integer := 0;
  begin
    var := var
#1[              + var]
          ;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  32

PATHNAME :  [.BENCH.B.P2]shell1.sh
            (UNIX equivalent : bench/b/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform subtraction
           operations on a variable; determine the number of subtraction
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a process.  The process
           consists of an integer variable declaration and a variable
           assignment statement containing a number of subtraction operations.
           The factor to be varied is the number of subtraction operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 9 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable subtractions in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var : integer := 0;
--          begin
--            var := var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var
--                   ;
--            wait;
--          end process;
```

```
--        end test;


entity test is end;
architecture test of test is
begin
  process
    variable var : integer := 0;
  begin
    var := var
#1[                - var]
           ;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  33

PATHNAME :  [.BENCH.B.P3]shell1.sh
            (UNIX equivalent : bench/b/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform multiplication
           operations on a variable; determine the number of multiplication
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a process.  The process
           consists of an integer variable declaration and a variable
           assignment statement containing a number of multiplication
           operations.  The factor to be varied is the number of
           multiplication operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 9 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable multiplications in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var : integer := 1;
--          begin
--            var := var
--                        * var
--                        * var
--                        * var
--                        * var
--                        * var
--                        * var
--                        * var
--                        * var
--                        * var
--                        * var
--                    ;
--            wait;
```

```
--          end process;
--        end test;



entity test is end;
architecture test of test is
begin
  process
    variable var : integer := 1;
  begin
    var := var
#1[                * var]
       .    ;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  34

PATHNAME :  [.BENCH.B.P4]shell1.sh
            (UNIX equivalent : bench/b/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform division
           operations on a variable; determine the number of division
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a process.  The process
           consists of an integer variable declaration and a variable
           assignment statement containing a number of division operations.
           The factor to be varied is the number of division operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 9 June 1989
--
--  PARAMETER MEANING :
--    1 : number of variable divisions in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable var : integer := 1;
--          begin
--            var := var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var
--                   ;
--            wait;
--          end process;
```

```
--        end test;


entity test is end;
architecture test of test is
begin
  process
    variable var : integer := 1;
  begin
    var := var
#1[                / var]
           ;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  35

PATHNAME :  [.BENCH.A.C.P1]shell0.sh
            (UNIX equivalent : bench/a/c/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           operations on signals.  The model simulated consists of a number of
           signal declarations and one addition signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of addition signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino

--  Date : 9 June 1989

--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal addition statements

--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 1;
--          signal sig1 : integer := 0;
--                      .
--                      .
--          signal sig10 : integer := 0;
--        begin
--          sig1 <= static_sig + 1;
--                      .
--                      .
--          sig10 <= static_sig + 1;
--        end test;



entity test is end;
architecture test of test is
  signal static_sig : integer := 1;
#1[  signal sig@ : integer := 0;]
begin
#1[  sig@ <= static_sig + 1;]
end test;
```

```
TEST NUMBER :  36

PATHNAME :  [.BENCH.A.C.P1]shell1.sh
            (UNIX equivalent : bench/a/c/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform addition
           operations on a signal; determine the number of addition operations
           allowed in one signal assignment statement.  The model simulated is
           an architecture consisting of an integer signal declaration and a
           signal assignment statement containing a number of addition
           operations.  The factor to be varied is the number of addition
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal additions in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 0;
--          signal sig : integer := 0;
--        begin
--          sig <= static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                              + static_sig
--                  ;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal static_sig : integer := 0;
  signal sig : integer := 0;
begin
  sig <= static_sig
#1[                    + static_sig]
        ;
end test;
```

```
TEST NUMBER :  37

PATHNAME :  [.BENCH.A.C.P2]shell0.sh
            (UNIX equivalent : bench/a/c/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute subtraction
           operations on signals.  The model simulated consists of a number of
           signal declarations and one subtraction signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of subtraction signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal subtraction statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 1;
--          signal sig1 : integer := 0;
--                      .
--                      .
--          signal sig10 : integer := 0;
--        begin
--          sig1 <= static_sig - 1;
--                  .
--                  .
--          sig10 <= static_sig - 1;
--        end test;



entity test is end;
architecture test of test is
  signal static_sig : integer := 1;
#1[  signal sig@ : integer := 0;]
begin
#1[  sig@ <= static_sig - 1;]
end test;
```

```
TEST NUMBER :  38

PATHNAME :  [.BENCH.A.C.P2]shell1.sh
            (UNIX equivalent : bench/a/c/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform subtraction
           operations on a signal; determine the number of subtraction
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of an integer signal
           declaration and a signal assignment statement containing a number of
           subtraction operations.  The factor to be varied is the number of
           subtraction operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal subtractions in ore statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 0;
--          signal sig : integer := 0;
--        begin
--          sig <= static_sig
--                              - static_sig
--                              - static_sig
-                               - static_sig
--                              - static_sig
--                              - static_sig
--                              - static_sig
--                              - static_sig
--                              - static_sig
--                              - static_sig
--                              - static_sig
--                  ;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal static_sig : integer := 0;
  signal sig : integer := 0;
begin
  sig <= static_sig
#1[                    - static_sig]
        ;
end test;
```

```
TEST NUMBER :  39

PATHNAME :  [.BENCH.A.C.P3]shell0.sh
            (UNIX equivalent : bench/a/c/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute multiplication
           operations on signals.  The model simulated consists of a number of
           signal declarations and one multiplication signal assignment
           statement for each signal.  The factor to be varied is the number of
           signal declarations/number of multiplication signal assignment
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal multiplication statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 1;
--          signal sig1 : integer := 0;
--                        .
--                        .
--          signal sig10 : integer := 0;
--        begin
--          sig1 <= static_sig * 1;
--                      .
--                      .
--          sig10 <= static_sig * 1;
--        end test;



entity test is end;
architecture test of test is
  signal static_sig : integer := 1;
#1[  signal sig@ : integer := 0;]
begin
#1[  sig@ <= static_sig * 1;]
```

```
end test;
```

```
TEST NUMBER :  40

PATHNAME :  [.BENCH.A.C.P3]shell1.sh
            (UNIX equivalent : bench/a/c/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform multiplication
           operations on a signal; determine the number of multiplication
           statements allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of an integer signal
           declaration and a signal assignment statement containing a number of
           multiplication operations.  The factor to be varied is the number of
           multiplication operations contained in the signal assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal multiplications in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 1;
--          signal sig : integer := 0;
--        begin
--          sig <= static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                              * static_sig
--                  ;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal static_sig : integer := 1;
  signal sig : integer := 0;
begin
  sig <= static_sig
#1[                    * static_sig]
        ;
end test;
```

```
TEST NUMBER :  41

PATHNAME :  [.BENCH.A.C.P4]shell0.sh
            (UNIX equivalent : bench/a/c/p4/shello.sh)

PURPOSE :  Determine the simulation CPU time required to execute division
           operations on signals.  The model simulated consists of a number of
           signal declarations and one division signal assignment statement for
           each signal.  The factor to be varied is the number of signal
           declarations/number of division signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal division statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 1;
--          signal sig1 : integer := 0;
--                        .
--                        .
--          signal sig10 : integer := 0;
--        begin
--          sig1 <= static_sig / 1;
--                  .
--                  .
--          sig10 <= static_sig / 1;
--        end test;



entity test is end;
architecture test of test is
  signal static_sig : integer := 1;
#1[  signal sig@ : integer := 0;]
begin
#1[  sig@ <= static_sig / 1;]
end test;
```

```
TEST NUMBER :  42

PATHNAME :  [.BENCH.A.C.P4]shell1.sh
            (UNIX equivalent : bench/a/c/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform division
           operations on a signal; determine the number of division operations
           allowed in one signal assignment statement.  The model simulated is
           an architecture consisting of an integer signal declaration and a
           signal assignment statement containing a number of division
           operations.  The factor to be varied is the number of division
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal divisions in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : integer := 1;
--          signal sig : integer := 0;
--        begin
--          sig <= static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                              / static_sig
--                  ;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal static_sig : integer := 1;
  signal sig : integer := 0;
begin
  sig <= static_sig
#1[                    / static_sig]
        ;
end test;
```

```
TEST NUMBER :  43

PATHNAME :  [.BENCH.A.C.P6]shell1.sh
            (UNIX equivalent : bench/a/c/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform logical AND
           operations on a signal; determine the number  of logical AND
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a signal declaration and
           a signal assignment statement containing a number of logical AND
           operations.  The factor to be varied is the number of logical AND
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of AND operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--          signal static_sig : bit := '1';
--        begin
--          sig <= static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                              AND static_sig
--                 ;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal sig : bit := '0';
  signal static_sig : bit := '1';
begin
  sig <= static_sig
#1[                    AND static_sig]
        ;
end test;
```

```
TEST NUMBER :  44

PATHNAME :  [.BENCH.B.C.P6]shell1.sh
            (UNIX equivalent : bench/b/c/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical AND
           operations on variables;  determine the number of logical AND
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of logical AND operations.
           The factors to be varied are the number of processes and the number
           of logical AND operations in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of AND operations in one statement of a process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit := '1';
--          begin
--            var := var
--                       AND var
--                          .
--                          . (10 repetitions total)
--                       AND var
--                    ;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit := '1';
--          begin
--            var := var
--                       AND var
```

```
--                          .
--                          . (10 repetitions total)
--                       AND var
--                    ;
--             wait;
--           end process pr2;
--         end test;



entity test is end;
architecture test of test is
begin
#2[  pr0 : process
    variable var : bit := '1';
  begin
    var := var
#1[                AND var]
           ;
    wait;
  end process pr0;]
end test;
```

```
TEST NUMBER :  45

PATHNAME :  [.BENCH.A.C.I2.P4]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division operations in a function; determine the number of division
           operations allowed in one variable assignment statement of a
           function.  The model simulated is an architecture consisting of a
           function declaration and a function call; the function consists of
           an integer variable declaration and a variable assignment statement
           containing a number of addition operations.  The factor to be varied
           is the number of division operations contained in the variable
           assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable division operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function divide return boolean is
--            variable var : integer := 1;
--          begin
--            var := var
--                       / var
--                       / var
--                       / var
--                       / var
--                       / var;
--            return true;
--          end divide;
--          signal done : boolean := false;
--        begin
--          done <= divide;
--        end test;
```

```
entity test is end;
architecture test of test is
  function divide return boolean is
    variable var : integer := 1;
  begin
    var := var
#1[               / var];
    return true;
  end divide;
  signal done : boolean := false;
begin
  done <= divide;
end test;
```

```
TEST NUMBER :  46

PATHNAME :  [.BENCH.A.C.H2.P4]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division operations in a procedure; determine the number of division
           operations allowed in one variable assignment statement of a
           procedure.  The model simulated is an architecture consisting of a
           procedure declaration and a procedure call; the procedure consists of
           an integer variable declaration and a variable assignment statement
           containing a number of addition operations.  The factor to be varied
           is the number of division operations contained in the variable
           assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable division operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--       entity test is end;
--       architecture test of test is
--         procedure divide is
--           variable var : integer := 1;
--         begin
--           var := var
--                      / var
--                      / var
--                      / var
--                      / var
--                      / var;
--         end divide;
--       begin
--         divide;
--       end test;
```

```
entity test is end;
architecture test of test is
  procedure divide is
    variable var : integer := 1;
  begin
    var := var
#1[                / var];
  end divide;
begin
  divide;
end test;
```

```
TEST NUMBER :  47

PATHNAME :  [.BENCH.A.C.I2.P3]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication operations in a function; determine the number of
           multiplication operations allowed in one variable assignment
           statement of a function.  The model simulated is an architecture
           consisting of a function declaration and a function call; the
           function consists of an integer variable declaration and a variable
           assignment statement containing a number of addition operations.
           The factor to be varied is the number of multiplication operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date . 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable multiplication operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function multiply return boolean is
--            variable var : integer := 1;
--          begin
--            var := var
--                       * var
--                       * var
--                       * var
--                       * var
--                       * var;
--            return true;
--          end multiply;
--          signal done : boolean := false;
--        begin
--          done <= multiply;
--        end test;
```

```
entity test is end;
architecture test of test is
  function multiply return boolean is
    variable var : integer := 1;
  begin
    var := var
#1[               * var];
    return true;
  end multiply;
  signal done : boolean := false;
begin
  done <= multiply;
end test;
```

```
TEST NUMBER :  48

PATHNAME :  [.BENCH.A.C.H2.P3]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p3/shell1.sh)

PURPOSE :   Determine the simulation CPU time required to execute variable
            multiplication operations in a procedure; determine the number of
            multiplication operations allowed in one variable assignment
            statement of a procedure.  The model simulated is an architecture
            consisting of a procedure declaration and a procedure call; the
            procedure consists of an integer variable declaration and a variable
            assignment statement containing a number of addition operations.
            The factor to be varied is the number of multiplication operations
            contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable multiplication operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure multiply is
--            variable var : integer := 1;
--          begin
--            var := var
--                       * var
--                       * var
--                       * var
--                       * var
--                       * var;
--          end multiply;
--        begin
--          multiply;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure multiply is
    variable var : integer := 1;
  begin
    var := var
#1[               * var];
  end multiply;
begin
  multiply;
end test;
```

```
TEST NUMBER :  49

PATHNAME :  [.BENCH.A.C.H2.P2]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction operations in a procedure; determine the number of
           subtraction operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call; the
           procedure consists of an integer variable declaration and a variable
           assignment statement containing a number of addition operations.
           The factor to be varied is the number of subtraction operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable subtraction operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure subtract is
--            variable var : integer := 0;
--          begin
--            var := var
--                        - var
--                        - var
--                        - var
--                        - var
--                        - var;
--          end subtract;
--        begin
--          subtract;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure subtract is
    variable var : integer := 0;
  begin
    var := var #1[- var];
  end subtract;
begin
  subtract;
end test;
```

```
TEST NUMBER :  50

PATHNAME :  [.BENCH.A.C.I2.P2]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction operations in a function; determine the number of
           subtraction operations allowed in one variable assignment
           statement of a function.  The model simulated is an architecture
           consisting of a function declaration and a function call; the
           function consists of an integer variable declaration and a variable
           assignment statement containing a number of addition operations.
           The factor to be varied is the number of subtraction operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable subtraction operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function subtract return boolean is
--            variable var : integer := 0;
--          begin
--            var := var
--                       - var
--                       - var
--                       - var
--                       - var
--                       - var;
--            return true;
--          end subtract;
--          signal done : boolean := false;
--        begin
--          done <= subtract;
--        end test;
```

```
entity test is end;
architecture test of test is
  function subtract return boolean is
    variable var : integer := 0;
  begin
    var := var #1[- var];
    return true;
  end subtract;
  signal done : boolean := false;
begin
  done <= subtract;
end test;
```

```
TEST NUMBER :  51

PATHNAME :  [.BENCH.A.C.I2.P1]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition operations in a function; determine the number of
           addition operations allowed in one variable assignment
           statement of a function.  The model simulated is an architecture
           consisting of a function declaration and a function call; the
           function consists of an integer variable declaration and a variable
           assignment statement containing a number of addition operations.
           The factor to be varied is the number of addition operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 9 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable addition operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function add return boolean is
--            variable var : integer := 0;
--          begin
--            var := var
--                       + var
--                       + var
--                       + var
--                       + var
--                       + var;
--            return true;
--          end add;
--          signal done : boolean := false;
--        begin
--          done <= add;
--        end test;
```

```
entity test is end;
architecture test of test is
  function add return boolean is
    variable var : integer := 0;
  begin
    var := var #1[+ var];
    return true;
  end add;
  signal done : boolean := false;
begin
  done <= add;
end test;
```

```
TEST NUMBER :  52

PATHNAME :  [.BENCH.A.C.H2.P1]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition operations in a procedure; determine the number of
           addition operations allowed in one variable assignment statement of
           a procedure.  The model simulated is an architecture consisting of a
           procedure declaration and a procedure call; the procedure consists
           of an integer variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of addition operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino

--  Date : 9 June 1989

--  PARAMETER NUMBER MEANING :
--    1 : number of variable addition operations in one statement in
--        procedure

--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure add is
--            variable var : integer := 0;
--          begin
--            var := var
--                       + var
--                       + var
--                       + var
--                       + var
--                       + var;
--          end add;
--        begin
--          add;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure add is
    variable var : integer := 0;
  begin
    var := var #1[+ var];
  end add;
begin
  add;
end test;
```

```
TEST NUMBER :  53

PATHNAME :  [.BENCH.B.C.P7]shell0.sh
            (UNIX equivalent : bench/b/c/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing logical
           OR operations on variables.  The model simulated is an architecture
           consisting of a number of processes; each process consists of a
           number of variable declarations and one logical OR variable
           assignment statement for each variable.  The factors to be varied
           are the number of processes and the number of variable declarations/
           number of logical OR variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/OR statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",5,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,5,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--            variable var4 : bit := '1';
--            variable var5 : bit := '0';
--          begin
--            var1 := var1 OR var1;
--            var2 := var2 OR var2;
--            var3 := var3 OR var3;
--            var4 := var4 OR var4;
--            var5 := var5 OR var5;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
```

```
--            variable var3 : bit := '0';
--            variable var4 : bit := '1';
--            variable var5 : bit := '0';
--          begin
--            var1 := var1 OR var1;
--            var2 := var2 OR var2;
--            var3 := var3 OR var3;
--            var4 := var4 OR var4;
--            var5 := var5 OR var5;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ OR var@;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  54

PATHNAME :  [.BENCH.B.C.K.L1.P7]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p7/shell0.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes; each process consists of a variable bit_vector
           declaration and a for-loop; the for-loop contains a logical OR
           statement, and the number of iterations of the loop is equal to
           the size of the variable bit_vector : number of processes, bit_vector
           size/number of iterations of for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of OR statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) OR var(i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) OR var(i);
--            end loop;
--            wait;
--          end process pr2;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit_vector(1 to %1%);
  begin
    for i in 1 to %1% loop
      var(i) := var(i) OR var(i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  55

PATHNAME :  [.BENCH.A.C.P7]shell0.sh
            (UNIX equivalent : bench/a/c/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical OR
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical OR signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of logical OR signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/OR statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : bit := '0';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '0';
--        begin
--          sig1 <= sig1 OR sig1;
--          sig2 <= sig2 OR sig2;
--          sig3 <= sig3 OR sig3;
--        end test;



entity test is end;
architecture test of test is
#1[  signal sig@ : bit := '$2$0$1$';]
begin
#1[  sig@ <= sig@ OR sig@;]
end test;
```

```
TEST NUMBER :  56

PATHNAME :  [.BENCH.A.C.P7]shell1.sh
            (UNIX equivalent : bench/a/c/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform logical OR
           operations on a signal; determine the number  of logical OR
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a signal declaration and
           a signal assignment statement containing a number of logical OR
           operations.  The factor to be varied is the number of logical OR
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of OR operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--          signal static_sig : bit := '1';
--        begin
--          sig <= static_sig
--                              OR static_sig
--                              OR static_sig
--                              OR static_sig;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit := '0';
  signal static_sig : bit := '1';
begin
  sig <= static_sig
#1[                     OR static_sig];
end test;
```

```
TEST NUMBER :  57

PATHNAME :  [.BENCH.B.C.P7]shell1.sh
            (UNIX equivalent : bench/b/c/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical OR
           operations on variables;  determine the number of logical OR
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of logical OR operations.
           The factors to be varied are the number of processes and the number
           of logical OR operations in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of OR operations in one statement of a process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit := '1';
--          begin
--            var := var
--                       OR var
--                       OR var
--                       OR var;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit := '1';
--          begin
--            var := var
--                       OR var
--                       OR var
--                       OR var;
```

```
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit := '1';
  begin
    var := var
#1[                OR var];
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  58

PATHNAME :  [.BENCH.B.C.P8]shell0.sh
            (UNIX equivalent : bench/b/c/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing logical
           NAND operations on variables.  The model simulated is an architecture
           consisting of a number of processes; each process consists of a
           number of variable declarations and one logical NAND variable
           assignment statement for each variable.  The factors to be varied
           are the number of processes and the number of variable declarations/
           number of logical NAND variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/NAND statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NAND var1;
--            var2 := var2 NAND var2;
--            var3 := var3 NAND var3;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NAND var1;
--            var2 := var2 NAND var2;
```

```
--            var3 := var3 NAND var3;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NAND var@;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  59

PATHNAME :  [.BENCH.B.C.K.L1.P8]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p8/shell0.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes; each process consists of a variable bit_vector
           declaration and a for-loop; the for-loop contains a logical NAND
           statement, and the number of iterations of the loop is equal to
           the size of the variable bit_vector : number of processes, bit_vector
           size/number of iterations of for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of NAND statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) NAND var(i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) NAND var(i);
--            end loop;
--            wait;
--          end process pr2;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit_vector(1 to %1%);
  begin
    for i in 1 to %1% loop
      var(i) := var(i) NAND var(i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  60

PATHNAME :  [.BENCH.A.C.P8]shell0.sh
            (UNIX equivalent : bench/a/c/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NAND
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical NAND signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of logical NAND signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/NAND statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= sig1 NAND '0';
--          sig2 <= sig2 NAND '0';
--          sig3 <= sig3 NAND '0';
--        end test;



entity test is end;
architecture test of test is
#1[  signal sig@ : bit := '1';]
begin
#1[  sig@ <= sig@ NAND '0';]
end test;
```

```
TEST NUMBER :  61

PATHNAME :  [.BENCH.A.C.P8]shell1.sh
            (UNIX equivalent : bench/a/c/p8/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform logical NAND
           operations on a signal; determine the number  of logical NAND
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a signal declaration and
           a signal assignment statement containing a number of logical NAND
           operations.  The factor to be varied is the number of logical NAND
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of NAND operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--          signal static_sig : bit := '0';
--        begin
--          sig <= static_sig
--                          NAND (static_sig
--                          NAND (static_sig
--                          NAND (static_sig
--                                        )
--                                        )
--                                        );
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit := '0';
  signal static_sig : bit := '0';
begin
```

```
  sig <= static_sig
#1[                    NAND (static_sig]#1[)];
end test;
```

```
TEST NUMBER :  62

PATHNAME :  [.BENCH.B.C.P8]shell1.sh
            (UNIX equivalent : bench/b/c/p8/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NAND
           operations on variables;  determine the number of logical NAND
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of logical NAND operations.
           The factors to be varied are the number of processes and the number
           of logical NAND operations in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 12 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of NAND operations in one statement of a process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--      entity test is end;
--      architecture test of test is
--      begin
--        pr1 : process
--          variable var : bit := '1';
--        begin
--          var := var
--                     NAND (var
--                     NAND (var
--                     NAND (var
--                              )
--                              )
--                              );
--          wait;
--        end process pr1;
--        pr2 : process
--          variable var : bit := '1';
--        begin
--          var := var
```

```
--                      NAND (var
--                      NAND (var
--                      NAND (var
--                                )
--                                )
--                                );
--          wait;
--        end process pr2;
--      end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit := '1';
  begin
    var := var
#1[                NAND (var]
#1[                          )];
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  63

PATHNAME :  [.BENCH.B.C.P9]shell0.sh
            (UNIX equivalent : bench/b/c/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing logical
           NOR operations on variables.  The model simulated is an architecture
           consisting of a number of processes; each process consists of a
           number of variable declarations and one logical NOR variable
           assignment statement for each variable.  The factors to be varied
           are the number of processes and the number of variable declarations/
           number of logical NOR variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/NOR statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NOR var1;
--            var2 := var2 NOR var2;
--            var3 := var3 NOR var3;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NOR var1;
--            var2 := var2 NOR var2;
```

```
--            var3 := var3 NOR var3;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NOR var@;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  64

PATHNAME :  [.BENCH.B.C.K.L1.P9]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p9/shell0.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes; each process consists of a variable bit_vector
           declaration and a for-loop; the for-loop contains a logical NOR
           statement, and the number of iterations of the loop is equal to
           the size of the variable bit_vector : number of processes, bit_vector
           size/number of iterations of for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of NOR statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) NOR var(i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) NOR var(i);
--            end loop;
--            wait;
--          end process pr2;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit_vector(1 to %1%);
  begin
    for i in 1 to %1% loop
      var(i) := var(i) NOR var(i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  65

PATHNAME :  [.BENCH.A.C.P9]shell0.sh
            (UNIX equivalent : bench/a/c/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOR
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical NOR signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of logical NOR signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/NOR statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : bit := '0';
--          signal sig2 : bit := '0';
--          signal sig3 : bit := '0';
--        begin
--          sig1 <= sig1 NOR '1';
--          sig2 <= sig2 NOR '1';
--          sig3 <= sig3 NOR '1';
--        end test;



entity test is end;
architecture test of test is
#1[  signal sig@ : bit := '0';]
begin
#1[  sig@ <= sig@ NOR '1';]
end test;
```

```
TEST NUMBER :  66

PATHNAME :  [.BENCH.A.C.P9]shell1.sh
            (UNIX equivalent : bench/a/c/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform logical NOR
           operations on a signal; determine the number  of logical NOR
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a signal declaration and
           a signal assignment statement containing a number of logical NOR
           operations.  The factor to be varied is the number of logical NOR
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of NOR operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--          signal static_sig : bit := '0';
--        begin
--          sig <= static_sig
--                          NOR (static_sig
--                          NOR (static_sig
--                          NOR (static_sig
--                                       )
--                                       )
--                                       );
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit := '0';
  signal static_sig : bit := '0';
begin
```

```
  sig <= static_sig
#1[                    NOR (static_sig]#1[)];
end test;
```

```
TEST NUMBER :  67

PATHNAME :  [.BENCH.B.C.P9]shell1.sh
            (UNIX equivalent : bench/b/c/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOR
           operations on variables;  determine the number of logical NOR
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of logical NOR operations.
           The factors to be varied are the number of processes and the number
           of logical NOR operations in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of NOR operations in one statement of a process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit := '1';
--          begin
--            var := var
--                       NOR (var
--                       NOR (var
--                       NOR (var
--                                )
--                                )
--                                );
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit := '1';
--          begin
--            var := var
```

```
--                      NOR (var
--                      NOR (var
--                      NOR (var
--                              )
--                              )
--                              );
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit := '1';
  begin
    var := var
#1[               NOR (var]
#1[                       )];
    wait;
  end process pr@;]
end test;
```

TEST NUMBER : 68

PATHNAME : [.BENCH.B.C.P10]shell0.sh
(UNIX equivalent : bench/b/c/p10/shell0.sh)

PURPOSE : Determine the simulation CPU time required for executing logical XOR operations on variables. The model simulated is an architecture consisting of a number of processes; each process consists of a number of variable declarations and one logical XOR variable assignment statement for each variable. The factors to be varied are the number of processes and the number of variable declarations/ number of logical XOR variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/XOR statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 XOR var1;
--            var2 := var2 XOR var2;
--            var3 := var3 XOR var3;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 XOR var1;
--            var2 := var2 XOR var2;
```

```
--            var3 := var3 XOR var3;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ XOR var@;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  69

PATHNAME :  [.BENCH.B.C.K.L1.P10]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p10/shell0.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes; each process consists of a variable bit_vector
           declaration and a for-loop; the for-loop contains a logical XOR
           statement, and the number of iterations of the loop is equal to
           the size of the variable bit_vector : number of processes, bit_vector
           size/number of iterations of for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of XOR statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) XOR var(i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := var(i) XOR var(i);
--            end loop;
--            wait;
--          end process pr2;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit_vector(1 to %1%);
  begin
    for i in 1 to %1% loop
      var(i) := var(i) XOR var(i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  70

PATHNAME :  [.BENCH.A.C.P10]shell0.sh
            (UNIX equivalent : bench/a/c/p10/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical XOR
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical XOR signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of logical XOR signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/XOR statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= sig1 XOR '0';
--          sig2 <= sig2 XOR '0';
--          sig3 <= sig3 XOR '0';
--        end test;



entity test is end;
architecture test of test is
#1[  signal sig@ : bit := '1';]
begin
#1[  sig@ <= sig@ XOR '0';]
end test;
```

```
TEST NUMBER :  71

PATHNAME :  [.BENCH.A.C.P10]shell1.sh
            (UNIX equivalent : bench/a/c/p10/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform logical XOR
           operations on a signal; determine the number  of logical XOR
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a signal declaration and
           a signal assignment statement containing a number of logical XOR
           operations.  The factor to be varied is the number of logical XOR
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of XOR operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--          signal static_sig : bit := '0';
--        begin
--          sig <= static_sig
--                              XOR static_sig
--                              XOR static_sig
--                              XOR static_sig;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit := '0';
  signal static_sig : bit := '0';
begin
  sig <= static_sig
#1[                     XOR static_sig];
end test;
```

```
TEST NUMBER :  72

PATHNAME :  [.BENCH.B.C.P10]shell1.sh
            (UNIX equivalent : bench/b/c/p10/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical XOR
           operations on variables;  determine the number of logical XOR
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of logical XOR operations.
           The factors to be varied are the number of processes and the number
           of logical XOR operations in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of XOR operations in one statement of a process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit := '1';
--          begin
--            var := var
--                       XOR var
--                       XOR var
--                       XOR var;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit := '1';
--          begin
--            var := var
```

```
--                          XOR var
--                          XOR var
--                          XOR var;
--              wait;
--            end process pr2;
--          end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit := '1';
  begin
    var := var
#1[                XOR var];
    wait;
  end process pr@;]
end test;
```

TEST NUMBER : 73

PATHNAME : [.BENCH.B.C.P11]shell0.sh
(UNIX equivalent : bench/b/c/p11/shell0.sh)

PURPOSE : Determine the simulation CPU time required for executing logical NOT operations on variables. The model simulated is an architecture consisting of a number of processes; each process consists of a number of variable declarations and one logical NOT variable assignment statement for each variable. The factors to be varied are the number of processes and the number of variable declarations/ number of logical NOT variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/NOT statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := NOT var1;
--            var2 := NOT var2;
--            var3 := NOT var3;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := NOT var1;
--            var2 := NOT var2;
```

```
--            var3 := NOT var3;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := NOT var@;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  74

PATHNAME :  [.BENCH.B.C.K.L1.P11]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p11/shell0.sh)

PURPOSE :  Determine the effect on simulation CPU time when the following
           factors are varied in an architecture consisting of a number of
           processes; each process consists of a variable bit_vector
           declaration and a for-loop; the for-loop contains a logical NOT
           statement, and the number of iterations of the loop is equal to
           the size of the variable bit_vector : number of processes, bit_vector
           size/number of iterations of for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of NOT statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := NOT var(i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit_vector(1 to 3);
--          begin
--            for i in 1 to 3 loop
--              var(i) := NOT var(i);
--            end loop;
--            wait;
--          end process pr2;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit_vector(1 to %1%);
  begin
    for i in 1 to %1% loop
      var(i) := NOT var(i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  75

PATHNAME :  [.BENCH.A.C.P11]shell0.sh
            (UNIX equivalent : bench/a/c/p11/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOT
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical NOT signal assignment statement
           for each signal.  The factor to be varied is the number of signal
           declarations/number of logical NOT signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/NOT statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal static_sig : bit := '1';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= NOT static_sig;
--          sig2 <= NOT static_sig;
--          sig3 <= NOT static_sig;
--        end test;



entity test is end;
architecture test of test is
  signal static_sig : bit := '1';
#1[  signal sig@ : bit := '1';]
begin
#1[  sig@ <= NOT static_sig;]
end test;
```

```
TEST NUMBER :  76

PATHNAME :  [.BENCH.A.C.P11]shell1.sh
            (UNIX equivalent : bench/a/c/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform logical NOT
           operations on a signal; determine the number  of logical NOT
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a signal declaration and
           a signal assignment statement containing a number of logical NOT
           operations.  The factor to be varied is the number of logical NOT
           operations contained in the signal assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of NOT operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--          signal static_sig : bit := '0';
--        begin
--          sig <=
--               NOT (
--               NOT (
--               NOT (
--                    static_sig
--                              )
--                              )
--                              );
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit := '0';
  signal static_sig : bit := '0';
```

```
begin
  sig <=
#1[       NOT (]
            static_sig
#1[                        )];
end test;
```

```
TEST NUMBER :  77

PATHNAME :  [.BENCH.B.C.P11]shell1.sh
            (UNIX equivalent : bench/b/c/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOT
           operations on variables;  determine the number of logical NOT
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of logical NOT operations.
           The factors to be varied are the number of processes and the number
           of logical NOT operations in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 13 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of NOT operations in one statement of a process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : bit := '1';
--          begin
--            var :=
--                NOT (
--                NOT (
--                NOT (
--                     var
--                        )
--                        )
--                        );
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : bit := '1';
--          begin
```

```
--            var :=
--                NOT (
--                NOT (
--                NOT (
--                     var
--                        )
--                        )
--                        );
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : bit := '1';
  begin
    var :=
#1[        NOT (]
             var
#1[                )];
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  78

PATHNAME :  [.BENCH.A.C.P5]shell0.sh
            (UNIX equivalent : bench/a/c/p5/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute concatenation
           operations on string signals.  The model simulated consists of a
           number of string signal declarations and one concatenation signal
           assignment statement for each signal.  The factors to be varied are
           the number of signal declarations/number of concatenation signal
           assignment statements and the lengths of the strings to be
           concatenated.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 14 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of string_1
--    2 : length of string_2
--    3 : number of result_string declarations/number of concatenation
--        statements
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,5,3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,5,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal string_1 : string(1 to 10) := "ABCDEFGHIJ";
--          signal string_2 : string(1 to 5) := "ABCDE";
--          signal result_string1 : string(1 to (10 + 5));
--          signal result_string2 : string(1 to (10 + 5));
--          signal result_string3 : string(1 to (10 + 5));
--        begin
--          result_string1 <= string_1 & string_2;
--          result_string2 <= string_1 & string_2;
--          result_string3 <= string_1 & string_2;
--        end test;



entity test is end;
architecture test of test is
  signal string_1 : string(1 to %1%) := "?1?";
```

```
  signal string_2 : string(1 to %2%) := "?2?";
#3[  signal result_string@ : string(1 to (%1% + %2%));]
begin
#3[  result_string@ <= string_1 & string_2;]
end test;
```

```
TEST NUMBER :  79

PATHNAME :  [.BENCH.A.C.P5]shell1.sh
            (UNIX equivalent : bench/a/c/p5/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform concatenation
           operations on a signal; determine the number  of concatenation
           operations allowed in one signal assignment statement.  The model
           simulated is an architecture consisting of a string signal
           declaration and a signal assignment statement containing a number of
           concatenation operations.  The factors to be varied are the number
           of concatenation operations contained in the signal assignment
           statement and the length of the string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 14 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of string used in concatenation operations
--    2 : number of concatenation operations in one statement
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal op_string : string(1 to 10) := "ABCDEFGHIJ";
--          signal result_string : string(1 to (10 * (3 + 1)));
--        begin
--          result_string <= op_string
--                                    & op_string
--                                    & op_string
--                                    & op_string,
--        end test;



entity test is end;
architecture test of test is
  signal op_string : string(1 to %1%) := "?1?";
  signal result_string : string(1 to (%1% * (%2% + 1)));
begin
  result_string <= op_string
```

```
#2[                              & op_string];
end test;
```

```
TEST NUMBER :  80

PATHNAME :  [.BENCH.B.C.P5]shell0.sh
            (UNIX equivalent : bench/b/c/p5/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing
           concatenation operations on variables.  The model simulated is an
           architecture consisting of a number of processes; each process
           consists of a number of string variable declarations and one
           concatenation variable assignment statement for each variable.  The
           factors to be varied are the number of processes, the number of
           variable declarations/number of concatenation variable assignment
           statements, and the length of the string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable declarations/concatenation statements per process
--    3 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var1 : string(1 to (10 + 10));
--            variable var2 : string(1 to (10 + 10));
--          begin
--            var1 := static_var & static_var;
--            var2 := static_var & static_var;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var1 : string(1 to (10 + 10));
--            variable var2 : string(1 to (10 + 10));
--          begin
--            var1 := static_var & static_var;
```

```
--            var2 := static_var & static_var;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#3[  pr@ : process
    variable static_var : string(1 to %1%) := "?1?";
#2[    variable var@ : string(1 to (%1% + %1%));]
  begin
#2[    var@ := static_var & static_var;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  81

PATHNAME :  [.BENCH.B.C.P5]shell1.sh
            (UNIX equivalent : bench/b/c/p5/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute concatenation
           operations on variables;  determine the number of concatenation
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of two string variable declarations and a
           variable assignment statement containing a number of concatenation
           operations.  The factors to be varied are the number of processes,
           the number of concatenation operations in the variable assignment
           statement, and the length of the string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of concatenation operations in one statement of a process
--    3 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10,3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var : string(1 to (10 * (3 + 1)));
--          begin
--            var := static_var
--                              & static_var
--                              & static_var
--                              & static_var;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var : string(1 to (10 * (3 + 1)));
--          begin
```

```
--            var := static_var
--                                & static_var
--                                & static_var
--                                & static_var;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#3[  pr@ : process
    variable static_var : string(1 to %1%) := "?1?";
    variable var : string(1 to (%1% * (%2% + 1)));
  begin
    var := static_var
#2[                       & static_var];
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  82

PATHNAME :  [.BENCH.A.C.H2.P5]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p5/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of string variable declarations
           and a concatenation statement for each variable.  The factors to be
           varied are the number of variable declarations/concatenation
           statements in the procedure and the length of the strings used in
           the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable declarations/variable concatenation statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure concatenate is
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var1 : string(1 to (10 + 10));
--            variable var2 : string(1 to (10 + 10));
--            variable var3 : string(1 to (10 + 10));
--          begin
--            var1 := static_var & static_var;
--            var2 := static_var & static_var;
--            var3 := static_var & static_var;
--          end concatenate;
--        begin
--          concatenate;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure concatenate is
    variable static_var : string(1 to %1%) := "?1?";
#2[    variable var@ : string(1 to (%1% + %1%));]
  begin
#2[    var@ := static_var & static_var;]
  end concatenate;
begin
  concatenate;
end test;
```

```
TEST NUMBER :  83

PATHNAME :  [.BENCH.A.C.H2.P5]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p5/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation operations in a procedure; determine the number of
           concatenation operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call; the
           procedure consists of a string variable declaration and a variable
           assignment statement containing a number of concatenation operations.
           The factors to be varied are the number of concatenation operations
           contained in the variable assignment statement and the length of the
           string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable concatenation operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure concatenate is
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var : string(1 to (10 * (3 + 1)));
--          begin
--            var := static_var
--                                & static_var
--                                & static_var
--                                & static_var;
--          end concatenate;
--        begin
--          concatenate;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure concatenate is
    variable static_var : string(1 to %1%) := "?1?";
    variable var : string(1 to (%1% * (%2% + 1)));
  begin
    var := static_var #2[& static_var];
  end concatenate;
begin
  concatenate;
end test;
```

TEST NUMBER : 84

PATHNAME : [.BENCH.A.C.I2.P5]shell0.sh
(UNIX equivalent : bench/a/c/i2/p5/shell0.sh)

PURPOSE : Determine the simulation CPU time required to execute variable concatenation statements. The model simulated is an architecture consisting of a function declaration and a function call. The function consists of a number of string variable declarations and a concatenation statement for each variable. The factors to be varied are the number of variable declarations/concatenation statements in the function and the length of the string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable declarations/variable concatenation statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function concatenate return boolean is
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var1 : string(1 to (10 + 10));
--            variable var2 : string(1 to (10 + 10));
--            variable var3 : string(1 to (10 + 10));
--          begin
--            var1 := static_var & static_var;
--            var2 := static_var & static_var;
--            var3 := static_var & static_var;
--            return true;
--          end concatenate;
--          signal return_sig : boolean := false;
--        begin
--          return_sig <= concatenate;
--        end test;
```

```
entity test is end;
architecture test of test is
  function concatenate return boolean is
    variable static_var : string(1 to %1%) := "?1?";
#2[    variable va @ : string(1 to (%1% + %1%));]
  begin
#2[    var@ := static_var & static_var;]
    return true;
  end concatenate;
  signal return_sig : boolean := false;
begin
  return_sig <= concatenate;
end test;
```

```
TEST NUMBER :  85

PATHNAME :  [.BENCH.A.C.I2.P5]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p5/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation operations in a function; determine the number of
           concatenation operations allowed in one variable assignment
           statement of a function.  The model simulated is an architecture
           consisting of a function declaration and a function call; the
           function consists of two string variable declarations and a variable
           assignment statement containing a number of concatenation operations.
           The factors to be varied are the number of concatenation operations
           contained in the variable assignment statement and the length of the
           string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable concatenation operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function concatenate return boolean is
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var : string(1 to (10 * (3 + 1)));
--          begin
--            var := static_var
--                              & static_var
--                              & static_var
--                              & static_var;
--            return true;
--          end concatenate;
--          signal return_sig : boolean := false;
--        begin
--          return_sig <= concatenate;
--        end test;
```

```
entity test is end;
architecture test of test is
  function concatenate return boolean is
    variable static_var : string(1 to %1%) := "?1?";
    variable var : string(1 to (%1% * (%2% + 1)));
  begin
    var := static_var #2[& static_var];
    return true;
  end concatenate;
  signal return_sig : boolean := false;
begin
  return_sig <= concatenate;
end test;
```

```
TEST NUMBER :  86

PATHNAME :  [.BENCH.A.C.H1.L1.M.S2]shell.sh
            (UNIX equivalent : bench/a/c/h1/l1/m/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required for writing characters
           to an output file.  The model simulated is an entity consisting of a
           procedure declaration and an architecture consisting of a procedure
           call.  The procedure consists of a for-loop whose number of
           iterations is equal to the number of characters to write (excluding
           linefeeds).  The for-loop contains a write character statement and
           an if-then construct containing a write linefeed statement.  The
           factor to be varied is the number of non-linefeed characters to
           write.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 15 June 1989
--
--  PARAMETER MEANING :
--    1 : number of characters to write to output file "data_file.dat";
--        after every 75 characters, a linefeed is written
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file(number_of_characters : in integer) is
--          begin
--            for i in 1 to number_of_characters loop
--              write(out_file,'*');
--              if (i mod 75) =  0 then
--                write(out_file,lf);
--              end if;
--            end loop;
--          end write_file;
--        end test;
--        architecture test of test is
--        begin
--          write_file(100);
--        end test;
--
```

```
--      After simulating kernel test, "data_file.dat" will be as follows :
--  ********************************************************************************
--  *************************


entity test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file(number_of_characters : in integer) is
  begin
    for i in 1 to number_of_characters loop
      write(out_file,'*');
      if (i mod 75) =  0 then
        write(out_file,lf);
      end if;
    end loop;
  end write_file;
end test;
architecture test of test is
begin
  write_file(%1%);
end test;
```

```
TEST NUMBER :  87

PATHNAME :  [.BENCH.A.C.H1.L3.S2]shell.sh
            (UNIX equivalent : bench/a/c/h1/l3/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required for writing characters
           to an output file.  The model simulated is an entity consisting of
           a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a while-loop whose number
           of iterations is equal to the number of non-linefeed characters to
           write.  The while-loop contains a write character/write linefeed
           combination and a subtraction statement to decrement the loop-
           counter.  The factor to be varied is the number of non-linefeed
           characters to write.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 15 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file is
--            variable number_of_lines : integer := 100;
--          begin
--            while number_of_lines > 0 loop
--              write(out_file,'*');
--              write(out_file,lf);
--              number_of_lines := number_of_lines - 1;
--            end loop;
--          end write_file;
--        end test;
--        architecture test of test is
--        begin
--          write_file;
--        end test;
--
```

```
--      After simulating kernel test, "data_file.dat" will be as follows :
--      (100 lines)
--        *
--        *
--        .
--        .
--        *
--        *


entity test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file is
    variable number_of_lines : integer := %1%;
  begin
    while number_of_lines > 0 loop
      write(out_file,'*');
      write(out_file,lf);
      number_of_lines := number_of_lines - 1;
    end loop;
  end write_file;
end test;
architecture test of test is
begin
  write_file;
end test;
```

```
TEST NUMBER :  88

PATHNAME :  [.BENCH.A.C.H1.S2]shell.sh
            (UNIX equivalent : bench/a/c/h1/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required to write characters
           to an output file.  The model simulated is an entity consisting of
           a procedure declaration and an architecture consisting of a
           a procedure call.  The procedure consists of a number of write
           character/write linefeed combinations.  The factor to be varied is
           the number of write character/write linefeed pairs in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE : 15 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",100
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,100)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file is
--          begin
--            write(out_file,'*');
--            write(out_file,lf);
--                    .
--                    .   (100 repetitions)
--                    .
--            write(out_file,'*');
--            write(out_file,lf);
--          end write_file;
--        end test;
--        architecture test of test is
--        begin
--          write_file;
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
--      (100 lines)
```

```
--        *
--        *
--        .
--        .
--        *
--        *


entity test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file is
  begin
#1[    write(out_file,'*');
    write(out_file,lf);]
  end write_file;
end test;
architecture test of test is
begin
  write_file;
end test;
```

```
TEST NUMBER :  89

PATHNAME :  [.BENCH.A.C.H1R.M.S2]shell.sh
            (UNIX equivalent : bench/a/c/h1r/m/s2/shell.sh)

PURPOSE :  Determine the simulation CPU time required for writing characters
           to an output file.  The model simulated is an entity consisting of
           a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a write character/write
           linefeed combination and a recursive call to itself.  The factor to
           be varied is the n mber of times the procedure will recursively call
           itself.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  DATE . 15 June 1989
--
--  PARAMETER MEANING :
--    1 : number of lines to write to output file "data_file.dat"; each
--        line consists of a "*" followed by a linefeed
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",4
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          type char_file is file of character;
--          file out_file : char_file is out "data_file.dat";
--          procedure write_file(number_of_lines : in integer) is
--          begin
--            write(out_file,'*');
--            write(out_file,lf);
--            if number_of_lines > 1 then
--              write_file(number_of_lines - 1);
--            end if;
--          end write_file;
--        end test;
--        architecture test of test is
--        begin
--          write_file(4);
--        end test;
--
--      After simulating kernel test, "data_file.dat" will be as follows :
--        *
--        *
```

```
--        *
--        *


entity test is
  type char_file is file of character;
  file out_file : char_file is out "data_file.dat";
  procedure write_file(number_of_lines : in integer) is
  begin
    write(out_file,'*');
    write(out_file,lf);
    if number_of_lines > 1 then
      write_file(number_of_lines - 1);
    end if,
  end write_file;
end test;
architecture test of test is
begin
  write_file(%1%);
end test;
```

```
TEST NUMBER :  90

PATHNAME :  [.BENCH.A.C.H1.P1]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of integer
           variable declarations and an addition statement for each variable.
           The factor to be varied is the number of variable declarations/
           addition statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable addition statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure add is
--            variable var1 : integer := 0;
--            variable var2 : integer := 0;
--            variable var3 : integer := 0;
--          begin
--            var1 := var1 +  1;
--            var2 := var2 +  1;
--            var3 := var3 +  1;
--          end add;
--        end test;
--        architecture test of test is
--        begin
--          add;
--        end test;



entity test is
  procedure add is
```

```
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ +  1;]
  end add;
end test;
architecture test of test is
begin
  add;
end test;
```

```
TEST NUMBER :  91

PATHNAME :  [.BENCH.A.C.H1.P1]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition operations in a procedure; determine the number of
           addition operations allowed in one variable assignment statement of
           a procedure.  The model simulated is an entity consisting of a
           procedure declaration and an architecture consisting of a procedure
           call; the procedure consists of an integer variable declaration and
           a variable assignment statement containing a number of addition
           operations.  The factor to be varied is the number of addition
           operations contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable addition operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure add is
--            variable var : integer := 0;
--          begin
--            var := var
--                       + var
--                       + var
--                       + var;
--          end add;
--        end test;
--        architecture test of test is
--        begin
--          add;
--        end test;


entity test is
```

```
  procedure add is
    variable var : integer := 0;
  begin
    var := var #1[+ var];
  end add;
end test;
architecture test of test is
begin
  add;
end test;
```

```
TEST NUMBER :  92

PATHNAME :  [.BENCH.A.C.I1.P1]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition statements.  The model simulated is an entity consisting
           of a function declaration and an architecture consisting of a
           function call.  The function consists of a number of integer
           variable declarations and an addition statement for each variable.
           The factor to be varied is the number of variable declarations/
           addition statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable addition statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function add return boolean is
--            variable var1 : integer := 0;
--            variable var2 : integer := 0;
--            variable var3 : integer := 0;
--          begin
--            var1 := var1 +  1;
--            var2 := var2 +  1;
--            var3 := var3 +  1;
--            return true;
--          end add;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= add;
--        end test;


entity test is
```

```
  function add return boolean is
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ +  1;]
    return true;
  end add;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= add;
end test;
```

```
TEST NUMBER :  93

PATHNAME :  [.BENCH.A.C.I1.P1]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition operations in a function; determine the number of
           addition operations allowed in one variable assignment
           statement of a function.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call; the function consists of an integer variable
           declaration and a variable assignment statement containing a number
           of addition operations.  The factor to be varied is the number of
           addition operations contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable addition operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function add return boolean is
--            variable var : integer := 0;
--          begin
--            var := var
--                       + var
--                       + var
--                       + var;
--            return true;
--          end add;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= add;
--        end test;
```

```
entity test is
  function add return boolean is
    variable var : integer := 0;
  begin
    var := var #1[+ var];
    return true;
  end add;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= add;
end test;
```

```
TEST NUMBER :  94

PATHNAME :  [.BENCH.A.C.I1.P2]shell0.sh
            ("NIX equivalent : bench/a/c/i1/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction statements.  The model simulated is an entity consisting
           of a function declaration and an architecture consisting of a
           function call.  The function consists of a number of integer
           variable declarations and a subtraction statement for each variable.
           The factor to be varied is the number of variable declarations/
           subtraction statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable subtraction statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalen  : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function subtract return boolean is
--            variable var1 : integer := 0;
--            variable var2 : integer := 0;
--            variable var3 : integer := 0;
--          begin
--            var1 := var1 -  1;
--            var2 := var2 -  1;
--            var3 := var3 -  1;
--            return true;
--          end subtract;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= subtract;
--        end test;


entity test is
```

```
  function subtract return boolean is
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ -  1;]
    return true;
  end subtract;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= subtract;
end test;
```

```
TEST NUMBER :  95

PATHNAME :  [.BENCH.A.C.I1.P2]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction operations in a function; determine the number of
           subtraction operations allowed in one variable assignment
           statement of a function.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call; the function consists of an integer variable
           declaration and a variable assignment statement containing a number
           of subtraction operations.  The factor to be varied is the number of
           subtraction operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable subtraction operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function subtract return boolean is
--            variable var : integer := 0;
--          begin
--            var := var
--                       - var
--                       - var
--                       - var;
--            return true;
--          end subtract;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= subtract;
--        end test;
```

```
entity test is
  function subtract return boolean is
    variable var : integer := 0;
  begin
    var := var #1[- var];
    return true;
  end subtract;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= subtract;
end test;
```

```
TEST NUMBER :  96

PATHNAME :  [.BENCH.A.C.H1.P2]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of integer
           variable declarations and a subtraction statement for each variable.
           The factor to be varied is the number of variable declarations/
           subtraction statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable subtraction statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure subtract is
--            variable var1 : integer := 0;
--            variable var2 : integer := 0;
--            variable var3 : integer := 0;
--          begin
--            var1 := var1 -  1;
--            var2 := var2 -  1;
--            var3 := var3 -  1;
--          end subtract;
--        end test;
--        architecture test of test is
--        begin
--          subtract;
--        end test;



entity test is
  procedure subtract is
```

```
#1[    variable var@ : integer := 0;]
  begin
#1[    var@ := var@ -  1;]
  end subtract;
end test;
architecture test of test is
begin
  subtract;
end test;
```

```
TEST NUMBER :  97

PATHNAME :  [.BENCH.A.C.H1.P2]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction operations in a procedure; determine the number of
           subtraction operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an entity consisting of a
           procedure declaration and an architecture consisting of a procedure
           call; the procedure consists of an integer variable declaration and
           a variable assignment statement containing a number of subtraction
           operations.  The factor to be varied is the number of subtraction
           operations contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable subtraction operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure subtract is
--            variable var : integer := 0;
--          begin
--            var := var
--                       - var
--                       - var
--                       - var;
--          end subtract;
--        end test;
--        architecture test of test is
--        begin
--          subtract;
--        end test;


entity test is
```

```
  procedure subtract is
    variable var : integer := 0;
  begin
    var := var #1[~ var];
  end subtract;
end test;
architecture test of test is
begin
  subtract;
end test;
```

```
TEST NUMBER :  98

PATHNAME :  [.BENCH.A.C.H1.P3]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication statements.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call.  The procedure consists of a number of integer
           variable declarations and a multiplication statement for each
           variable.  The factor to be varied is the number of variable
           declarations/multiplication statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable multiplication statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure multiply is
--            variable var1 : integer := 5;
--            variable var2 : integer := 5;
--            variable var3 : integer := 5;
--          begin
--            var1 := var1 *  2;
--            var2 := var2 *  2;
--            var3 := var3 *  2;
--          end multiply;
--        end test;
--        architecture test of test is
--        begin
--          multiply;
--        end test;


entity test is
  procedure multiply is
```

```
#1[    variable var@ : integer := 5;]
  begin
#1[    var@ := var@ *  2;]
  end multiply;
end test;
architecture test of test is
begin
  multiply;
end test;
```

```
TEST NUMBER :  99

PATHNAME :  [.BENCH.A.C.H1.P3]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication operations in a procedure; determine the number of
           multiplication operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of an integer variable
           declaration and a variable assignment statement containing a number
           of multiplication operations.  The factor to be varied is the number
           of multiplication operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable multiplication operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure multiply is
--            variable var : integer := 1;
--          begin
--            var := var
--                       * var
--                       * var
--                       * var;
--          end multiply;
--        end test;
--        architecture test of test is
--        begin
--          multiply;
--        end test;
```

```
entity test is
  procedure multiply is
    variable var : integer := 1;
  begin
    var := var
#1[               * var];
  end multiply;
end test;
architecture test of test is
begin
  multiply;
end test;
```

```
TEST NUMBER :  100

PATHNAME :  [.BENCH.A.C.I1.P3]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of integer
           variable declarations and a multiplication statement for each
           variable.  The factor to be varied is the number of variable
           declarations/multiplication statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable multiplication statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function multiply return boolean is
--            variable var1 : integer := 5;
--            variable var2 : integer := 5;
--            variable var3 : integer := 5;
--          begin
--            var1 := var1 *  2;
--            var2 := var2 *  2;
--            var3 := var3 *  2;
--            return true;
--          end multiply;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= multiply;
--        end test;


entity test is
```

```
  function multiply return boolean is
#1[    variable var@ : integer := 5;]
  begin
#1[    var@ := var@ *  2;]
    return true;
  end multiply;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= multiply;
end test;
```

```
TEST NUMBER :  101

PATHNAME :  [.BENCH.A.C.I1.P3]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication operations in a function; determine the number of
           multiplication operations allowed in one variable assignment
           statement of a function.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call; the function consists of an integer variable
           declaration and a variable assignment statement containing a number
           of multiplication operations.  The factor to be varied is the number
           of multiplication operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable multiplication operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function multiply return boolean is
--            variable var : integer := 1;
--          begin
--            var := var
--                       * var
--                       * var
--                       * var;
--            return true;
--          end multiply;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= multiply;
--        end test;
```

```
entity test is
  function multiply return boolean is
    variable var : integer := 1;
  begin
    var := var
#1[               * var];
    return true;
  end multiply;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= multiply;
end test;
```

```
TEST NUMBER :  102

PATHNAME :  [.BENCH.A.C.H1.P4]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p4/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division statements.  The model simulated is an entity consisting of
           a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of integer
           variable declarations and a division statement for each variable.
           The factor to be varied is the number of variable declarations/
           division statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable division statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure divide is
--            variable var1 : integer := 10;
--            variable var2 : integer := 10;
--            variable var3 : integer := 10;
--          begin
--            var1 := var1 /  2;
--            var2 := var2 /  2;
--            var3 := var3 /  2;
--          end divide;
--        end test;
--        architecture test of test is
--        begin
--          divide;
--        end test;



entity test is
  procedure divide is
```

```
#1[    variable var@ : integer := 10;]
  begin
#1[    var@ := var@ /  2;]
  end divide;
end test;
architecture test of test is
begin
  divide;
end test;
```

```
TEST NUMBER :  103

PATHNAME :  [.BENCH.A.C.H1.P4]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division operations in a procedure; determine the number of
           division operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of an integer variable
           declaration and a variable assignment statement containing a number
           of division operations.  The factor to be varied is the number
           of division operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable division operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure divide is
--            variable var : integer := 1;
--          begin
--            var := var
--                       / var
--                       / var
--                       / var;
--          end divide;
--        end test;
--        architecture test of test is
--        begin
--          divide;
--        end test;
```

```
entity test is
  procedure divide is
    variable var : integer := 1;
  begin
    var := var
#1[               / var];
  end divide;
end test;
architecture test of test is
begin
  divide;
end test;
```

```
TEST NUMBER :  104

PATHNAME :  [.BENCH.A.C.I1.P4]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p4/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division statements.  The model simulated is an entity consisting of
           a function declaration and an architecture consisting of a function
           call.  The function consists of a number of integer variable
           declarations and a division statement for each variable.  The
           factor to be varied is the number of variable declarations/division
           statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable division statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function divide return boolean is
--            variable var1 : integer := 10;
--            variable var2 : integer := 10;
--            variable var3 : integer := 10;
--          begin
--            var1 := var1 /  2;
--            var2 := var2 /  2;
--            var3 := var3 /  2;
--            return true;
--          end divide;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= divide;
--        end test;


entity test is
```

```
  function divide return boolean is
#1[    variable var@ : integer := 10;]
  begin
#1[    var@ := var@ /  2;]
    return true;
  end divide;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= divide;
end test;
```

```
TEST NUMBER :  105

PATHNAME :  [.BENCH.A.C.I1.P4]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           division operations in a function; determine the number of
           division operations allowed in one variable assignment statement of
           a function.  The model simulated is an entity consisting of a
           function declaration and an architecture consisting of a function
           call; the function consists of an integer variable declaration and
           a variable assignment statement containing a number of division
           operations.  The factor to be varied is the number of division
           operations contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable division operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function divide return boolean is
--            variable var : integer := 1;
--          begin
--            var := var
--                       / var
--                       / var
--                       / var;
--            return true;
--          end divide;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= divide;
--        end test;
```

```
entity test is
  function divide return boolean is
    variable var : integer := 1;
  begin
    var := var
#1[                / var];
    return true;
  end divide;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= divide;
end test;
```

```
TEST NUMBER :  106

PATHNAME :  [.BENCH.A.C.I1.P5]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p5/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of string
           variable declarations and a concatenation statement for each
           variable.  The factors to be varied are the number of variable
           declarations/concatenation statements in the function and the length
           of the string used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable declarations/variable concatenation statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function concatenate return boolean is
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--            variable var1 : string(1 to (10 + 10));
--            variable var2 : string(1 to (10 + 10));
--            variable var3 : string(1 to (10 + 10));
--          begin
--            var1 := static_var & static_var;
--            var2 := static_var & static_var;
--            var3 := static_var & static_var;
--            return true;
--          end concatenate;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= concatenate;
--        end test;
```

```
entity test is
  function concatenate return boolean is
    variable static_var : string(1 to %1%) := "?1?";
#2[    variable var@ : string(1 to (%1% + %1%));]
  begin
#2[    var@ := static_var & static_var;]
    return true;
  end concatenate;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= concatenate;
end test;
```

```
TEST NUMBER :  107

PATHNAME :  [.BENCH.A.C.I1.P5]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p5/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation operations in a function; determine the number of
           concatenation operations allowed in one variable assignment
           statement of a function.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call; the function consists of two string variable
           declarations and a variable assignment statement containing a number
           of concatenation operations.  The factors to be varied are the
           number of concatenation operations contained in the variable
           assignment statement and the length of the string used in the
           concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable concatenation operations in one statement in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function concatenate return boolean is
--            variable var : string(1 to 10) := "ABCDEFGHIJ";
--            variable result_var : string(1 to (10 * (3 + 1)));
--          begin
--            result_var := var
--                                & var
--                                & var
--                                & var;
--            return true;
--          end concatenate;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
```

```
--          done <= concatenate;
--        end test;


entity test is
  function concatenate return boolean is
    variable var : string(1 to %1%) := "?1?";
    variable result_var : string(1 to (%1% * (%2% + 1)));
  begin
    result_var := var
#2[                        & var];
    return true;
  end concatenate;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= concatenate;
end test;
```

```
TEST NUMBER :  108

PATHNAME :  [.BENCH.A.C.H1.P5]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p5/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation statements.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call.  The procedure consists of a number of string
           variable declarations and a concatenation statement for each
           variable.  The factors to be varied are the number of variable
           declarations/concatenation statements in the procedure and the
           length of the strings used in the concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable declarations/variable concatenation statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure concatenate is
--            variable var1 : string(1 to (10 + 10));
--            variable var2 : string(1 to (10 + 10));
--            variable var3 : string(1 to (10 + 10));
--            variable static_var : string(1 to 10) := "ABCDEFGHIJ";
--          begin
--            var1 := static_var & static_var;
--            var2 := static_var & static_var;
--            var3 := static_var & static_var;
--          end concatenate;
--        end test;
--        architecture test of test is
--        begin
--          concatenate;
--        end test;
```

```
entity test is
  procedure concatenate is
#2[    variable var@ : string(1 to (%1% + %1%));]
    variable static_var : string(1 to %1%) := "?1?";
  begin
#2[    var@ := static_var & static_var;]
  end concatenate;
end test;
architecture test of test is
begin
  concatenate;
end test;
```

```
TEST NUMBER :  109

PATHNAME :  [.BENCH.A.C.H1.P5]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p5/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           concatenation operations in a procedure; determine the number of
           concatenation operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of a string variable
           declaration and a variable assignment statement containing a number
           of concatenation operations.  The factors to be varied are the
           number of concatenation operations contained in the variable
           assignment statement and the length of the string used in the
           concatenations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of strings
--    2 : number of variable concatenation operations in one statement in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",10,3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure concatenate is
--            variable result_var : string(1 to (10 * (3 + 1)));
--            variable var : string(1 to 10) := "ABCDEFGHIJ";
--          begin
--            result_var := var
--                                & var
--                                & var
--                                & var;
--          end concatenate;
--        end test;
--        architecture test of test is
--        begin
--          concatenate;
--        end test;
```

```
entity test is
  procedure concatenate is
    variable result_var : string(1 to (%1% * (%2% + 1)));
    variable var : string(1 to %1%) := "?1?";
  begin
    result_var := var
#2[                        & var];
  end concatenate;
end test;
architecture test of test is
begin
  concatenate;
end test;
```

```
TEST NUMBER :  110

PATHNAME :  [.BENCH.A.C.H1.P6]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of variable
           declarations and a logical AND statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           AND statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/AND statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_and is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 AND var1;
--            var2 := var2 AND var2;
--            var3 := var3 AND var3;
--          end bool_and;
--        end test;
--        architecture test of test is
--        begin
--          bool_and;
--        end test;



entity test is
  procedure bool_and is
#1[     variable var@ : bit := '$2$0$1$';]
```

```
  begin
#1[     var@ := var@ AND var@;]
  end bool_and;
end test;
architecture test of test is
begin
  bool_and;
end test;
```

```
TEST NUMBER :  111

PATHNAME :  [.BENCH.A.C.H1.P6]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND operations in a procedure; determine the number of
           logical AND operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of a variable
           declaration and a variable assignment statement containing a number
           of logical AND operations.  The factor to be varied is the number
           of logical AND operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable AND operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_and is
--            variable var : bit;
--          begin
--            var := var
--                        AND var
--                        AND var
--                        AND var;
--          end bool_and;
--        end test;
--        architecture test of test is
--        begin
--          bool_and;
--        end test;


entity test is
```

```
  procedure bool_and is
    variable var : bit;
  begin
    var := var
#1[                AND var];
  end bool_and;
end test;
architecture test of test is
begin
  bool_and;
end test;
```

```
TEST NUMBER :  112

PATHNAME :  [.BENCH.A.C.H2.P6]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of variable declarations and a
           logical AND statement for each variable.  The factor to be varied is
           the number of variable declarations/logical AND statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable AND statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_and is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 AND var1;
--            var2 := var2 AND var2;
--            var3 := var3 AND var3;
--          end bool_and;
--        begin
--          bool_and;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_and is
```

```
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ AND var@;]
  end bool_and;
begin
  bool_and;
end test;
```

```
TEST NUMBER :  113

PATHNAME :  [.BENCH.A.C.H2.P6]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND operations in a procedure; determine the number of
           logical AND operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call; the procedure
           consists of a variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of logical AND operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable AND operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_and is
--            variable var : bit;
--          begin
--            var := var
--                        AND var
--                        AND var
--                        AND var;
--          end bool_and;
--        begin
--          bool_and;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_and is
```

```
    variable var : bit;
  begin
    var := var
#1[                AND var];
  end bool_and;
begin
  bool_and;
end test;
```

```
TEST NUMBER :  114

PATHNAME :  [.BENCH.A.C.I1.P6]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND statements.  The model simulated is an entity consisting
           of a function declaration and an architecture consisting of a
           function call.  The function consists of a number of variable
           declarations and a logical AND statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           AND statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable AND statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_and return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 AND var1;
--            var2 := var2 AND var2;
--            var3 := var3 AND var3;
--            return true;
--          end bool_and;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_and;
--        end test;


entity test is
```

```
  function bool_and return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ AND var@;]
    return true;
  end bool_and;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_and;
end test;
```

```
TEST NUMBER :  115

PATHNAME :  [.BENCH.A.C.I1.P6]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND operations in a function; determine the number of
           logical AND operations allowed in one variable assignment statement
           of a function.  The model simulated is an entity consisting of a
           function declaration and an architecture consisting of a function
           call; the function consists of a variable declaration and a variable
           assignment statement containing a number of logical AND operations.
           The factor to be varied is the number of logical AND operations
           contained in the variable assignment statement.

EXPECTED RESULTS

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable AND operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_and return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       AND var
--                       AND var
--                       AND var;
--            return true;
--          end bool_and;
--        end test,
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_and;
--        end test;


entity test is
```

```
  function bool_and return boolean is
    variable var : bit;
  begin
    var := var
#1[                AND var];
    return true;
  end bool_and;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_and;
end test;
```

```
TEST NUMBER :  116

PATHNAME :  [.BENCH.A.C.I2.P6]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of variable declarations and a logical
           AND statement for each variable.  The factor to be varied is the
           number of variable declarations/logical AND statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable AND statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_and return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 AND var1;
--            var2 := var2 AND var2;
--            var3 := var3 AND var3;
--            return true;
--          end bool_and;
--          signal done : boolean := false;
--        begin
--          done <= bool_and;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_and return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ AND var@;]
    return true;
  end bool_and;
  signal done : boolean := false;
begin
  done <= bool_and;
end test;
```

```
TEST NUMBER :  117

PATHNAME :  [.BENCH.A.C.I2.P6]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical AND operations in a function; determine the number of
           logical AND operations allowed in one variable assignment statement
           of a function.  The model simulated is an architecture consisting of
           a function declaration and a function call; the function consists of
           a variable declaration and a variable assignment statement
           containing a number of logical AND operations.  The factor to be
           varied is the number of logical AND operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable AND operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_and return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       AND var
--                       AND var
--                       AND var;
--            return true;
--          end bool_and;
--          signal done : boolean := false;
--        begin
--          done <= bool_and;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_and return boolean is
    variable var : bit;
  begin
    var := var
#1[                AND var];
    return true;
  end bool_and;
  signal done : boolean := false;
begin
  done <= bool_and;
end test;
```

```
TEST NUMBER :  118

PATHNAME :  [.BENCH.A.C.H1.P7]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of variable
           declarations and a logical OR statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           OR statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/OR statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_or is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 OR var1;
--            var2 := var2 OR var2;
--            var3 := var3 OR var3;
--          end bool_or;
--        end test;
--        architecture test of test is
--        begin
--          bool_or;
--        end test;



entity test is
  procedure bool_or is
#1[     variable var@ : bit := '$2$0$1$';]
```

```
  begin
#1[     var@ := var@ OR var@;]
  end bool_or;
end test;
architecture test of test is
begin
  bool_or;
end test;
```

```
TEST NUMBER :  119

PATHNAME :  [.BENCH.A.C.H1.P7]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR operations in a procedure; determine the number of
           logical OR operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of a variable
           declaration and a variable assignment statement containing a number
           of logical OR operations.  The factor to be varied is the number
           of logical OR operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable OR operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_or is
--            variable var : bit;
--          begin
--            var := var
--                        OR var
--                        OR var
--                        OR var;
--          end bool_or;
--        end test;
--        architecture test of test is
--        begin
--          bool_or;
--        end test;



entity test is
```

```
  procedure bool_or is
    variable var : bit;
  begin
    var := var
#1[               OR var];
  end bool_or;
end test;
architecture test of test is
begin
  bool_or;
end test;
```

```
TEST NUMBER :  120

PATHNAME :  [.BENCH.A.C.H2.P7]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of variable declarations and a
           logical OR statement for each variable.  The factor to be varied is
           the number of variable declarations/logical OR statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable OR statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_or is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 OR var1;
--            var2 := var2 OR var2;
--            var3 := var3 OR var3;
--          end bool_or;
--        begin
--          bool_or;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_or is
```

```
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ OR var@;]
  end bool_or;
begin
  bool_or;
end test;
```

```
TEST NUMBER :  121

PATHNAME :  [.BENCH.A.C.H2.P7]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR operations in a procedure; determine the number of
           logical OR operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call; the procedure
           consists of a variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of logical OR operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable OR operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_or is
--            variable var : bit;
--          begin
--            var := var
--                        OR var
--                        OR var
--                        OR var;
--          end bool_or;
--        begin
--          bool_or;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_or is
```

```
    variable var : bit;
  begin
    var := var
#1[                OR var];
  end bool_or;
begin
  bool_or;
end test;
```

```
TEST NUMBER :  122

PATHNAME :  [.BENCH.A.C.I1.P7]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR statements.  The model simulated is an entity consisting
           of a function declaration and an architecture consisting of a
           function call.  The function consists of a number of variable
           declarations and a logical OR statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           OR statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable OR statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_or return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 OR var1;
--            var2 := var2 OR var2;
--            var3 := var3 OR var3;
--            return true;
--          end bool_or;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_or;
--        end test;


entity test is
```

```
  function bool_or return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ OR var@;]
    return true;
  end bool_or;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_or;
end test;
```

```
TEST NUMBER :  123

PATHNAME :  [.BENCH.A.C.I1.P7]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR operations in a function; determine the number of
           logical OR operations allowed in one variable assignment statement
           of a function.  The model simulated is an entity consisting of a
           function declara ion and an architecture consisting of a function
           call; the functi›n consists of a variable declaration and a variable
           assignment statement containing a number of logical OR operations.
           The factor to be varied is the number of logical OR operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable OR operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_or return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       OR var
--                       OR var
--                       OR var;
--            return true;
--          end bool_or;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_or;
--        end test;


entity test is
```

```
  function bool_or return boolean is
    variable var : bit;
  begin
    var := var
#1[                OR var];
    return true;
  end bool_or;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_or;
end test;
```

```
TEST NUMBER :  124

PATHNAME :  [.BENCH.A.C.I2.P7]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of variable declarations and a logical
           OR statement for each variable.  The factor to be varied is the
           number of variable declarations/logical OR statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable OR statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_or return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 OR var1;
--            var2 := var2 OR var2;
--            var3 := var3 OR var3;
--            return true;
--          end bool_or;
--          signal done : boolean := false;
--        begin
--          done <= bool_or;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_or return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ OR var@;]
    return true;
  end bool_or;
  signal done : boolean := false;
begin
  done <= bool_or;
end test;
```

```
TEST NUMBER :  125

PATHNAME :  [.BENCH.A.C.I2.P7]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical OR operations in a function; determine the number of
           logical OR operations allowed in one variable assignment statement
           of a function.  The model simulated is an architecture consisting of
           a function declaration and a function call; the function consists of
           a variable declaration and a variable assignment statement
           containing a number of logical OR operations.  The factor to be
           varied is the number of logical OR operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable OR operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_or return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       OR var
--                       OR var
--                       OR var;
--            return true;
--          end bool_or;
--          signal done : boolean := false;
--        begin
--          done <= bool_or;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_or return boolean is
    variable var : bit;
  begin
    var := var
#1[                OR var];
    return true;
  end bool_or;
  signal done : boolean := false;
begin
  done <= bool_or;
end test;
```

```
TEST NUMBER :  126

PATHNAME :  [.BENCH.A.C.H1.P8]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of variable
           declarations and a logical NAND statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           NAND statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/NAND statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_nand is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NAND var1;
--            var2 := var2 NAND var2;
--            var3 := var3 NAND var3;
--          end bool_nand;
--        end test;
--        architecture test of test is
--        begin
--          bool_nand;
--        end test;



entity test is
  procedure bool_nand is
#1[    variable var@ : bit := '$2$0$1$';]
```

```
  begin
#1[    var@ := var@ NAND var@;]
  end bool_nand;
end test;
architecture test of test is
begin
  bool_nand;
end test;
```

```
TEST NUMBER :  127

PATHNAME :  [.BENCH.A.C.H1.P8]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p8/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND operations in a procedure; determine the number of
           logical NAND operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of a variable
           declaration and a variable assignment statement containing a number
           of logical NAND operations.  The factor to be varied is the number
           of logical NAND operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 15 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NAND operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_nand is
--            variable var : bit;
--          begin
--            var := var
--                       NAND (var
--                       NAND (var
--                       NAND (var
--                                )
--                                )
--                                );
--          end bool_nand;
--        end test;
--        architecture test of test is
--        begin
--          bool_nand;
--        end test;
```

```
entity test is
  procedure bool_nand is
    variable var : bit;
  begin
    var := var
#1[                NAND (var]
#1[                         )];
  end bool_nand;
end test;
architecture test of test is
begin
  bool_nand;
end test;
```

```
TEST NUMBER :  128

PATHNAME :  [.BENCH.A.C.H2.P8]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of variable declarations and a
           logical NAND statement for each variable.  The factor to be varied is
           the number of variable declarations/logical NAND statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NAND statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_nand is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NAND var1;
--            var2 := var2 NAND var2;
--            var3 := var3 NAND var3;
--          end bool_nand;
--        begin
--          bool_nand;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_nand is
```

```
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NAND var@;]
  end bool_nand;
begin
  bool_nand;
end test;
```

```
TEST NUMBER :  129

PATHNAME :  [.BENCH.A.C.H2.P8]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p8/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND operations in a procedure; determine the number of
           logical NAND operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call; the procedure
           consists of a variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of logical NAND operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NAND operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_nand is
--            variable var : bit;
--          begin
--            var := var
--                       NAND (var
--                       NAND (var
--                       NAND (var
--                                )
--                                )
--                                );
--          end bool_nand;
--        begin
--          bool_nand;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure bool_nand is
    variable var : bit;
  begin
    var := var
#1[                NAND (var]
#1[                         )];
  end bool_nand;
begin
  bool_nand;
end test;
```

```
TEST NUMBER :  130

PATHNAME :  [ BENCH.A.C.I1.P8]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of variable
           declarations and a logical NAND statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           NAND statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NAND statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_nand return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NAND var1;
--            var2 := var2 NAND var2;
--            var3 := var3 NAND var3;
--            return true;
--          end bool_nand;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_nand;
--        end test;
```

```
entity test is
  function bool_nand return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NAND var@;]
    return true;
  end bool_nand;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_nand;
end test;
```

TEST NUMBER : 131

PATHNAME : [.BENCH.A.C.I1.P8]shell1.sh
(UNIX equivalent : bench/a/c/i1/p8/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute variable logical NAND operations in a function; determine the number of logical NAND operations allowed in one variable assignment statement of a function. The model simulated is an entity consisting of a function declaration and an architecture consisting of a function call; the function consists of a variable declaration and a variable assignment statement containing a number of logical NAND operations. The factor to be varied is the number of logical NAND operations contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NAND operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_nand return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       NAND (var
--                       NAND (var
--                       NAND (var
--                                )
--                                )
--                                );
--            return true;
--          end bool_nand;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_nand;
--        end test;
```

```
entity test is
  function bool_nand return boolean is
    variable var : bit;
  begin
    var := var
#1[                NAND (var]
#1[                         )];
    return true;
  end bool_nand;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_nand;
end test;
```

```
TEST NUMBER :  132

PATHNAME :  [.BENCH.A.C.I2.P8]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of variable declarations and a logical
           NAND statement for each variable.  The factor to be varied is the
           number of variable declarations/logical NAND statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NAND statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_nand return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NAND var1;
--            var2 := var2 NAND var2;
--            var3 := var3 NAND var3;
--            return true;
--          end bool_nand;
--          signal done : boolean := false;
--        begin
--          done <= bool_nand;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_nand return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NAND var@;]
    return true;
  end bool_nand;
  signal done : boolean := false;
begin
  done <= bool_nand;
end test;
```

```
TEST NUMBER :  133

PATHNAME :  [.BENCH.A.C.I2.P8]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p8/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NAND operations in a function; determine the number of
           logical NAND operations allowed in one variable assignment statement
           of a function.  The model simulated is an architecture consisting of
           a function declaration and a function call; the function consists of
           a variable declaration and a variable assignment statement
           containing a number of logical NAND operations.  The factor to be
           varied is the number of logical NAND operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NAND operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_nand return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       NAND (var
--                       NAND (var
--                       NAND (var
--                                )
--                                )
--                                );
--            return true;
--          end bool_nand;
--          signal done : boolean := false;
--        begin
--          done <= bool_nand;
--        end test;
```

```
entity test is end;
architecture test of test is
  function bool_nand return boolean is
    variable var : bit;
  begin
    var := var
#1[               NAND (var]
#1[                        )];
    return true;
  end bool_nand;
  signal done : boolean := false;
begin
  done <= bool_nand;
end test;
```

```
TEST NUMBER :  134

PATHNAME :  [.BENCH.A.C.H1.P9]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of variable
           declarations and a logical NOR statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           NOR statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/NOR statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_nor is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NOR var1;
--            var2 := var2 NOR var2;
--            var3 := var3 NOR var3;
--          end bool_nor;
--        end test;
--        architecture test of test is
--        begin
--          bool_nor;
--        end test;



entity test is
  procedure bool_nor is
#1[    variable var@ : bit := '$2$0$1$';]
```

```
  begin
#1[    var© := var© NOR var©;]
  end bool_nor;
end test;
architecture test of test is
begin
  bool_nor;
end test;
```

```
TEST NUMBER :  135

PATHNAME :  [.BENCH.A.C.H1.P9]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR operations in a procedure; determine the number of
           logical NOR operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of a variable
           declaration and a variable assignment statement containing a number
           of logical NOR operations.  The factor to be varied is the number
           of logical NOR operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOR operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_nor is
--            variable var : bit;
--          begin
--            var := var
--                        NOR (var
--                        NOR (var
--                        NOR (var
--                                  )
--                                  )
--                                  );
--          end bool_nor;
--        end test;
--        architecture test of test is
--        begin
--          bool_nor;
--        end test;
```

```
entity test is
  procedure bool_nor is
    variable var : bit;
  begin
    var := var
#1[                NOR (var]
#1[                        )];
  end bool_nor;
end test;
architecture test of test is
begin
  bool_nor;
end test;
```

```
TEST NUMBER :  136

PATHNAME :  [.BENCH.A.C.H2.P9]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of variable declarations and a
           logical NOR statement for each variable.  The factor to be varied is
           the number of variable declarations/logical NOR statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NOR statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_nor is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NOR var1;
--            var2 := var2 NOR var2;
--            var3 := var3 NOR var3;
--          end bool_nor;
--        begin
--          bool_nor;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_nor is
```

```
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NOR var@;]
  end bool_nor;
begin
  bool_nor;
end test;
```

```
TEST NUMBER :  137

PATHNAME :  [.BENCH.A.C.H2.P9]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR operations in a procedure; determine the number of
           logical NOR operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call; the procedure
           consists of a variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of logical NOR operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOR operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_nor is
--            variable var : bit;
--          begin
--            var := var
--                       NOR (var
--                       NOR (var
--                       NOR (var
--                                )
--                                )
--                                );
--          end bool_nor;
--        begin
--          bool_nor;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure bool_nor is
    variable var : bit;
  begin
    var := var
#1[                NOR (var]
#1[                        )];
  end bool_nor;
begin
  bool_nor;
end test;
```

```
TEST NUMBER :  138

PATHNAME :  [.BENCH.A.C.I1.P9]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of variable
           declarations and a logical NOR statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           NOR statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NOR statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_nor return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NOR var1;
--            var2 := var2 NOR var2;
--            var3 := var3 NOR var3;
--            return true;
--          end bool_nor;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_nor;
--        end test;


entity test is
```

```
  function bool_nor return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NOR var@;]
    return true;
  end bool_nor;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_nor;
end test;
```

```
TEST NUMBER :  139

PATHNAME :  [.BENCH.A.C.I1.P9]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR operations in a function; determine the number of
           logical NOR operations allowed in one variable assignment statement
           of a function.  The model simulated is an entity consisting of a
           function declaration and an architecture consisting of a function
           call; the function consists of a variable declaration and a variable
           assignment statement containing a number of logical NOR operations.
           The factor to be varied is the number of logical NOR operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOR operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--       entity test is
--         function bool_nor return boolean is
--           variable var : bit;
--         begin
--           var := var
--                      NOR (var
--                      NOR (var
--                      NOR (var
--                               )
--                               )
--                               );
--           return true;
--         end bool_nor;
--       end test;
--       architecture test of test is
--         signal done : boolean := false;
--       begin
--         done <= bool_nor;
--       end test;
```

```
entity test is
  function bool_nor return boolean is
    variable var : bit;
  begin
    var := var
#1[               NOR (var]
#1[                       )];
    return true;
  end bool_nor;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_nor;
end test;
```

```
TEST NUMBER :  140

PATHNAME :  [.BENCH.A.C.I2.P9]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of variable declarations and a logical
           NOR statement for each variable.  The factor to be varied is the
           number of variable declarations/logical NOR statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NOR statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_nor return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 NOR var1;
--            var2 := var2 NOR var2;
--            var3 := var3 NOR var3;
--            return true;
--          end bool_nor;
--          signal done : boolean := false;
--        begin
--          done <= bool_nor;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_nor return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ NOR var@;]
    return true;
  end bool_nor;
  signal done : boolean := false;
begin
  done <= bool_nor;
end test;
```

```
TEST NUMBER :  141

PATHNAME :  [.BENCH.A.C.I2.P9]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOR operations in a function; determine the number of
           logical NOR operations allowed in one variable assignment statement
           of a function.  The model simulated is an architecture consisting of
           a function declaration and a function call; the function consists of
           a variable declaration and a variable assignment statement
           containing a number of logical NOR operations.  The factor to be
           varied is the number of logical NOR operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 16 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOR operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_nor return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       NOR (var
--                       NOR (var
--                       NOR (var
--                                )
--                                )
--                                );
--            return true;
--          end bool_nor;
--          signal done : boolean := false;
--        begin
--          done <= bool_nor;
--        end test;
```

```
.tity test is end;
architecture test of test is
  function bool_nor return boolean is
    variable var : bit;
  begin
    var := var
#1[               NOR (var]
#1[                        )];
    return true;
  end bool_nor;
  signal done : boolean := false;
begin
  done <= bool_nor;
end test;
```

```
TEST NUMBER :  142

PATHNAME :  [.BENCH.A.C.H1.P10]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p10/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of variable
           declarations and a logical XOR statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           XOR statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/XOR statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\" \,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_xor is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 XOR var1;
--            var2 := var2 XOR var2;
--            var3 := var3 XOR var3;
--          end bool_xor;
--        end test;
--        architecture test of test is
--        begin
--          bool_xor;
--        end test;



entity test is
  procedure bool_xor is
#1[    variable var0 : bit :  '0';]
```

```
  begin
#1[    var@ := var@ XOR var@;]
  end bool_xor;
end test;
architecture test of test is
begin
  bool_xor;
end test;
```

```
TEST NUMBER :  143

PATHNAME :  [.BENCH.A.C.H1.P10]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p10/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR operations in a procedure; determine the number of
           logical XOR operations allowed in one variable assignment
           statement of a procedure.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call; the procedure consists of a variable
           declaration and a variable assignment statement containing a number
           of logical XOR operations.  The factor to be varied is the number
           of logical XOR operations contained in the variable assignment
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable XOR operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_xor is
--            variable var : bit;
--          begin
--            var := var
--                       XOR (var
--                       XOR (var
--                       XOR (var
--                               )
--                               )
--                               );
--          end bool_xor;
--        end test;
--        architecture test of test is
--        begin
--          bool_xor;
--        end test;
```

```
entity test is
  procedure bool_xor is
    variable var : bit;
  begin
    var := var
#1[                XOR (var]
#1[                        )];
  end bool_xor;
end test;
architecture test of test is
begin
  bool_xor;
end test;
```

```
TEST NUMBER :  144

PATHNAME :  [.BENCH.A.C.H2.P10]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p10/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of variable declarations and a
           logical XOR statement for each variable.  The factor to be varied is
           the number of variable declarations/logical XOR statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable XOR statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_xor is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 XOR var1;
--            var2 := var2 XOR var2;
--            var3 := var3 XOR var3;
--          end bool_xor;
--        begin
--          bool_xor;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_xor is
```

```
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ XOR var@;]
  end bool_xor;
begin
  bool_xor;
end test;
```

```
TEST NUMBER :  145

PATHNAME :  [.BENCH.A.C.H2.P10]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p10/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR operations in a procedure; determine the number of
           logical XOR operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call; the procedure
           consists of a variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of logical XOR operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable XOR operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_xor is
--            variable var : bit;
--          begin
--            var := var
--                       XOR (var
--                       XOR (var
--                       XOR (var
--                               )
--                               )
--                               );
--          end bool_xor;
--        begin
--          bool_xor;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure bool_xor is
    variable var : bit;
  begin
    var := var
#1[                XOR (var]
#1[                        )];
  end bool_xor;
begin
  bool_xor;
end test;
```

```
TEST NUMBER :  146

PATHNAME :  [.BENCH.A.C.I1.P10]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p10/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of variable
           declarations and a logical XOR statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           XOR statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable XOR statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_xor return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 XOR var1;
--            var2 := var2 XOR var2;
--            var3 := var3 XOR var3;
--            return true;
--          end bool_xor;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_xor;
--        end test;


entity test is
```

```
  function bool_xor return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ XOR var@;]
    return true;
  end bool_xor;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_xor;
end test;
```

```
TEST NUMBER :  147

PATHNAME :  [.BENCH.A.C.I1.P10]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p10/shell1.sh)

PURPOSE :   Determine the simulation CPU time required to execute variable
            logical XOR operations in a function; determine the number of
            logical XOR operations allowed in one variable assignment statement
            of a function.  The model simulated is an entity consisting of a
            function declaration and an architecture consisting of a function
            call; the function consists of a variable declaration and a variable
            assignment statement containing a number of logical XOR operations.
            The factor to be varied is the number of logical XOR operations
            contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable XOR operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_xor return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       XOR (var
--                       XOR (var
--                       XOR (var
--                               )
--                               )
--                               );
--            return true;
--          end bool_xor;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_xor;
--        end test;
```

```
entity test is
  function bool_xor return boolean is
    variable var : bit;
  begin
    var := var
#1[                XOR (var]
#1[                        )];
    return true;
  end bool_xor;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_xor;
end test;
```

```
TEST NUMBER :  148

PATHNAME :  [.BENCH.A.C.I2.P10]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p10/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of variable declarations and a logical
           XOR statement for each variable.  The factor to be varied is the
           number of variable declarations/logical XOR statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable XOR statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_xor return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := var1 XOR var1;
--            var2 := var2 XOR var2;
--            var3 := var3 XOR var3;
--            return true;
--          end bool_xor;
--          signal done : boolean := false;
--        begin
--          done <= bool_xor;
--        end test;


entity test is end;
architecture test of test is
```

```
  function bool_xor return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := var@ XOR var@;]
    return true;
  end bool_xor;
  signal done : boolean := false;
begin
  done <= bool_xor;
end test;
```

```
TEST NUMBER :  149

PATHNAME :  [.BENCH.A.C.I2.P10]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p10/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical XOR operations in a function; determine the number of
           logical XOR operations allowed in one variable assignment statement
           of a function.  The model simulated is an architecture consisting of
           a function declaration and a function call; the function consists of
           a variable declaration and a variable assignment statement
           containing a number of logical XOR operations.  The factor to be
           varied is the number of logical XOR operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable XOR operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_xor return boolean is
--            variable var : bit;
--          begin
--            var := var
--                       XOR (var
--                       XOR (var
--                       XOR (var
--                              )
--                              )
--                              );
--            return true;
--          end bool_xor;
--          signal done : boolean := false;
--        begin
--          done <= bool_xor;
--        end test;
```

```
entity test is end;
architecture test of test is
  function bool_xor return boolean is
    variable var : bit;
  begin
    var := var
#1[               XOR (var]
#1[                       )];
    return true;
  end bool_xor;
  signal done : boolean := false;
begin
  done <= bool_xor;
end test;
```

```
TEST NUMBER :  150

PATHNAME :  [.BENCH.A.C.H1.P11]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p11/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of variable
           declarations and a logical NOT statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           NOT statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/NOT statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_not is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := NOT var1;
--            var2 := NOT var2;
--            var3 := NOT var3;
--          end bool_not;
--        end test;
--        architecture test of test is
--        begin
--          bool_not;
--        end test;



entity test is
  procedure bool_not is
#1[    variable var@ : bit := '$2$0$1$';]
```

```
  begin
#1[    var0 := NOT var0;]
  end bool_not;
end test;
architecture test of test is
begin
  bool_not;
end test;
```

TEST NUMBER : 151

PATHNAME : [.BENCH.A.C.H1.P11]shell1.sh
(UNIX equivalent : bench/a/c/h1/p11/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute variable logical NOT operations in a procedure; determine the number of logical NOT operations allowed in one variable assignment statement of a procedure. The model simulated is an entity consisting of a procedure declaration and an architecture consisting of a procedure call; the procedure consists of a variable declaration and a variable assignment statement containing a number of logical NOT operations. The factor to be varied is the number of logical NOT operations contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOT operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure bool_not is
--            variable var : bit;
--          begin
--            var :=
--                   NOT (
--                   NOT (
--                   NOT (
--                        var
--                           )
--                           )
--                           );
--          end bool_not;
--        end test;
--        architecture test of test is
--        begin
--          bool_not;
--        end test;
```

```
entity test is
  procedure bool_not is
    variable var : bit;
  begin
    var :=
#1[           NOT (]
                 var
#1[                    )];
  end bool_not;
end test;
architecture test of test is
begin
  bool_not;
end test;
```

```
TEST NUMBER :  152

PATHNAME :  [.BENCH.A.C.H2.P11]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p11/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of variable declarations and a
           logical NOT statement for each variable.  The factor to be varied is
           the number of variable declarations/logical NOT statements in the
           procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date . 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NOT statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_not is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := NOT var1;
--            var2 := NOT var2;
--            var3 := NOT var3;
--          end bool_not;
--        begin
--          bool_not;
--        end test;



entity test is end;
architecture test of test is
  procedure bool_not is
```

```
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := NOT var@;]
  end bool_not;
begin
  bool_not;
end test;
```

```
TEST NUMBER :  153

PATHNAME :  [.BENCH.A.C.H2.P11]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT operations in a procedure; determine the number of
           logical NOT operations allowed in one variable assignment statement
           of a procedure.  The model simulated is an architecture consisting
           of a procedure declaration and a procedure call; the procedure
           consists of a variable declaration and a variable assignment
           statement containing a number of addition operations.  The factor to
           be varied is the number of logical NOT operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOT operations in one statement in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh", test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure bool_not is
--            variable var : bit;
--          begin
--            var :=
--                   NOT (
--                   NOT (
--                   NOT (
--                        var
--                           )
--                           )
--                           );
--          end bool_not;
--        begin
--          bool_not;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure bool_not is
    variable var : bit;
  begin
    var :=
#1[           NOT (]
                var
#1[                    )];
  end bool_not;
begin
  bool_not;
end test;
```

```
TEST NUMBER :  154

PATHNAME :  [.BENCH.A.C.I1.P11]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p11/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of variable
           declarations and a logical NOT statement for each variable.  The
           factor to be varied is the number of variable declarations/logical
           NOT statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 19 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NOT statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_not return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := NOT var1;
--            var2 := NOT var2;
--            var3 := NOT var3;
--            return true;
--          end bool_not;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_not;
--        end test;


entity test is
```

```
  function bool_not return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := NOT var@;]
    return true;
  end bool_not;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_not;
end test;
```

```
TEST NUMBER :  155

PATHNAME :  [.BENCH.A.C.I1.P11]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT operations in a function; determine the number of
           logical NOT operations allowed in one variable assignment statement
           of a function.  The model simulated is an entity consisting of a
           function declaration and an architecture consisting of a function
           call; the function consists of a variable declaration and a variable
           assignment statement containing a number of logical NOT operations.
           The factor to be varied is the number of logical NOT operations
           contained in the variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 20 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOT operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function bool_not return boolean is
--            variable var : bit;
--          begin
--            var :=
--                   NOT (
--                   NOT (
--                   NOT (
--                        var
--                           )
--                           )
--                           );
--            return true;
--          end bool_not;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= bool_not;
```

```
--        end test;


entity test is
  function bool_not return boolean is
    variable var : bit;
  begin
    var :=
#1[           NOT (]
                var
#1[                    )];
    return true;
  end bool_not;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= bool_not;
end test;
```

```
TEST NUMBER :  156

PATHNAME :  [.BENCH.A.C.I2.P11]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p11/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of variable declarations and a logical
           NOT statement for each variable.  The factor to be varied is the
           number of variable declarations/logical NOT statements in the
           function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 20 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable NOT statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_not return boolean is
--            variable var1 : bit := '0';
--            variable var2 : bit := '1';
--            variable var3 : bit := '0';
--          begin
--            var1 := NOT var1;
--            var2 := NOT var2;
--            var3 := NOT var3;
--            return true;
--          end bool_not;
--          signal done : boolean := false;
--        begin
--          done <= bool_not;
--        end test;


entity test is end,
architecture test of test is
```

```
  function bool_not return boolean is
#1[    variable var@ : bit := '$2$0$1$';]
  begin
#1[    var@ := NOT var@;]
    return true;
  end bool_not;
  signal done : boolean := false;
begin
  done <= bool_not;
end test;
```

```
TEST NUMBER :  157

PATHNAME :  [.BENCH.A.C.I2.P11]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           logical NOT operations in a function; determine the number of
           logical NOT operations allowed in one variable assignment statement
           of a function.  The model simulated is an architecture consisting of
           a function declaration and a function call; the function consists of
           a variable declaration and a variable assignment statement
           containing a number of logical NOT operations.  The factor to be
           varied is the number of logical NOT operations contained in the
           variable assignment statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 20 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable NOT operations in one statement in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function bool_not return boolean is
--            variable var : bit;
--          begin
--            var :=
--                   NOT (
--                   NOT (
--                   NOT (
--                        var
--                           )
--                           )
--                           );
--            return true;
--          end bool_not;
--          signal done : boolean := false;
--        begin
--          done <= bool_not;
--        end test;
```

```
entity test is end;
architecture test of test is
  function bool_not return boolean is
    variable var : bit;
  begin
    var :=
#1[           NOT (]
                var
#1[                    )];
    return true;
  end bool_not;
  signal done : boolean := false;
begin
  done <= bool_not;
end test;
```

```
TEST NUMBER :  158

PATHNAME :  [.BENCH.A.C.P1]shell2.sh
            (UNIX equivalent : bench/a/c/p1/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           operations on signals.  The model simulated consists of a number of
           integer signal declarations and one addition signal assignment
           statement for each signal, plus a signal declaration/signal
           assignment statement/assertion that controls how long to simulate
           the model.  The factors to be varied are the number of signal
           declarations/number of addition signal assignment statements and the
           length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal addition statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := 0;
--          signal sig2 : integer := 0;
--          signal sig3 : integer := 0;
--        begin
--          sig1 <= sig1 + 1 after 1 ns;
--          sig2 <= sig2 + 1 after 1 ns;
--          sig3 <= sig3 + 1 after 1 ns;
--          stop <= '0', '1' after 2 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;



entity test is end;
architecture test of test is
  signal stop : bit := '0';
```

```
#1[  signal sig@ : integer := 0;]
begin
#1[  sig@ <= sig@ + 1 after 1 ns;]
  stop <= '0', '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  159

PATHNAME :  [.BENCH.A.C.P2]shell2.sh
            (UNIX equivalent : bench/a/c/p2/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute subtraction
           operations on signals.  The model simulated consists of a number of
           integer signal declarations and one subtraction signal assignment
           statement for each signal, plus a signal declaration/signal
           assignment statement/assertion that controls how long to simulate
           the model.  The factors to be varied are the number of signal
           declarations/number of subtraction signal assignment statements and
           the length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal subtraction statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := 0;
--          signal sig2 : integer := 0;
--          signal sig3 : integer := 0;
--        begin
--          sig1 <= sig1 - 1 after 1 ns;
--          sig2 <= sig2 - 1 after 1 ns;
--          sig3 <= sig3 - 1 after 1 ns;
--          stop <= '0', '1' after 2 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : integer := 0;]
```

```
begin
#1[  sig0 <= sig0 - 1 after 1 ns;]
  stop <= '0', '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  160

PATHNAME :  [.BENCH.A.C.P3]shell2.sh
            (UNIX equivalent : bench/a/c/p3/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute multiplication
           operations on signals.  The model simulated consists of a number of
           integer signal declarations and one multiplication signal assignment
           statement for each signal, plus a signal declaration/signal
           assignment statement/assertion that controls how long to simulate
           the model.  The factors to be varied are the number of signal
           declarations/number of multiplication signal assignment statements
           and the length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal multiplication statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := 1;
--          signal sig2 : integer := 1;
--          signal sig3 : integer := 1;
--        begin
--          sig1 <= sig1 * 2 after 1 ns;
--          sig2 <= sig2 * 2 after 1 ns;
--          sig3 <= sig3 * 2 after 1 ns;
--          stop <= '0', '1' after 2 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : integer := 1;]
```

```
begin
#1[  sig@ <= sig@ * 2 after 1 ns;]
  stop <= '0', '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

TEST NUMBER : 161

PATHNAME : [.BENCH.A.C.P4]shell2.sh
(UNIX equivalent : bench/a/c/p4/shell2.sh)

PURPOSE : Determine the simulation CPU time required to execute division operations on signals. The model simulated consists of a number of integer signal declarations and one division signal assignment statement for each signal, plus a signal declaration/signal assignment statement/assertion that controls how long to simulate the model. The factors to be varied are the number of signal declarations/number of division signal assignment statements and the length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal division statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := integer'high;
--          signal sig2 : integer := integer'high;
--          signal sig3 : integer := integer'high;
--        begin
--          sig1 <= sig1 / 2 after 1 ns;
--          sig2 <= sig2 / 2 after 1 ns;
--          sig3 <= sig3 / 2 after 1 ns;
--          stop <= '0', '1' after 2 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : integer := integer'high;]
```

```
begin
#1[  sig@ <= sig@ / 2 after 1 ns;]
  stop <= '0', '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  162

PATHNAME :  [.BENCH.A.C.P6]shell2.sh
            (UNIX equivalent : bench/a/c/p6/shell2.sh)

PURPOSE :   Determine the simulation CPU time required to execute logical AND
            operations on signals.  The model simulated consists of a number of
            signal declarations and one logical AND signal assignment statement
            for each signal, plus a signal declaration/signal assignment
            statement/assertion that controls how long to simulate the model.
            The factors to be varied are the number of signal declarations/
            number of logical AND signal assignment statements and the length of
            time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal AND statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal other_sig : bit := '0';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= other_sig AND other_sig after 1 ns;
--          sig2 <= other_sig AND other_sig after 1 ns;
--          sig3 <= other_sig AND other_sig after 1 ns;
--          other_sig <= '0'
--                      ,'1' after 1 ns
--                      ,'0' after 2 ns;
--        end test;


entity test is end;
architecture test of test is
  signal other_sig : bit := '0';
```

```
#1[  signal sig@ : bit := '1';]
begin
#1[  sig@ <= other_sig AND other_sig after 1 ns;]
  other_sig <= '0'
#2[              ,'$2$1$0$' after @ ns];
end test;
```

```
TEST NUMBER :  163

PATHNAME :  [.BENCH.A.C.P7]shell2.sh
            (UNIX equivalent : bench/a/c/p7/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical OR
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical OR signal assignment statement
           for each signal, plus a signal declaration/signal assignment
           statement/assertion that controls how long to simulate the model.
           The factors to be varied are the number of signal declarations/
           number of logical OR signal assignment statements and the length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal OR statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal other_sig : bit := '0';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= other_sig OR other_sig after 1 ns;
--          sig2 <= other_sig OR other_sig after 1 ns;
--          sig3 <= other_sig OR other_sig after 1 ns;
--          other_sig <= '0'
--                      ,'1' after 1 ns
--                      ,'0' after 2 ns;
--        end test;


entity test is end;
architecture test of test is
  signal other_sig : bit := '0';
```

```
#1[  signal sig@ : bit := '1';]
begin
#1[  sig@ <= other_sig OR other_sig after 1 ns;]
  other_sig <= '0'
#2[               ,'$2$1$0$' after @ ns];
end test;
```

```
TEST NUMBER :  164

PATHNAME :  [.BENCH.A.C.P8]shell2.sh
            (UNIX equivalent : bench/a/c/p8/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NAND
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical NAND signal assignment statement
           for each signal, plus a signal declaration/signal assignment
           statement/assertion that controls how long to simulate the model.
           The factors to be varied are the number of signal declarations/
           number of logical NAND signal assignment statements and the length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal NAND statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '1';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= sig1 NAND stop after 1 ns;
--          sig2 <= sig2 NAND stop after 1 ns;
--          sig3 <= sig3 NAND stop after 1 ns;
--          stop <= '1', '0' after 2 ns;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '1';
#1[  signal sig@ : bit := '1';]
begin
```

```
#1[  sig@ <= sig@ NAND stop after 1 ns;]
  stop <= '1', '0' after %2% ns;
end test;
```

```
TEST NUMBER :  165

PATHNAME :  [.BENCH.A.C.P9]shell2.sh
            (UNIX equivalent : bench/a/c/p9/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOR
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical NOR signal assignment statement
           for each signal, plus a signal declaration/signal assignment
           statement/assertion that controls how long to simulate the model.
           The factors to be varied are the number of signal declarations/
           number of logical NOR signal assignment statements and the length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal NOR statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= sig1 NOR stop after 1 ns;
--          sig2 <= sig2 NOR stop after 1 ns;
--          sig3 <= sig3 NOR stop after 1 ns;
--          stop <= '0', '1' after 2 ns;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : bit := '1';]
begin
```

```
#1[  sig0 <= sig0 NOR stop after 1 ns;]
  stop <= '0', '1' after %2% ns;
end test;
```

```
TEST NUMBER :  166

PATHNAME :  [.BENCH.A.C.P10]shell2.sh
            (UNIX equivalent : bench/a/c/p10/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical XOR
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical XOR signal assignment statement
           for each signal, plus a signal declaration/signal assignment
           statement/assertion that controls how long to simulate the model.
           The factors to be varied are the number of signal declarations/
           number of logical XOR signal assignment statements and the length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal XOR statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '1';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= sig1 XOR stop after 1 ns;
--          sig2 <= sig2 XOR stop after 1 ns;
--          sig3 <= sig3 XOR stop after 1 ns;
--          stop <= '1', '0' after 2 ns;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '1';
#1[  signal sig@ : bit := '1';]
begin
```

```
#1[  sig@ <= sig@ XOR stop after 1 ns;]
  stop <= '1', '0' after %2% ns;
end test;
```

```
TEST NUMBER :  167

PATHNAME :  [.BENCH.A.C.P11]shell2.sh
            (UNIX equivalent : bench/a/c/p11/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOT
           operations on signals.  The model simulated consists of a number of
           signal declarations and one logical NOT signal assignment statement
           for each signal, plus a signal declaration/signal assignment
           statement/assertion that controls how long to simulate the model.
           The factors to be varied are the number of signal declarations/
           number of logical NOT signal assignment statements and the length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal NOT statements
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '1';
--          signal sig1 : bit := '1';
--          signal sig2 : bit := '1';
--          signal sig3 : bit := '1';
--        begin
--          sig1 <= NOT sig1 after 1 ns;
--          sig2 <= NOT sig2 after 1 ns;
--          sig3 <= NOT sig3 after 1 ns;
--          stop <= '1', '0' after 2 ns;
--          assert (stop = '1') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '1';
#1[  signal sig@ : bit := '1';]
```

```
begin
#1[  sig@ <= NOT sig@ after 1 ns;]
  stop <= '1', '0' after %2% ns;
  assert (stop = '1') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  168

PATHNAME :  [.BENCH.B.C.K.L1.P6]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical AND
           operations on a signal.  The model simulated is an architecture
           consisting of a process and two signal declarations, one bit_vector,
           one bit; the process consists of a for-loop; the for-loop contains a
           logical AND signal assignment statement, and the number of
           iterations of the loop is equal to the size of the bit_vector
           signal.  The factor to be varied is the bit_vector size/number of
           iterations of the for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of AND statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= sig(i) AND static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
```

```
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= sig(i) AND static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  169

PATHNAME :  [.BENCH.B.C.K.L1.P7]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p7/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical OR
           operations on a signal.  The model simulated is an architecture
           consisting of a process and two signal declarations, one bit_vector,
           one bit; the process consists of a for-loop; the for-loop contains a
           logical OR signal assignment statement, and the number of
           iterations of the loop is equal to the size of the bit_vector
           signal.  The factor to be varied is the bit_vector size/number of
           iterations of the for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of OR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= sig(i) OR static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
```

```
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= sig(i) OR static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  170

PATHNAME :  [.BENCH.B.C.K.L1.P8]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p8/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NAND
           operations on a signal.  The model simulated is an architecture
           consisting of a process and two signal declarations, one bit_vector,
           one bit; the process consists of a for-loop; the for-loop contains a
           logical NAND signal assignment statement, and the number of
           iterations of the loop is equal to the size of the bit_vector
           signal.  The factor to be varied is the bit_vector size/number of
           iterations of the for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of NAND statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= sig(i) NAND static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
```

```
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= sig(i) NAND static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  171

PATHNAME :  [.BENCH.B.C.K.L1.P9]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p9/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOR
           operations on a signal.  The model simulated is an architecture
           consisting of a process and two signal declarations, one bit_vector,
           one bit; the process consists of a for-loop; the for-loop contains a
           logical NOR signal assignment statement, and the number of
           iterations of the loop is equal to the size of the bit_vector
           signal.  The factor to be varied is the bit_vector size/number of
           iterations of the for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of NOR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vnd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '0';
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= sig(i) NOR static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '0';
```

```
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= sig(i) NOR static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  172

PATHNAME :  [.BENCH.B.C.K.L1.P10]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p10/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical XOR
           operations on a signal.  The model simulated is an architecture
           consisting of a process and two signal declarations, one bit_vector,
           one bit; the process consists of a for-loop; the for-loop contains a
           logical XOR signal assignment statement, and the number of
           iterations of the loop is equal to the size of the bit_vector
           signal.  The factor to be varied is the bit_vector size/number of
           iterations of the for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of XOR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= sig(i) XOR static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
```

```
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= sig(i) XOR static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  173

PATHNAME :  [.BENCH.B.C.K.L1.P11]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute logical NOT
           operations on a signal.  The model simulated is an architecture
           consisting of a process and two signal declarations, one bit_vector,
           one bit; the process consists of a for-loop; the for-loop contains a
           logical NOT signal assignment statement, and the number of
           iterations of the loop is equal to the size of the bit_vector
           signal.  The factor to be varied is the bit_vector size/number of
           iterations of the for-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of NOT statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= NOT sig(i);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
begin
  process
```

```
  begin
    for i in 1 to %1% loop
      sig(i) <= NOT sig(i);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  174

PATHNAME :  [.BENCH.B.C.K.L3.P11]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p11/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition and
           logical NOT signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one bit_vector, one bit; the process consists of an integer
           loop-counter variable declaration and a while-loop; the while-loop
           contains a variable addition statement for incrementing the
           loop-counter and a logical NOT signal assignment statement, and the
           number of iterations of the loop is equal to the size of the
           bit_vector signal.  The factor to be varied is the bit_vector size/
           number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of NOT statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= NOT sig(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
```

```
architecture test of test is
  signal sig : bit_vector(1 to %1%);
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= NOT sig(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

TEST NUMBER : 175

PATHNAME : [.BENCH.B.C.K.L3.P10]shell1.sh
(UNIX equivalent : bench/b/c/k/l3/p10/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute addition and logical XOR signal assignment statements. The model simulated is an architecture consisting of a process and two signal declarations, one bit_vector, one bit; the process consists of an integer loop-counter variable declaration and a while-loop; the while-loop contains a variable addition statement for incrementing the loop-counter and a logical XOR signal assignment statement, and the number of iterations of the loop is equal to the size of the bit_vector signal. The factor to be varied is the bit_vector size/ number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of XOR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= sig(loop_counter) XOR static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= sig(loop_counter) XOR static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  176

PATHNAME :  [.BENCH.B.C.K.L3.P9]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p9/shell1.sh)

PURPOSE :   Determine the simulation CPU time required to execute addition and
            logical NOR signal assignment statements.  The model simulated is an
            architecture consisting of a process and two signal declarations,
            one bit_vector, one bit; the process consists of an integer
            loop-counter variable declaration and a while-loop; the while-loop
            contains a variable addition statement for incrementing the
            loop-counter and a logical NOR signal assignment statement, and the
            number of iterations of the loop is equal to the size of the
            bit_vector signal.  The factor to be varied is the bit_vector size/
            number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of NOR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '0';
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= sig(loop_counter) NOR static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '0';
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= sig(loop_counter) NOR static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  177

PATHNAME :  [.BENCH.B.C.K.L3.P8]shell1.sh
            (UNIX equivalent : bench/b/c/k/13/p8/shell1.sh)

PURPOSE :   Determine the simulation CPU time required to execute addition and
            logical NAND signal assignment statements.  The model simulated is
            an architecture consisting of a process and two signal declarations,
            one bit_vector, one bit; the process consists of an integer
            loop-counter variable declaration and a while-loop; the while-loop
            contains a variable addition statement for incrementing the
            loop-counter and a logical NAND signal assignment statement, and the
            number of iterations of the loop is equal to the size of the
            bit_vector signal.  The factor to be varied is the bit_vector size/
            number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of NAND statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= sig(loop_counter) NAND static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= sig(loop_counter) NAND static_sig;
    end loop;
    wait;
  end process;
end test;
```

TEST NUMBER : 178

PATHNAME : [.BENCH.B.C.K.L3.P7]shell1.sh
(UNIX equivalent : bench/b/c/k/l3/p7/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute addition and logical OR signal assignment statements. The model simulated is an architecture consisting of a process and two signal declarations, one bit_vector, one bit; the process consists of an integer loop-counter variable declaration and a while-loop; the while-loop contains a variable addition statement for incrementing the loop-counter and a logical OR signal assignment statement, and the number of iterations of the loop is equal to the size of the bit_vector signal. The factor to be varied is the bit_vector size/ number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of OR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= sig(loop_counter) OR static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= sig(loop_counter) OR static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  179

PATHNAME :  [.BENCH.B.C.K.L3.P6]shell1.sh
            (UNIX equivalent : bench/b/c/k/13/p6/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition and
           logical AND signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one bit_vector, one bit; the process consists of an integer
           loop-counter variable declaration and a while-loop; the while-loop
           contains a variable addition statement for incrementing the
           loop-counter and a logical AND signal assignment statement, and the
           number of iterations of the loop is equal to the size of the
           bit_vector signal.  The factor to be varied is the bit_vector size/
           number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal bit_vector size/number of AND statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit_vector(1 to 10);
--          signal static_sig : bit := '1';
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= sig(loop_counter) AND static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal sig : bit_vector(1 to %1%);
  signal static_sig : bit := '1';
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= sig(loop_counter) AND static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  180

PATHNAME :  [.BENCH.B.C.K.L3.P6]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p6/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing addition
           and logical AND variable assignment statements.  The model simulated
           is an architecture consisting of a process.  The process consists of
           a variable bit_vector declaration , an integer variable declaration
           (loop-counter), and a while-loop; the while-loop contains a variable
           addition assignment statement to increment the loop-counter and a
           logical AND statement, and the number of iterations of the loop is
           equal to the size of the variable bit_vector. The factor to be
           varied is the bit_vector size/number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of AND statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            variable var : bit_vector(1 to 10);
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := var(loop_counter) AND var(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
begin
    process
    variable var : bit_vector(1 to %1%);
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := var(loop_counter) AND var(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  181

PATHNAME :  [.BENCH.B.C.K.L3.P7]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p7/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing addition
           and logical OR variable assignment statements.  The model simulated
           is an architecture consisting of a process.  The process consists of
           a variable bit_vector declaration , an integer variable declaration
           (loop-counter), and a while-loop; the while-loop contains a variable
           addition assignment statement to increment the loop-counter and a
           logical OR statement, and the number of iterations of the loop is
           equal to the size of the variable bit_vector. The factor to be
           varied is the bit_vector size/number of iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of OR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            variable var : bit_vector(1 to 10);
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := var(loop_counter) OR var(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
begin
    process
    variable var : bit_vector(1 to %1%);
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := var(loop_counter) OR var(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  182

PATHNAME :  [.BENCH.B.C.K.L3.P8]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p8/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing addition
           and logical NAND variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of a variable bit_vector declaration , an integer variable
           declaration (loop-counter), and a while-loop; the while-loop
           contains a variable addition assignment statement to increment the
           loop-counter and a logical NAND statement, and the number of
           iterations of the loop is equal to the size of the variable
           bit_vector. The factor to be varied is the bit_vector size/number of
           iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of NAND statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            variable var : bit_vector(1 to 10);
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := var(loop_counter) NAND var(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
```

```
architecture test of test is
begin
    process
    variable var : bit_vector(1 to %1%);
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := var(loop_counter) NAND var(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  183

PATHNAME :  [.BENCH.B.C.K.L3.P9]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p9/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing addition
           and logical NOR variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of a variable bit_vector declaration , an integer variable
           declaration (loop-counter), and a while-loop; the while-loop
           contains a variable addition assignment statement to increment the
           loop-counter and a logical NOR statement, and the number of
           iterations of the loop is equal to the size of the variable
           bit_vector. The factor to be varied is the bit_vector size/number of
           iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of NOR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            variable var : bit_vector(1 to 10);
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := var(loop_counter) NOR var(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
```

```
architecture test of test is
begin
    process
    variable var : bit_vector(1 to %1%);
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := var(loop_counter) NOR var(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  184

PATHNAME :  [.BENCH.B.C.K.L3.P10]shell0.sh
            (UNIX equivalent : bench/b/c/k/13/p10/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing addition
           and logical XOR variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of a variable bit_vector declaration , an integer variable
           declaration (loop-counter), and a while-loop; the while-loop
           contains a variable addition assignment statement to increment the
           loop-counter and a logical XOR statement, and the number of
           iterations of the loop is equal to the size of the variable
           bit_vector. The factor to be varied is the bit_vector size/number of
           iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of XOR statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            variable var : bit_vector(1 to 10);
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := var(loop_counter) XOR var(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
```

```
architecture test of test is
begin
    process
    variable var : bit_vector(1 to %1%);
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := var(loop_counter) XOR var(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  185

PATHNAME :  [.BENCH.B.C.K.L3.P11]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p11/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing addition
           and logical NOT variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of a variable bit_vector declaration , an integer variable
           declaration (loop-counter), and a while-loop; the while-loop
           contains a variable addition assignment statement to increment the
           loop-counter and a logical NOT statement, and the number of
           iterations of the loop is equal to the size of the variable
           bit_vector. The factor to be varied is the bit_vector size/number of
           iterations of the while-loop.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size/number of NOT statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            variable var : bit_vector(1 to 10);
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := NOT var(loop_counter);
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
```

```
architecture test of test is
begin
    process
    variable var : bit_vector(1 to %1%);
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := NOT var(loop_counter);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  186

PATHNAME :  [.BENCH.B.C.K.L1.P1]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           addition assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an integer array variable declaration and a for-loop.
           The number of iterations of the loop is equal to the size of the
           array.  The for-loop contains a variable addition assignment
           statement.  The factors to be varied are the number of processes and
           the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of addition statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 10) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
--              var(i) := var(i) + 1;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 10) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
--              var(i) := var(i) + 1;
--            end loop;
```

```
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
  begin
    for i in 1 to %  % loop
      var(i) := var( ) + 1;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  187

PATHNAME :  [.BENCH.B.C.K.L1.P2]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           subtraction assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an integer array variable declaration and a for-loop.
           The number of iterations of the loop is equal to the size of the
           array.  The for-loop contains a variable subtraction assignment
           statement.  The factors to be varied are the number of processes and
           the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of subtraction statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 0;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
--              var(i) := static_var - 1;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 0;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
```

```
--              var(i) := static_var - 1;
--            end loop;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 0;
    variable var : var_array;
  begin
    for i in 1 to %1% loop
      var(i) := static_var - 1;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  188

PATHNAME :  [.BENCH.B.C.K.L1.P3]shell0.sh
            (UNIX equivalent . bench/b/c/k/l1/p3/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           multiplication assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an integer array variable declaration and a for-loop.
           The number of iterations of the loop is equal to the size of the
           array.  The for-loop contains a variable multiplication assignment
           statement.  The factors to be varied are the number of processes and
           the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of multiplication statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 2;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
--              var(i) := static_var * 5;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 2;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
```

```
--              var(i) := static_var * 5;
--            end loop;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 2;
    variable var : var_array;
  begin
    for i in 1 to %1% loop
      var(i) := static_var * 5;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  189

PATHNAME :  [.BENCH.B.C.K.L1.P4]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p4/shell0.sh)

PURPOSE :   Determine the simulation CPU time required to execute variable
            division assignment statements.  The model simulated is an
            architecture consisting of a number of processes.  Each process
            consists of an integer array variable declaration and a for-loop.
            The number of iterations of the loop is equal to the size of the
            array.  The for-loop contains a variable division assignment
            statement.  The factors to be varied are the number of processes and
            the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of division statement iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10,2
--    (UNIX equivalent : % sim gen -param=\"\shell0 sh\"\,\"\test.vhd\"\,10,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 2;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
--              var(i) := 10 / static_var;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 2;
--            variable var : var_array;
--          begin
--            for i in 1 to 10 loop
```

```
--              var(i) := 10 / static_var;
--            end loop;
--            wait;
--          end process pr2;
--        end test;



entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 2;
    variable var : var_array;
  begin
    for i in 1 to %1% loop
      var(i) := 10 / static_var;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  190

PATHNAME :  [.BENCH.B.C.K.L1.P1]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           for-loop.  The number of iterations of the loop is equal to the
           size of the array.  The for-loop contains an addition signal
           assignment statement.  The factor to be varied is the array size/
           number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of addition statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
          architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 1;
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= sig(i) + static_sig;
--            end loop;
--            wait;
--          end process;
--        end test,



entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
```

```
  signal sig : sig_array;
  signal static_sig : integer := 1;
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= sig(i) + static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  191

PATHNAME :  [.BENCH.B.C.K.L1.P2]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p2/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute subtraction
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           for-loop.  The number of iterations of the loop is equal to the
           size of the array.  The for-loop contains a subtraction signal
           assignment statement.  The factor to be varied is the array size/
           number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of subtraction statement iterations in
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 5;
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= static_sig - 1;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
  signal static_sig : integer := 5;
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= static_sig - 1;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  192

PATHNAME :  [.BENCH.B.C.K.L1.P3]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute multiplication
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           for-loop.  The number of iterations of the loop is equal to the
           size of the array.  The for-loop contains a multiplication signal
           assignment statement.  The factor to be varied is the array size/
           number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of multiplication statement iterations in
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 5;
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= static_sig * 2;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
```

```
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
  signal static_sig : integer := 5;
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= static_sig * 2;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  193

PATHNAME :  [.BENCH.B.C.K.L1.P4]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute division
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           for-loop.  The number of iterations of the loop is equal to the
           size of the array.  The for-loop contains a division signal
           assignment statement.  The factor to be varied is the array size/
           number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of division statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 10;
--        begin
--          process
--          begin
--            for i in 1 to 10 loop
--              sig(i) <= static_sig / 2;
--            end loop;
--            wait;
--          end process;
--        end test;



entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
```

```
  signal sig : sig_array;
  signal static_sig : integer := 10;
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= static_sig / 2;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  194

PATHNAME :  [.BENCH.B.C.K.L3.P1]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p1/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           variable assignment statements.  The model simulated is an
           architecture consisting of a process.  The process consists of
           two integer variable declarations (one is a loop-counter), one
           integer array variable declaration, and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains two addition variable assignment statements (one
           to increment the loop-counter).  The factor to be varied is the array
           size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : variable array size/number of addition statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 1;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := static_var + 1;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
    process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 1;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := static_var + 1;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  195

PATHNAME :  [.BENCH.B.C.K.L3.P2]shell0.sh
            (UNIX equivalent : bench/b/c/k/13/p2/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           and subtraction variable assignment statements.  The model simulated
           is an architecture consisting of a process.  The process consists of
           two integer variable declarations (one is a loop-counter), one
           integer array variable declaration, and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and a subtraction variable assignment
           statement.  The factor to be varied is the array size/number of loop
           iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : variable array size/number of subtraction statement iterations in
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 1;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := static_var - 1;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
    process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 1;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := static_var - 1;
    end loop;
    wait;
  end process;
end test;
```

TEST NUMBER : 196

PATHNAME : [.BENCH.B.C.K.L3.P3]shell0.sh
(UNIX equivalent : bench/b/c/k/l3/p3/shell0.sh)

PURPOSE : Determine the simulation CPU time required to execute addition and multiplication variable assignment statements. The model simulated is an architecture consisting of a process. The process consists of two integer variable declarations (one is a loop-counter), one integer array variable declaration, and a while-loop. The number of iterations of the loop is equal to the size of the array. The while-loop contains an addition variable assignment statement to increment the loop-counter and a multiplication variable assignment statement. The factor to be varied is the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : variable array size/number of multiplication statement iterations in
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 2;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := static_var * 5;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
    process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 2;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := static_var * 5;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  197

PATHNAME :  [.BENCH.B.C.K.L3.P4]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p4/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           and division variable assignment statements.  The model simulated is
           an architecture consisting of a process.  The process consists of
           two integer variable declarations (one is a loop-counter), one
           integer array variable declaration, and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and a division variable assignment
           statement.  The factor to be varied is the array size/number of loop
           iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : variable array size/number of division statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--            process
--            type var_array is array(1 to 10) of integer;
--            variable static_var : integer := 2;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := 10 / static_var;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
    process
    type var_array is array(1 to %1%) of integer;
    variable static_var : integer := 2;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := 10 / static_var;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  198

PATHNAME :  [.BENCH.B.C.K.L3.P1]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           while-loop and an integer variable declaration (loop-counter).  The
           number of iterations of the loop is equal to the size of the array.
           The while-loop contains one addition signal assignment statement and
           one addition variable assignment statement to increment the
           loop-counter.  The factor to be varied is the array size/number of
           loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of addition statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 1;
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= static_sig + 1;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
  signal static_sig : integer := 1;
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= static_sig + 1;
    end loop;
    wait;
  end process;
end test;
```

TEST NUMBER : 199

PATHNAME : [.BENCH.B.C.K.L3.P2]shell1.sh
(UNIX equivalent : bench/b/c/k/l3/p2/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute subtraction signal assignment statements. The model simulated is an architecture consisting of a process and two signal declarations, one integer and one integer array. The process consists of a while-loop and an integer variable declaration (loop-counter). The number of iterations of the loop is equal to the size of the array. The while-loop contains one subtraction signal assignment statement and one addition variable assignment statement to increment the loop-counter. The factor to be varied is the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of subtraction statement iterations in
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 1;
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= static_sig - 1;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
  signal static_sig : integer := 1;
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= static_sig - 1;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  200

PATHNAME :  [.BENCH.B.C.K.L3.P3]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p3/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute multiplication
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           while-loop and an integer variable declaration (loop-counter).  The
           number of iterations of the loop is equal to the size of the array.
           The while-loop contains one multiplication signal assignment
           statement and one addition variable assignment statement to
           increment the loop-counter.  The factor to be varied is the array
           size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of multiplication statement iterations in
--        process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 2;
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= static_sig * 5;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
  signal static_sig : integer := 2;
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= static_sig * 5;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  201

PATHNAME :  [.BENCH.B.C.K.L3.P4]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p4/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute division
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           while-loop and an integer variable declaration (loop-counter).  The
           number of iterations of the loop is equal to the size of the array.
           The while-loop contains one division signal assignment statement and
           one addition variable assignment statement to increment the
           loop-counter.  The factor to be varied is the array size/number of
           loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of division statement iterations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 10) of integer;
--          signal sig : sig_array;
--          signal static_sig : integer := 2;
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 10 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= 10 / static_sig;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
  signal static_sig : integer := 2;
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= 10 / static_sig;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  202

PATHNAME :  [.BENCH.B.C.T10]shell.sh
            (UNIX equivalent : bench/b/c/t10/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for
           process labels.  The model simulated is an architecture consisting
           of a signal declaration and a process.  The process consists of a
           logical NOT signal assignment statement.  The factor to be varied is
           the length of the process label.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of process label
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : bit := '0';
--        begin
--          ABCDEFGHIJ : process
--          begin
--            sig <= not sig;
--            wait;
--          end process ABCDEFGHIJ;
--        end test;


entity test is end;
architecture test of test is
  signal sig : bit := '0';
begin
  ?1? : process
  begin
    sig <= not sig;
    wait;
  end process ?1?;
end test;
```

```
TEST NUMBER :  203

PATHNAME :  [.BENCH.B.C.T3]shell.sh
            (UNIX equivalent : bench/b/c/t3/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for
           constant identifiers.  The model simulated is an architecture
           consisting of a constant declaration, a signal declaration, and a
           signal assignment statement.  The factor to be varied is the length
           of the constant identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of constant identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          constant ABCDEFGHIJ : bit := '1';
--          signal sig : bit := '0';
--        begin
--          sig <= ABCDEFGHIJ;
--        end test;


entity test is end;
architecture test of test is
  constant ?1? : bit := '1';
  signal sig : bit := '0';
begin
  sig <= ?1?;
end test;
```

```
TEST NUMBER :  204

PATHNAME :  [.BENCH.B.C.T4]shell.sh
            (UNIX equivalent : bench/b/c/t4/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           type identifier.  The model simulated is an architecture consisting
           of a type declaration, a signal declaration, and a signal assignment
           statement.  The factor to be varied is the length of the type
           identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of type identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type ABCDEFGHIJ is (first,second,third);
--          signal sig : ABCDEFGHIJ;
--        begin
--          sig <= ABCDEFGHIJ'right;
--        end test;


entity test is end;
architecture test of test is
  type ?1? is (first,second,third);
  signal sig : ?1?;
begin
  sig <= ?1?'right;
end test;
```

```
TEST NUMBER :  205

PATHNAME :  [.BENCH.B.C.T5]shell.sh
            (UNIX equivalent : bench/b/c/t5/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           subtype identifier.  The model simulated is an architecture
           consisting of a subtype declaration, a signal declaration, and a
           signal assignment statement.  The factor to be varied is the length
           of the subtype identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of subtype identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          subtype ABCDEFGHIJ is string(1 to 5);
--          signal sig : ABCDEFGHIJ;
--        begin
--          sig <= "abcde";
--        end test;


entity test is end;
architecture test of test is
  subtype ?1? is string(1 to 5);
  signal sig : ?1?;
begin
  sig <= "abcde";
end test;
```

```
TEST NUMBER :  206

PATHNAME :  [.BENCH.B.C.T1A]shell.sh
            (UNIX equivalent : bench/b/c/t1a/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           signal identifier for a signal declared in an architecture.  The
           model simulated is an architecture consisting of a signal declaration
           and a signal assignment statement.  The factor to be varied is the
           length of the signal identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of signal identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal ABCDEFGHIJ : bit := '1';
--        begin
--          ABCDEFGHIJ <= '0';
--        end test;


entity test is end;
architecture test of test is
  signal ?1? . bit := '1';
begin
  ?1? <= '0';
end test;
```

```
TEST NUMBER :  207

PATHNAME :  [.BENCH.B.C.T1B]shell.sh
            (UNIX equivalent : bench/b/c/t1b/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           signal identifier for a signal declared in a block.  The model
           simulated is an architecture consisting of a block.  The block
           consists of a signal declaration and a signal assignment statement.
           The factor to be varied is the length of the signal identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of signal identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          blk : block
--            signal ABCDEFGHIJ : bit := '1';
--          begin
--            ABCDEFGHIJ <= '0';
--          end block blk;
--        end test;


entity test is end;
architecture test of test is
begin
  blk : block
    signal ?1? : bit := '1';
  begin
    ?1? <= '0';
  end block blk;
end test;
```

```
TEST NUMBER :  208

PATHNAME :  [.BENCH.B.C.T1C]shell.sh
            (UNIX equivalent : bench/b/c/t1c/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           signal identifier for a signal declared in a port clause.  The model
           simulated is a two-level hierarchy of architectures.  The lower-
           level architecture is a component in the upper-level architecture.
           The component has a port clause with one interface-signal-
           declaration.  The factor to be varied is the length of the signal
           identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of signal identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test1 is
--          port(signal ABCDEFGHIJ : out bit := '1');
--        end test1;
--        architecture test1 of test1 is
--        begin
--          ABCDEFGHIJ <= '0';
--        end test1;
--        entity test0 is end;
--        architecture test0 of test0 is
--          signal other_sig : bit := '1';
--          component test1
--            port(signal ABCDEFGHIJ : out bit);
--          end component;
--          for all : test1 use entity work.test1(test1);
--        begin
--          comp : test1
--            port map(ABCDEFGHIJ => other_sig);
--        end test0;


entity test1 is
```

```
  port(signal ?1? : out bit := '1');
end test1;
architecture test1 of test1 is
begin
  ?1? <= '0';
end test1;
entity test0 is end;
architecture test0 of test0 is
  signal other_sig : bit := '1';
  component test1
    port(signal ?1? : out bit);
  end component;
  for all : test1 use entity work.test1(test1);
begin
  comp : test1
    port map(?1? => other_sig);
end test0;
```

```
TEST NUMBER :  209

PATHNAME :  [.BENCH.B.C.T2A]shell.sh
            (UNIX equivalent : bench/b/c/t2a/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           variable identifier of a variable declared in a process.  The
           model simulated is an architecture consisting of a process.  The
           process consists of a variable declaration and a variable assignment
           statement.  The factor to be varied is the length of the variable
           identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of variable identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            variable ABCDEFGHIJ : bit := '1';
--          begin
--            ABCDEFGHIJ := '0';
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
begin
  process
    variable ?1? : bit := '1';
  begin
    ?1? := '0';
    wait;
  end process;
end test;
```

```
TEST NUMBER :  210

PATHNAME :  [.BENCH.B.C.T6]shell.sh
            (UNIX equivalent : bench/b/c/t6/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           component identifier.  The model simulated is a two-level hierarchy
           of architectures.  The lower-level architecture is a component in
           the upper-level architecture.  The factor to be varied is the length
           of the component identifier.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of component instantiation identifier
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test1 is
--          port(signal sig : out bit := '1');
--        end test1;
--        architecture test1 of test1 is
--        begin
--          sig <= '0';
--        end test1;
--        entity test0 is end;
--        architecture test0 of test0 is
--          signal other_sig : bit := '1';
--          component test1
--            port(signal sig : out bit);
--          end component;
--          for all : test1 use entity work.test1(test1);
--        begin
--          ABCDEFGHIJ : test1
--            port map(sig => other_sig);
--        end test0;
```

```
entity test1 is
  port(signal sig : out bit := '1');
end test1;
architecture test1 of test1 is
begin
  sig <= '0';
end test1;
entity test0 is end;
architecture test0 of test0 is
  signal other_sig : bit := '1';
  component test1
    port(signal sig : out bit);
  end component;
  for all : test1 use entity work.test1(test1);
begin
  ?1? : test1
    port map(sig => other_sig);
end test0;
```

```
TEST NUMBER :  211

PATHNAME :  [.BENCH.B.C.T9]shell.sh
            (UNIX equivalent : bench/b/c/t9/shell.sh)

PURPOSE :  Determine the maximum length (number of characters) allowed for a
           block label.  The model simulated is an architecture consisting of a
           block.  The block consists of a signal declaration and a signal
           assignment statement.  The factor to be varied is the length of the
           block label.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : length of block label
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          ABCDEFGHIJ : block
--            signal sig : bit := '1';
--          begin
--            sig <= '0';
--          end block ABCDEFGHIJ;
--        end test;


entity test is end;
architecture test of test is
begin
  ?1? : block
    signal sig : bit := '1';
  begin
    sig <= '0';
  end block ?1?;
end test;
```

```
TEST NUMBER :  212

PATHNAME :  [.BENCH.A.C.H1]shell.sh
            (UNIX equivalent : bench/a/c/h1/shell.sh)

PURPOSE :  Determine the maximum number of procedure declarations/calls allowed
           in one model; try to get an idea of the simulation CPU time required
           for executing procedure calls.  The model simulated is an entity
           consisting of a number of procedure declarations and an architecture
           consisting of a procedure call for each declaration.  The procedure
           consists of a variable declaration and a variable assignment
           statement.  The factor to be varied is the number of procedure
           declarations/calls.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of procedures in entity declaration/number of procedure calls
--        in architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure pro1(param : in bit) is
--            variable local_var : bit;
--          begin
--            local_var := param;
--          end pro1;
--          procedure pro2(param : in bit) is
--            variable local_var : bit;
--          begin
--            local_var := param;
--          end pro2;
--        end test;
--        architecture test of test is
--        begin
--          pro1('0');
--          pro2('1');
--        end test;
```

```
entity test is
#1[  procedure pro@(param : in bit) is
    variable local_var : bit;
  begin
    local_var := param;
  end pro@;]
end test;
architecture test of test is
begin
#1[  pro@('$2$0$1$');]
end test;
```

```
TEST NUMBER :  213

PATHNAME :  [.BENCH.A.C.H2]shell.sh
            (UNIX equivalent : bench/a/c/h2/shell.sh)

PURPOSE :  Determine the maximum number of procedure declarations/calls allowed
           in one model; try to get an idea of the simulation CPU time required
           for executing procedure calls.  The model simulated is an
           architecture consisting of a number of procedure declarations and a
           procedure call for each declaration.  The procedure consists of a
           variable declaration and a variable assignment statement.  The
           factor to be varied is the number of procedure declarations/calls.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of procedures/procedure calls in architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure pro1(param : in bit) is
--            variable local_var : bit;
--          begin
--            local_var := param;
--          end pro1;
--          procedure pro2(param : in bit) is
--            variable local_var : bit;
--          begin
--            local_var := param;
--          end pro2;
--        begin
--          pro1('0');
--          pro2('1');
--        end test;


entity test is end;
architecture test of test is
#1[  procedure pro@(param : in bit) is
```

```
    variable local_var : bit;
  begin
    local_var := param;
  end pro@;]
begin
#1[  pro@('$2$0$1$');]
end test;
```

```
TEST NUMBER :  214

PATHNAME :  [.BENCH.A.C.H3]shell.sh
            (UNIX equivalent : bench/a/c/h3/shell.sh)

PURPOSE :  Determine the maximum number of procedure declarations/calls allowed
           in a process; try to get an idea of the simulation CPU time required
           for executing procedure calls.  The model simulated is an
           architecture consisting of a process.  The process consists of a
           number of procedure declarations and a procedure call for each
           declaration.  The procedure consists of a variable declaration and a
           variable assignment statement.  The factor to be varied is the
           number of procedure declarations/calls.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of procedures/procedure calls in process
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            procedure pro1(param : in bit) is
--              variable local_var : bit;
--            begin
--              local_var := param;
--            end pro1;
--            procedure pro2(param : in bit) is
--              variable local_var : bit;
--            begin
--              local_var := param;
--            end pro2;
--          begin
--            pro1('0');
--            pro2('1');
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
  process
#1[    procedure pro@(param : in bit) is
      variable local_var : bit;
    begin
      local_var := param;
    end pro@;]
  begin
#1[    pro@('$2$0$1$');]
    wait;
  end process;
end test;
```

TEST NUMBER : 215

PATHNAME : [.BENCH.A.C.I1]shell.sh
(UNIX equivalent : bench/a/c/i1/shell.sh)

PURPOSE : Determine the maximum number of function declarations/calls allowed in one model; try to get an idea of the simulation CPU time required for executing function calls. The model simulated is an entity consisting of a number of function declarations and an architecture consisting of the same number of signal declarations and a function call (via a signal assignment statement) for each declaration. The function consists of a return statement. The factor to be varied is the number of function declarations/calls.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of functions in entity declaration/number of function calls/
--        signal declarations in architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\.2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function fun1(param : in bit) return bit is
--          begin
--            return (not param);
--          end fun1;
--          function fun2(param : in bit) return bit is
--          begin
--            return (not param);
--          end fun2;
--        end test;
--        architecture test of test is
--          signal sig1 : bit;
--          signal sig2 : bit;
--        begin
--          sig1 <= fun1('0');
--          sig2 <= fun2('1');
--        end test;
```

```
entity test is
#1[  function fun@(param : in bit) return bit is
  begin
    return (not param);
  end fun@;]
end test;
architecture test of test is
#1[  signal sig@ : bit;]
begin
#1[  sig@ <= fun@('$2$0$1$');]
end test;
```

```
TEST NUMBER :  216

PATHNAME :  [.BENCH.A.C.I2]shell.sh
            (UNIX equivalent : bench/a/c/i2/shell.sh)

PURPOSE :  Determine the maximum number of function declarations/calls allowed
           in one model; try to get an idea of the simulation CPU time required
           for executing function calls.  The model simulated is an
           architecture consisting of a number of function declarations, the
           same number of signal declarations, and a function call (via signal
           assignment statement) for each function declaration.  The function
           consists of a return statement.  The factor to be varied is the
           number of function declarations/calls.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of functions/function calls/signal declarations in architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture test of test is
--          signal sig1 : bit;
--          signal sig2 : bit;
--          function fun1(param : in bit) return bit is
--          begin
--            return (not param);
--          end fun1;
--          function fun2(param : in bit) return bit is
--          begin
--            return (not param);
--          end fun2;
--        begin
--          sig1 <= fun1('0');
--          sig2 <= fun2('1');
--        end test;


entity test is
```

```
end test;
architecture test of test is
#1[  signal sig@ : bit;]
#1[  function fun@(param : in bit) return bit is
  begin
    return (not param);
  end fun@;]
begin
#1[  sig@ <= fun@('$2$0$1$');]
end test;
```

TEST NUMBER : 217

PATHNAME : [.BENCH.A.C.I3]shell.sh
(UNIX equivalent : bench/a/c/i3/shell.sh)

PURPOSE : Determine the maximum number of function declarations/calls allowed in a process; try to get an idea of the simulation CPU time required for executing function calls. The model simulated is an architecture consisting of a process. The process consists of a a variable declaration, a number of function declarations and a function call (via variable assignment statement) for each variable declaration. The function consists of a return statement. The factor to be varied is the number of function declarations/calls.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of functions/function calls in process
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture test of test is
--        begin
--          process
--            variable var : bit;
--            function fun1(param : in bit) return bit is
--            begin
--              return (not param);
--            end fun1;
--            function fun2(param : in bit) return bit is
--            begin
--              return (not param);
--            end fun2;
--          begin
--            var := fun1('0');
--            var := fun2('1');
--            wait;
--          end process;
--        end test;
```

```
entity test is
end test;
architecture test of test is
begin
  process
    variable var : bit;
#1[    function fun@(param : in bit) return bit is
    begin
      return (not param);
    end fun@;]
  begin
#1[    var := fun@('$2$0$1$');]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  218

PATHNAME :  [.BENCH.B.C.N]shell.sh
            (UNIX equivalent : bench/b/c/n/shell.sh)

PURPOSE :   Determine the maximum number of choices allowed in a case statement.
            The model simulated is an architecture consisting of a signal
            declaration and a process.  The process consists of a case statement
            with a number of choices, each containing a signal assignment
            statement.  The factor to be varied is the number of choices in the
            case statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 27 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of case_statement_alternatives
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          subtype sig_type is integer range 1 to 3;
--          signal sig : sig_type;
--        begin
--          process
--          begin
--            case sig is
--              when 1 => sig <= sig_type'low;
--              when 2 => sig <= sig_type'high;
--              when 3 => sig <= sig_type'low;
--            end case;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
  subtype sig_type is integer range 1 to %1%;
  signal sig : sig_type;
begin
```

```
  process
  begin
    case sig is
#1[       when @ => sig <= $2$sig_type'low$sig_type'high$;]
    end case;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  219

PATHNAME :  [.BENCH.B.C.F1.K.L1.P10]shell.sh
            (UNIX equivalent : bench/b/c/f1/k/l1/p10/shell.sh)

PURPOSE :  Determine the effect on CPU time when analyzing, model generating,
           building, and simulating the following model :  two entities/
           architectures, where one architecture is a component in the other
           architecture.  The component-level entity's port clause contains
           one bit type signal (output) and one bit_vector type signal (input).
           The component-level architecture consists of a process.  The process
           has a variable declaration and a for-loop whose number of iterations
           is equal to the size of the bit_vector in the port clause.  The for-
           loop contains a logical XOR signal assignment statement.  The top-
           level architecture consists of a bit-vector signal declaration for
           output, a number of bit_vector signal declarations for input, a
           component declaration matching the component-level entity/
           architecture, and a number of component instantiations, where the
           input signal maps to the corresponding input bit_vector signal
           declared above, and the output signal maps to one element of the
           output bit_vector signal declared above.  The factors to be varied
           are the size of the input bit_vector in the port clause, and the
           number of components in the top-level architecture/size of output
           bit_vector signal in the top-level architecture/number of input
           bit_vector signal declarations in the top-level architecture.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 28 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size of bit_vector signal in port clause
--    2 : number of components in top-level architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test1 is
--          port(signal sig_out : out bit := '0';
--               signal sig_vec : in bit_vector(1 to 3));
--        end test1;
--        architecture test1 of test1 is
--        begin
--          process
```

```
--            variable bit_var : bit := '0';
--          begin
--            for i in sig_vec'range loop
--              bit_var := bit_var XOR sig_vec(i);
--            end loop;
--            sig_out <= bit_var;
--            wait;
--          end process;
--        end test1;
--        entity test0 is end;
--        architecture test0 of test0 is
--          signal sig_out_vec : bit_vector(1 to 2);
--          signal sig_in_vec1 : bit_vector(1 to 3);
--          signal sig_in_vec2 : bit_vector(1 to 3);
--          component test1
--            port(signal sig_out : out bit;
--                 signal sig_vec : in bit_vector(1 to 3));
--          end component;
--          for all : test1 use entity work.test1(test1);
--        begin
--          comp1 : test1
--            port map(sig_out => sig_out_vec(1),
--                     sig_vec => sig_in_vec1);
--          comp2 : test1
--            port map(sig_out => sig_out_vec(2),
--                     sig_vec => sig_in_vec2);
--        end test0;


entity test1 is
  port(signal sig_out : out bit := '0';
       signal sig_vec : in bit_vector(1 to %1%));
end test1;
architecture test1 of test1 is
begin
  process
    variable bit_var : bit := '0';
  begin
    for i in sig_vec'range loop
      bit_var := bit_var XOR sig_vec(i);
    end loop;
    sig_out <= bit_var;
    wait;
  end process;
end test1;
entity test0 is end;
architecture test0 of test0 is
  signal sig_out_vec : bit_vector(1 to %2%);
#2[  signal sig_in_vec@ : bit_vector(1 to %1%);]
  component test1
    port(signal sig_out : out bit;
```

```
        signal sig_vec : in bit_vector(1 to %1%));
  end component;
  for all : test1 use entity work.test1(test1);
begin
#2[  comp@ : test1
    port map(sig_out => sig_out_vec(@),
             sig_vec => sig_in_vec@);]
end test0;
```

TEST NUMBER : 220

PATHNAME : [.BENCH.B.C.F1.K.L3.P10]shell.sh
(UNIX equivalent : bench/b/c/f1/k/l3/p10/shell.sh)

PURPOSE : Determine the effect on CPU time when analyzing, model generating, building, and simulating the following model : two entities/ architectures, where one architecture is a component in the other architecture. The component-level entity's port clause contains one bit type signal (output) and one bit_vector type signal (input). The component-level architecture consists of a process. The process has two variable declarations (one is a loop-counter) and a while-loop whose number of iterations is equal to the size of the bit_vector in the port clause. The while-loop contains a logical XOR signal assignment statement and a variable assignment statement to increment the loop-counter. The top-level architecture consists of a bit-vector signal declaration for output, a number of bit_vector signal declarations for input, a component declaration matching the component-level entity/architecture, and a number of component instantiations, where the input signal maps to the corresponding input bit_vector signal declared above, and the output signal maps to one element of the output bit_vector signal declared above. The factors to be varied are the size of the input bit_vector in the port clause, and the number of components in the top-level architecture/size of output bit_vector signal in the top-level architecture/number of input bit_vector signal declarations in the top-level architecture.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 28 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size of bit_vector signal in port clause
--    2 : number of components in top-level architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test1 is
--          port(signal sig_out : out bit := '0';
--               signal sig_vec : in bit_vector(1 to 3));
--        end test1;
--        architecture test1 of test1 is
```

```
--        begin
--          process
--            variable bit_var : bit := '0';
--            variable loop_counter : integer := 1;
--          begin
--            while loop_counter <= sig_vec'high loop
--              bit_var := bit_var XOR sig_vec(loop_counter);
--              loop_counter := loop_counter + 1;
--            end loop;
--            sig_out <= bit_var;
--            wait;
--          end process;
--        end test1;
--      . entity test0 is end;
--        architecture test0 of test0 is
--          signal sig_out_vec : bit_vector(1 to 2);
--          signal sig_in_vec1 : bit_vector(1 to 3);
--          signal sig_in_vec2 : bit_vector(1 to 3);
--          component test1
--            port(signal sig_out : out bit;
--                 signal sig_vec : in bit_vector(1 to 3));
--          end component;
--          for all : test1 use entity work.test1(test1);
--        begin
--          comp1 : test1
--            port map(sig_out => sig_out_vec(1),
--                     sig_vec => sig_in_vec1);
--          comp2 : test1
--            port map(sig_out => sig_out_vec(2),
--                     sig_vec => sig_in_vec2);
--        end test0;


entity test1 is
  port(signal sig_out : out bit := '0';
       signal sig_vec : in bit_vector(1 to %1%));
end test1;
architecture test1 of test1 is
begin
  process
    variable bit_var : bit := '0';
    variable loop_counter : integer := 1;
  begin
    while loop_counter <= sig_vec'high loop
      bit_var := bit_var XOR sig_vec(loop_counter);
      loop_counter := loop_counter + 1;
    end loop;
    sig_out <= bit_var;
    wait;
  end process;
end test1;
```

```
entity test0 is end;
architecture test0 of test0 is
  signal sig_out_vec : bit_vector(1 to %2%);
#2[  signal sig_in_vec@ : bit_vector(1 to %1%);]
  component test1
    port(signal sig_out : out bit;
         signal sig_vec : in bit_vector(1 to %1%));
  end component;
  for all : test1 use entity work.test1(test1);
begin
#2[  comp@ : test1
    port map(sig_out => sig_out_vec(@),
             sig_vec => sig_in_vec@);]
end test0;
```

TEST NUMBER : 221

PATHNAME : [.BENCH.B.C.F1.K.L4.P10]shell.sh
(UNIX equivalent : bench/b/c/f1/k/l4/p10/shell.sh)

PURPOSE : Determine the effect on CPU time when analyzing, model generating, building, and simulating the following model : two entities/ architectures, where one architecture is a component in the other architecture. The component-level entity's port clause contains one bit type signal (output) and one bit_vector type signal (input). The component-level architecture consists of a process. The process has two variable declarations (one is a loop-counter) and a while-loop whose iteration scheme is "true". The while-loop contains a logical XOR signal assignment statement, a variable assignment statement to increment the loop-counter, and an exit statement that limits the number of iterations to the size of the bit_vector in the port clause. The top-level architecture consists of a bit-vector signal declaration for output, a number of bit_vector signal declarations for input, a component declaration matching the component-level entity/architecture, and a number of component instantiations, where the input signal maps to the corresponding input bit_vector signal declared above, and the output signal maps to one element of the output bit_vector signal declared above. The factors to be varied are the size of the input bit_vector in the port clause, and the number of components in the top-level architecture/size of output bit_vector signal in the top-level architecture/number of input bit_vector signal declarations in the top-level architecture.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 28 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size of bit_vector signal in port clause
--    2 : number of components in top-level architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test1 is
--          port(signal sig_out : out bit := '0';
--               signal sig_vec : in bit_vector(1 to 3));
--        end test1;
```

```
--        architecture test1 of test1 is
--        begin
--          process
--            variable bit_var : bit := '0';
--            variable loop_counter : integer := 1;
--          begin
--            while true loop
--              bit_var := bit_var XOR sig_vec(loop_counter);
--              loop_counter := loop_counter + 1;
--              exit when loop_counter > sig_vec'high;
--            end loop;
--            sig_out <= bit_var;
--            wait;
--          end process;
--        end test1;
--        entity test0 is end;
--        architecture test0 of test0 is
--          signal sig_out_vec : bit_vector(1 to 2);
--          signal sig_in_vec1 : bit_vector(1 to 3);
--          signal sig_in_vec2 : bit_vector(1 to 3);
--          component test1
--            port(signal sig_out : out bit;
--                 signal sig_vec : in bit_vector(1 to 3));
--          end component;
--          for all : test1 use entity work.test1(test1);
--        begin
--          comp1 : test1
--            port map(sig_out => sig_out_vec(1),
--                     sig_vec => sig_in_vec1);
--          comp2 : test1
--            port map(sig_out => sig_out_vec(2),
--                     sig_vec => sig_in_vec2);
--        end test0;


entity test1 is
  port(signal sig_out : out bit := '0';
       signal sig_vec : in bit_vector(1 to %1%));
end test1;
architecture test1 of test1 is
begin
  process
    variable bit_var : bit := '0';
    variable loop_counter : integer := 1;
  begin
    while true loop
      bit_var := bit_var XOR sig_vec(loop_counter);
      loop_counter := loop_counter + 1;
      exit when loop_counter > sig_vec'high;
    end loop;
    sig_out <= bit_var;
```

```
    wait;
  end process;
end test1;
entity test0 is end;
architecture test0 of test0 is
  signal sig_out_vec : bit_vector(1 to %2%);
#2[  signal sig_in_vec@ : bit_vector(1 to %1%);]
  component test1
    port(signal sig_out : out bit;
         signal sig_vec : in bit_vector(1 to %1%));
  end component;
  for all : test1 use entity work.test1(test1);
begin
#2[  comp@ : test1
    port map(sig_out => sig_out_vec(@),
             sig_vec => sig_in_vec@);]
end test0;
```

```
TEST NUMBER :  222

PATHNAME :  [.BENCH.B.C.F1.K.L2.P10]shell.sh
            (UNIX equivalent : bench/b/c/f1/k/l2/p10/shell.sh)

PURPOSE :  Determine the effect on CPU time when analyzing, model generating,
           building, and simulating the following model :  two entities/
           architectures, where one architecture is a component in the other
           architecture.  The component-level entity's port clause contains
           one bit type signal (output) and one bit_vector type signal (input).
           The component-level architecture consists of a process.  The process
           has a variable declaration and a for-loop whose iteration scheme is
           "1 to integer'high".  The for-loop contains a logical XOR signal
           assignment statement and an exit statement that limits the number of
           iterations to the size of the bit_vector in the port clause.  The
           top-level architecture consists of a bit-vector signal declaration
           for output, a number of bit_vector signal declarations for input, a
           component declaration matching the component-level entity/
           architecture, and a number of component instantiations, where the
           input signal maps to the corresponding input bit_vector signal
           declared above, and the output signal maps to one element of the
           output bit_vector signal declared above.  The factors to be varied
           are the size of the input bit_vector in the port clause, and the
           number of components in the top-level architecture/size of output
           bit_vector signal in the top-level architecture/number of input
           bit_vector signal declarations in the top-level architecture.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 28 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size of bit_vector signal in port clause
--    2 : number of components in top-level architecture
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test1 is
--          port(signal sig_out : out bit := '0';
--               signal sig_vec : in bit_vector(1 to 3));
--        end test1;
--        architecture test1 of test1 is
--        begin
```

```
--          process
--            variable bit_var : bit := '0';
--          begin
--            for i in 1 to integer'high loop
--              bit_var := bit_var XOR sig_vec(i);
--              exit when i = sig_vec'high;
--            end loop;
--            sig_out <= bit_var;
--            wait;
--          end process;
--        end test1;
--        entity test0 is end;
--        architecture test0 of test0 is
--          signal sig_out_vec : bit_vector(1 to 2);
--          signal sig_in_vec1 : bit_vector(1 to 3);
--          signal sig_in_vec2 : bit_vector(1 to 3);
--          component test1
--            port(signal sig_out : out bit;
--                 signal sig_vec : in bit_vector(1 to 3));
--          end component;
--          for all : test1 use entity work.test1(test1);
--        begin
--          comp1 : test1
--            port map(sig_out => sig_out_vec(1),
--                     sig_vec => sig_in_vec1);
--          comp2 : test1
--            port map(sig_out => sig_out_vec(2),
--                     sig_vec => sig_in_vec2);
--        end test0;


entity test1 is
  port(signal sig_out : out bit := '0';
       signal sig_vec : in bit_vector(1 to %1%));
end test1;
architecture test1 of test1 is
begin
  process
    variable bit_var : bit := '0';
  begin
    for i in 1 to integer'high loop
      bit_var := bit_var XOR sig_vec(i);
      exit when i = sig_vec'high;
    end loop;
    sig_out <= bit_var;
    wait;
  end process;
end test1;
entity test0 is end;
architecture test0 of test0 is
  signal sig_out_vec : bit_vector(1 to %2%);
```

```
#2[  signal sig_in_vec@ : bit_vector(1 to %1%);]
  component test1
    port(signal sig_out : out bit;
         signal sig_vec : in bit_vector(1 to %1%));
  end component;
  for all : test1 use entity work.test1(test1);
begin
#2[  comp@ : test1
    port map(sig_out => sig_out_vec(@),
             sig_vec => sig_in_vec@);]
end test0;
```

```
TEST NUMBER :  223

PATHNAME :  [.BENCH.B.C.M]shell.sh
            (UNIX equivalent : bench/b/c/m/shell.sh)

PURPOSE :  Determine the maximum number of "elsif" conditions allowed in an
           if-statement.  The model simulated is an architecture consisting
           of a signal declaration and a process.  The process consists of
           an if-statement with a number of "elsif" conditions, each containing
           a signal assignment statement.  The factor to be varied is the number
           of "elsif" conditions.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 28 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of if_then_else_statement conditions
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          subtype sig_type is integer range 1 to 3;
--          signal sig : sig_type;
--        begin
--          process
--          begin
--            if sig < sig_type'low then sig <= sig_type'low;
--            elsif sig = 1 then sig <= sig_type'low;
--            elsif sig = 2 then sig <= sig_type'high;
--            elsif sig = 3 then sig <= sig_type'low;
--            end if;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
  subtype sig_type is integer range 1 to %1%;
  signal sig : sig_type;
begin
```

```
  process
  begin
    if sig < sig_type'low then sig <= sig_type'low;
#1[    elsif sig = @ then sig <= $2$sig_type'low$sig_type'high$;]
    end if;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  224

PATHNAME :  [.BENCH.A.C.I2R]shell.sh
            (UNIX equivalent : bench/a/c/i2r/shell.sh)

PURPOSE :  Determine the effect of recursive function calls on simulation CPU
           time.  The model simulated is an architecture consisting of two
           signal declarations, a function declaration, and a function call
           via a signal assignment statement.  The function contains an if-
           statement where the "if" section has a return statement and the
           "else" section has a function call (recursive) to itself.  The factor
           to be varied is the number of recursive function call to make in one
           simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive function calls
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function summer(n : in natural) return natural is
--          begin
--            if n = 0 then return 0;
--            else return (n + summer(n-1));
--            end if;
--          end summer;
--          signal s1 : natural := 0;
--          signal s2 : natural := 10;
--        begin
--          s1 <= summer(s2);
--        end test;


entity test is end;
architecture test of test is
  function summer(n : in natural) return natural is
  begin
    if n = 0 then return 0;
```

```
    else return (n + summer(n-1));
    end if;
  end summer;
  signal s1 : natural := 0;
  signal s2 : natural := %1%;
begin
  s1 <= summer(s2);
end test;
```

TEST NUMBER : 225

PATHNAME : [.BENCH.A.C.I1R]shell.sh
(UNIX equivalent : bench/a/c/i1r/shell.sh)

PURPOSE : Determine the effect of recursive function calls on simulation CPU time. The model simulated is an entity consisting of a function declaration and an architecture consisting of two signal declarations and a function call via a signal assignment statement. The function contains an if-statement where the "if" section has a return statement and the "else" section has a function call (recursive) to itself. The factor to be varied is the number of recursive function call to make in one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive function calls
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function summer(n : in natural) return natural is
--          begin
--            if n = 0 then return 0;
--            else return (n + summer(n-1));
--            end if;
--          end summer;
--        end test;
--        architecture test of test is
--          signal s1 : natural := 0;
--          signal s2 : natural := 10;
--        begin
--          s1 <= summer(s2);
--        end test;


entity test is
  function summer(n : in natural) return natural is
  begin
    if n = 0 then return 0;
```

```
    else return (n + summer(n-1));
    end if;
  end summer;
end test;
architecture test of test is
  signal s1 : natural := 0;
  signal s2 : natural := %1%;
begin
  s1 <= summer(s2);
end test;
```

```
TEST NUMBER :  226

PATHNAME :  [.BENCH.B.C.H1R]shell.sh
            (UNIX equivalent : bench/b/c/h1r/shell.sh)

PURPOSE :  Determine the effect of recursive procedure calls on simulation
           CPU time.  The model simulated is an entity consisting of a
           procedure declaration and an architecture consisting of a signal
           declaration and a process.  The procedure has a variable declaration
           and an if-statement, where the "if" section has a return statement
           and the "else" section has a recursive procedure call and a variable
           assignment statement.  The process has a variable declaration, a
           procedure call, and a signal assignment statement.  The factor to be
           varied is the number of recursive procedure calls to make in one
           simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive procedure calls (must be >= 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure summer(n . in natural;
--                           s : inout natural)  is
--            variable temp_s : natural;
--          begin
--            if n = 0 then s := 0;
--            else
--              summer((n-1),temp_s);
--              s := n + temp_s;
--            end if;
--          end summer;
--        end test;
--        architecture test of test is
--          signal sig : natural;
--        begin
--          process
--            variable s : natural := 0;
--          begin
```

```
--            summer(10,s);
--            sig <= s;
--            wait;
--          end process;
--        end test;


entity test is
  procedure summer(n : in natural;
                   s : inout natural)  is
    variable temp_s : natural;
  begin
    if n = 0 then s := 0;
    else
      summer((n-1),temp_s);
      s := n + temp_s;
    end if;
  end summer;
end test;
architecture test of test is
  signal sig : natural;
begin
  process
    variable s : natural := 0;
  begin
    summer(%1%,s);
    sig <= s;
    wait;
  end process;
end test;
```

TEST NUMBER : 227

PATHNAME : [.BENCH.B.C.H2R]shell.sh
(UNIX equivalent : bench/b/c/h2r/shell.sh)

PURPOSE : Determine the effect of recursive procedure calls on simulation CPU time. The model simulated is an architecture consisting of a procedure declaration, a signal declaration and a process. The procedure has a variable declaration and an if-statement, where the "if" section has a return statement and the "else" section has a recursive procedure call and a variable assignment statement. The process has a variable declaration, a procedure call, and a signal assignment statement. The factor to be varied is the number of recursive procedure calls to make in one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive procedure calls (must be >= 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end test;
--        architecture test of test is
--          procedure summer(n : in natural;
--                           s : inout natural)  is
--            variable temp_s : natural;
--          begin
--            if n = 0 then s := 0;
--            else
--              summer((n-1),temp_s);
--              s := n + temp_s;
--            end if;
--          end summer;
--          signal sig : natural;
--        begin
--          process
--            variable s : natural := 0;
--          begin
--            summer(10,s);
--            sig <= s;
```

```
--            wait;
--          end process;
--        end test;


entity test is end test;
architecture test of test is
  procedure summer(n : in natural;
                   s : inout natural ) is
    variable temp_s : natural;
  begin
    if n = 0 then s := 0;
    else
      summer((n-1),temp_s);
      s := n + temp_s;
    end if;
  end summer;
  signal sig : natural;
begin
  process
    variable s : natural := 0;
  begin
    summer(%1%,s);
    sig <= s;
    wait;
  end process;
end test;
```

TEST NUMBER : 228

PATHNAME : [.BENCH.B.C.I3R]shell.sh
(UNIX equivalent : bench/b/c/i3r/shell.sh)

PURPOSE : Determine the effect of recursive function calls on simulation CPU time. The model simulated is an architecture consisting of a signal declaration and a process. The process has a function declaration and a function call via a signal assignment statement. The function has an if-statement, where the "if" section has a return statement and the "else" section has a recursive function call. The factor to be varied is the number of recursive function calls to make in one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive function calls
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal s1 : natural := 0;
--        begin
--          process
--            function summer(n : in natural) return natural is
--            begin
--              if n = 0 then return 0;
--              else return (n + summer(n-1));
--              end if;
--            end summer;
--          begin
--            s1 <= summer(10);
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
```

```
  signal s1 : natural := 0;
begin
  process
    function summer(n : in natural) return natural is
    begin
      if n = 0 then return 0;
      else return (n + summer(n-1));
      end if;
    end summer;
  begin
    s1 <= summer(%1%);
    wait;
  end process;
end test;
```

```
TEST NUMBER :  229

PATHNAME :  [.BENCH.B.C.H3R]shell.sh
            (UNIX equivalent : bench/b/c/h3r/shell.sh)

PURPOSE :   Determine the effect of recursive procedure calls on simulation
            CPU time.  The model simulated is an architecture consisting of
            a signal declaration and a process.  The process consists of a
            procedure declaration, a variable declaration, a procedure call,
            and a signal assignment statement.  The procedure has an if-
            statement where the "if" section has a variable assignment statement
            and the "else" section has a recursive procedure call and a
            variable assignment statement.  The factor to be varied is the number
            of recursive procedure calls to make in one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 19__
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive procedure calls (must be >= 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end test;
--        architecture test of test is
--          signal sig : natural;
--        begin
--          process
--            procedure summer(n : in natural;
--                             s : inout natural)  is
--              variable temp_s : natural;
--            begin
--              if n = 0 then s := 0;
--              else
--                summer((n-1),temp_s);
--                s := n + temp_s;
--              end if;
--            end summer;
--            variable s : natural := 0;
--          begin
--            summer(10,s);
--            sig <= s;
```

```
--            wait;
--          end process;
--        end test;


entity test is end test;
architecture test of test is
  signal sig : natural;
begin
  process
    procedure summer(n : in natural;
                     s : inout natural)  is
      variable temp_s : natural;
    begin
      if n = 0 then s := 0;
      else
        summer((n-1),temp_s);
        s := n + temp_s;
      end if;
    end summer;
    variable s : natural := 0;
  begin
    summer(%1%,s);
    sig <= s;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  230

PATHNAME :  [.BENCH.B.C.I1R]shell.sh
            (UNIX equivalent : bench/b/c/i1r/shell.sh)

PURPOSE :  Determine the effect of recursive function calls on simulation CPU
           time.  The model simulated is an entity consisting of a function
           declaration and an architecture consisting of a signal declaration
           and a process.  The function has an if-statement where the "if"
           section has a return statement and the "else" section has a recursive
           function call.  The process has a function call via a signal
           assignment statement.  The factor to be varied is the number of
           recursive function calls to make in one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive function calls
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function summer(n : in natural) return natural is
--          begin
--            if n = 0 then return 0;
--            else return (n + summer(n-1));
--            end if;
--          end summer;
--        end test;
--        architecture test of test is
--          signal s1 : natural := 0;
--        begin
--          process
--          begin
--            s1 <= summer(10);
--            wait;
--          end process;
--        end test;


entity test is
```

```
  function summer(n : in natural) return natural is
  begin
    if n = 0 then return 0;
    else return (n + summer(n-1));
    end if;
  end summer;
end test;
architecture test of test is
  signal s1 : natural := 0;
begin
  process
  begin
    s1 <= summer(%1%);
    wait;
  end process;
end test;
```

TEST NUMBER : 231

PATHNAME : [.BENCH.B.C.I2R]shell.sh
(UNIX equivalent : bench/b/c/i2r/shell.sh)

PURPOSE : Determine the effect of recursive function calls on simulation CPU time. The model simulated is an architecture consisting of a function declaration, a signal declaration, and a process. The function has an if-statement where the "if" section has a return statement and the "else" section has a recursive function call. The process has a function call via a signal assignment statement. The factor to be varied is the number of recursive function calls to make in one simulation.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number to sum/number of recursive function calls
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end test;
--        architecture test of test is
--          function summer(n : in natural) return natural is
--          begin
--            if n = 0 then return 0;
--            else return (n + summer(n-1));
--            end if;
--          end summer;
--          signal s1 : natural := 0;
--        begin
--          process
--          begin
--            s1 <= summer(10);
--            wait;
--          end process;
--        end test;


entity test is end test;
architecture test of test is
```

```
  function summer(n : in natural) return natural is
  begin
    if n = 0 then return 0;
    else return (n + summer(n-1));
    end if;
  end summer;
  signal s1 : natural := 0;
begin
  process
  begin
    s1 <= summer(%1%);
    wait;
  end process;
end test;
```

```
TEST NUMBER :  232

PATHNAME :  [.BENCH.A.C.H1.P12]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of integer
           variable declarations and a modulo statement for each variable.
           The factors to be varied are the number of variable declarations/
           modulo statements in the procedure and the value of the right
           operand of the modulo statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 30 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/modulus statements in procedure
--    2 : perform modulo with respect to this integer ("b" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",5,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,5,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure modulo is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--            variable var4 : integer := 4;
--            variable var5 : integer := 5;
--          begin
--            var1 := var1 mod 2;
--            var2 := var2 mod 2;
--            var3 := var3 mod 2;
--            var4 := var4 mod 2;
--            var5 := var5 mod 2;
--          end modulo;
--        end test;
--        architecture test of test is
--        begin
--          modulo;
--        end test;
```

```
entity test is
  procedure modulo is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ mod %2%;]
  end modulo;
end test;
architecture test of test is
begin
  modulo;
end test;
```

```
TEST NUMBER :  233

PATHNAME :  [.BENCH.A.C.H1.P12]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo operations in a procedure; determine the number of
           modulo operations allowed in one variable assignment statement of
           a procedure.  The model simulated is an entity consisting of a
           procedure declaration and an architecture consisting of a procedure
           call; the procedure consists of an integer variable declaration and
           a variable assignment statement containing a number of modulo
           operations.  The factors to be varied are the number of modulo
           operations contained in the variable assignment statement and the
           value of the left operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 30 June 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable modulus operations in one statement in procedure
--    2 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure modulo is
--            variable var : integer := 2;
--          begin
--            var := var
--                        mod (1
--                        mod (2
--                        mod (3
--                              )
--                              )
--                              );
--          end modulo;
--        end test;
--        architecture test of test is
--        begin
--          modulo;
--        end test;
```

```
entity test is
  procedure modulo is
    variable var : integer := %2%;
  begin
    var := var
#1[               mod (@]
#1[                    )];
  end modulo;
end test;
architecture test of test is
begin
  modulo;
end test;
```

```
TEST NUMBER :  234

PATHNAME :  [.BENCH.A.C.H2.P12]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and a modulo statement for each variable.  The factors to be varied
           are the number of variable declarations/modulo statements in the
           procedure and the value of the right operand in the modulo
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/modulus statements in procedure
--    2 : perform modulo with respect to this integer ("b" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",5,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,5,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure modulo is
--            variable var1 : integer := 1;
--            variable var2 : integer  = 2;
--            variable var3 : integer := 3;
--            variable var4 : integer := 4;
--            variable var5 : integer := 5;
--          begin
--            var1 := var1 mod 2;
--            var2 := var2 mod 2;
--            var3 := var3 mod 2;
--            var4 := var4 mod 2;
--            var5 := var5 mod 2;
--          end modulo;
--        begin
--          modulo;
--        end test;
```

```
entity test is end;
architecture test of test is
  procedure modulo is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ mod %2%;]
  end modulo;
begin
  modulo;
end test;
```

TEST NUMBER : 235

PATHNAME : [.BENCH.A.C.H2.P12]shell1.sh
(UNIX equivalent : bench/a/c/h2/p12/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute variable modulo operations in a procedure; determine the number of modulo operations allowed in one variable assignment statement of a procedure. The model simulated is an architecture consisting of a procedure declaration and a procedure call; the procedure consists of an integer variable declaration and a variable assignment statement containing a number of modulo operations. The factors to be varied are the number of modulo operations contained in the variable assignment statement and the value of the left operand of the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable modulus operations in one statement in procedure
--    2 : find modulo of this irteger ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",5,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,5,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--         entity test is end;
--         architecture test of test is
--           procedure modulo is
--             variable var : integer := 2;
--           begin
--             var := var
--                        mod (1
--                        mod (2
--                        mod (3
--                        mod (4
--                        mod (5)))));
--           end modulo;
--         begin
--           modulo;
--         end test;
```

```
entity test is end;
architecture test of test is
  procedure modulo is
    variable var : integer := %2%;
  begin
    var := var
#1[                mod (@]#1[-)];
  end modulo;
begin
  modulo;
end test;
```

```
TEST NUMBER :  236

PATHNAME :  [.BENCH.A.C.I1.P12]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo statements.  The model simulated is an entity consisting
           of a function declaration and an architecture consisting of a
           function call.  The function consists of a number of integer
           variable declarations and a modulo statement for each variable.
           The factors to be varied are the number of variable declarations/
           modulo statements in the function and the value of the right operand
           in the modulo statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable modulus statements in
--        function
--    2 : perform modulo with respect to this integer ("b" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function modulo return boolean is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 mod 2;
--            var2 := var2 mod 2;
--            var3 := var3 mod 2;
--            return true;
--          end modulo;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= modulo;
--        end test;
```

```
entity test is
  function modulo return boolean is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ mod %2%;]
    return true;
  end modulo;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= modulo;
end test;
```

```
TEST NUMBER :  237

PATHNAME :  [.BENCH.A.C.I1.P12]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo operations in a function; determine the number of
           modulo operations allowed in one variable assignment
           statement of a function.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call; the function consists of an integer variable
           declaration and a variable assignment statement containing a number
           of modulo operations.  The factors to be varied are the number of
           modulo operations contained in the variable assignment statement
           and the value of the left operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable modulus operations in one statement in function
--    2 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function modulo return boolean is
--            variable var : integer := 2;
--          begin
--            var := var
--                       mod (1
--                       mod (2
--                       mod (3)));
--            return true;
--          end modulo;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= modulo;
--        end test;
```

```
entity test is
  function modulo return boolean is
    variable var : integer := %2%;
  begin
    var := var
#1[               mod (@]#1[~)];
    return true;
  end modulo;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= modulo;
end test;
```

```
TEST NUMBER :  238

PATHNAME :  [.BENCH.A.C.I2.P12]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and a modulo statement for each variable.  The factors to be varied
           are the number of variable declarations/modulo statements in the
           function and the value of the right operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable modulus statements in
--        function
--    2 : perform modulo with respect to this integer ("b" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function modulo return boolean is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 mod 2;
--            var2 := var2 mod 2;
--            var3 := var3 mod 2;
--            return true;
--          end modulo;
--          signal done : boolean := false;
--        begin
--          done <= modulo;
--        end test;


entity test is end;
```

```
architecture test of test is
  function modulo return boolean is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ mod %2%;]
    return true;
  end modulo;
  signal done : boolean := false;
begin
  done <= modulo;
end test;
```

```
TEST NUMBER :  239

PATHNAME :  [.BENCH.A.C.I2.P12]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo operations in a function; determine the number of
           modulo operations allowed in one variable assignment
           statement of a function.  The model simulated is an architecture
           consisting of a function declaration and a function call; the
           function consists of an integer variable declaration and a variable
           assignment statement containing a number of modulo operations.
           The factors to be varied are the number of modulo operations
           contained in the variable assignment statement and the value of the
           left operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable modulus operations in one statement in function
--    2 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        end test;
--        entity test is end;
--        architecture test of test is
--          function modulo return boolean is
--            variable var : integer := 2;
--          begin
--            var := var
--                       mod (1
--                       mod (2
--                       mod (3)));
--            return true;
--          end modulo;
--          signal done : boolean := false;
--        begin
--          done <= modulo;
--        end test;
```

```
entity test is end;
architecture test of test is
  function modulo return boolean is
    variable var : integer := %2%;
  begin
    var := var
#1[                mod (@]#1[~)];
    return true;
  end modulo;
  signal done : boolean := false;
begin
  done <= modulo;
end test;
```

```
TEST NUMBER :  240

PATHNAME :  [.BENCH.A.C.P12]shell0.sh
            (UNIX equivalent : bench/a/c/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute modulo
           operations on signals.  The model simulated consists of a number of
           signal declarations and one modulo signal assignment statement
           for each signal.  The factors to be varied are the number of signal
           declarations/number of modulo signal assignment statements, the
           value of the right operand in the modulo statement, and the length
           of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/modulus statements
--    2 : perform modulo with respect to this integer ("b" in "a mod b")
--    3 : length of time (ns) to simulate model (> 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := 1;
--          signal sig2 : integer := 2;
--          signal sig3 : integer := 3;
--        begin
--          sig1 <= sig1 mod 2 after 1 ns;
--          sig2 <= sig2 mod 2 after 1 ns;
--          sig3 <= sig3 mod 2 after 1 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;



entity test is end;
architecture test of test is
  signal stop : bit := '0';
```

```
#1[  signal sig@ : integer := @;]
begin
#1[  sig@ <= sig@ mod %2% after 1 ns;]
  stop <= '1' after %3% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  241

PATHNAME :  [.BENCH.A.C.P12]shell1.sh
            (UNIX equivalent : bench/a/c/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform modulo
           operations on a signal; determine the number of modulo operations
           allowed in one signal assignment statement.  The model simulated is
           an architecture consisting of an integer signal declaration and a
           signal assignment statement containing a number of modulo
           operations.  The factors to be varied are the number of modulo
           operations contained in the signal assignment statement, the value
           of the right operands in the modulo statement, and the length of time
           (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of modulus operations in one statement
--    2 : find modulo of this integer ("a" in "a mod b")
--    3 : length of time (ns) to simulate model (> 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : integer := 2;
--          signal stop : bit := '0';
--        begin
--          sig <= sig
--                     mod (1
--                     mod (2
--                     mod (3))) after 1 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal sig : integer := %2%;
```

```
  signal stop : bit := '0';
begin
  sig <= sig
#1[             mod (@]#1[~)] after 1 ns;
  stop <= '1' after %3% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  242

PATHNAME :  [.BENCH.B.C.K.L1.P12]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           modulo assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an integer array variable declaration and a for-loop.
           The number of iterations of the loop is equal to the size of the
           array.  The for-loop contains a variable modulo assignment
           statement.  The factors to be varied are the number of processes,
           the array size/number of loop iterations, and the value of the left
           operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of modulus statement iterations per process
--    2 : number of processes
--    3 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,6
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,6)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 3 loop
--              var(i) := 6 mod i;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 3 loop
```

```
--              var(i) := 6 mod i;
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
  begin
    for i in 1 to %1% loop
      var(i) := %3% mod i;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  243

PATHNAME :  [.BENCH.B.C.K.L1.P12]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute modulo
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           for-loop.  The number of iterations of the loop is equal to the
           size of the array.  The for-loop contains a modulo signal
           assignment statement.  The factors to be varied are the array size/
           number of loop iterations and the value of the left operand in the
           modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of modulus statement iterations in process
--    2 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig : sig_array;
--        begin
--          process
--          begin
--            for i in 1 to 3 loop
--              sig(i) <= 2 mod i;
--            end loop;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
```

```
  signal sig : sig_array;
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= %2% mod i;
    end loop;
    wait;
  end process;
end test;
```

TEST NUMBER : 244

PATHNAME : [.BENCH.B.C.K.L3.P12]shell0.sh
(UNIX equivalent : bench/b/c/k/l3/p12/shell0.sh)

PURPOSE : Determine the simulation CPU time required to execute addition and modulo variable assignment statements. The model simulated is an architecture consisting of a process. The process consists of two integer variable declarations (one is a loop-counter), one integer array variable declaration, and a while-loop. The number of iterations of the loop is equal to the size of the array. The while-loop contains an addition variable assignment statement to increment the loop-counter and a modulo variable assignment statement. The factors to be varied are the array size/number of loop iterations and the value of the left operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of modulus statement iterations in process
--    2 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := 2 mod loop_counter;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
  process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := %2% mod loop_counter;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  245

PATHNAME :  [.BENCH.B.C.K.L3.P12]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute modulo
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           while-loop and an integer variable declaration (loop-counter).  The
           number of iterations of the loop is equal to the size of the array.
           The while-loop contains one modulo signal assignment statement
           and one addition variable assignment statement to increment the
           loop-counter.  The factors to be varied are the array size/number of
           loop iterations and the value of the left operand in the modulo
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of modulus statement iterations in process
--    2 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig : sig_array;
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= 2 mod loop_counter;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= %2% mod loop_counter;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  246

PATHNAME :  [.BENCH.B.C.P12]shell0.sh
            (UNIX equivalent : bench/b/c/p12/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing modulo
           operations on variables.  The model simulated is an architecture
           consisting of a number of processes; each process consists of a
           number of variable declarations and one modulo variable assignment
           statement for each variable.  The factors to be varied are the
           number of processes, the number of variable declarations/number of
           modulo variable assignment statements, and the value of the left
           operand in the modulo statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/modulus statements per process
--    2 : number of processes
--    3 : perform modulo with respect to this integer ("b" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 mod 2;
--            var2 := var2 mod 2;
--            var3 := var3 mod 2;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
```

```
--            var1 := var1 mod 2;
--            var2 := var2 mod 2;
--            var3 := var3 mod 2;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ mod %3%;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  247

PATHNAME :  [.BENCH.B.C.P12]shell1.sh
            (UNIX equivalent : bench/b/c/p12/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute modulo
           operations on variables;  determine the number of modulo
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of modulo operations.
           The factors to be varied are the number of processes, the number
           of modulo operations in the variable assignment statement, and the
           value of the left operand in the modulo statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of modulus operations in one statement of a process
--    2 : number of processes
--    3 : find modulo of this integer ("a" in "a mod b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : integer := 2;
--          begin
--            var := var
--                       mod (1
--                       mod (2
--                       mod (3)));
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : integer := 2;
--          begin
--            var := var
--                       mod (1
```

```
--                          mod (2
--                          mod (3)));
--              wait;
--            end process pr2;
--          end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
     variable var : integer := %3%;
  begin
     var := var
#1[                mod (4]#1[~)];
     wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  248

PATHNAME :  [.BENCH.A.C.H1.P13]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder statements.  The model simulated is an entity consisting
           of a procedure declaration and an architecture consisting of a
           procedure call.  The procedure consists of a number of integer
           variable declarations and a remainder statement for each variable.
           The factors to be varied are the number of variable declarations/
           remainder statements in the procedure and the value of the right
           operand of the remainder statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/remainder statements in procedure
--    2 : perform REM with respect to this integer ("b" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure remainder is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 REM 2;
--            var2 := var2 REM 2;
--            var3 := var3 REM 2;
--          end remainder;
--        end test;
--        architecture test of test is
--        begin
--          remainder;
--        end test;


entity test is
  procedure remainder is
```

```
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ REM %2%;]
  end remainder;
end test;
architecture test of test is
begin
  remainder;
end test;
```

```
TEST NUMBER :  249

PATHNAME :  [.BENCH.A.C.H1.P13]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder operations in a procedure; determine the number of
           remainder operations allowed in one variable assignment statement of
           a procedure.  The model simulated is an entity consisting of a
           procedure declaration and an architecture consisting of a procedure
           call; the procedure consists of an integer variable declaration and
           a variable assignment statement containing a number of remainder
           operations.  The factors to be varied are the number of remainder
           operations contained in the variable assignment statement and the
           value of the left operand in the remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable remainder operations in one statement in procedure
--    2 : find remainder of this integer ("a" in "a rem b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure remainder is
--            variable var : integer := 2;
--          begin
--            var := var
--                       REM (1
--                       REM (2
--                       REM (3)));
--          end remainder;
--        end test;
--        architecture test of test is
--        begin
--          remainder;
--        end test;


entity test is
```

```
  procedure remainder is
    variable var : integer := %2%;
  begin
    var := var
#1[                REM (@]#1[~)];
  end remainder;
end test;
architecture test of test is
begin
  remainder;
end test;
```

```
TEST NUMBER :  250

PATHNAME :  [.BENCH.A.C.H2.P13]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and a remainder statement for each variable.  The factors to be
           varied are the number of variable declarations/remainder statements
           in the procedure and the value of the right operand in the remainder
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/remainder statements in procedure
--    2 : perform REM with respect to this integer ("b" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure remainder is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 REM 2;
--            var2 := var2 REM 2;
--            var3 := var3 REM 2;
--          end remainder;
--        begin
--          remainder;
--        end test;


entity test is end;
architecture test of test is
  procedure remainder is
```

```
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ REM %2%;]
  end remainder;
begin
  remainder;
end test;
```

```
TEST NUMBER :  251

PATHNAME :  [.BENCH.A.C.H2.P13]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder operations in a procedure; determine the number of
           remainder operations allowed in one variable assignment statement of
           a procedure.  The model simulated is an architecture consisting of a
           procedure declaration and a procedure call; the procedure consists
           of an integer variable declaration and a variable assignment
           statement containing a number of remainder operations.  The factors
           to be varied are the number of remainder operations contained in the
           variable assignment statement and the value of the left operand of
           the remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 3 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable remainder operations in one statement in procedure
--    2 : find remainder of this integer ("a" in "a rem b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture test of test is
--          procedure remainder is
--            variable var : integer := 2;
--          begin
--            var := var
--                       REM (1
--                       REM (2
--                       REM (3)));
--          end remainder;
--        begin
--          remainder;
--        end test;


entity test is
```

```
end test;
architecture test of test is
  procedure remainder is
    variable var : integer := %2%;
  begin
    var := var
#1[               REM (@]#1[~)];
  end remainder;
begin
  remainder;
end test;
```

```
TEST NUMBER :  252

PATHNAME :  [.BENCH.A.C.I1.P13]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder statements.  The model simulated is an entity consisting
           of a function declaration and an architecture consisting of a
           function call.  The function consists of a number of integer
           variable declarations and a remainder statement for each variable.
           The factors to be varied are the number of variable declarations/
           remainder statements in the function and the value of the right
           operand in the remainder statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable remainder statements in
--        function
--    2 : perform REM with respect to this integer ("b" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function remainder return boolean is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 REM 2;
--            var2 := var2 REM 2;
--            var3 := var3 REM 2;
--            return true;
--          end remainder;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= remainder;
--        end test;
```

```
entity test is
  function remainder return boolean is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ REM %2%;]
    return true;
  end remainder;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= remainder;
end test;
```

```
TEST NUMBER :  253

PATHNAME :  [.BENCH.A.C.I1.P13]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder operations in a function; determine the number of
           remainder operations allowed in one variable assignment
           statement of a function.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call; the function consists of an integer variable
           declaration and a variable assignment statement containing a number
           of remainder operations.  The factors to be varied are the number of
           remainder operations contained in the variable assignment statement
           and the value of the left operand in the remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable remainder operations in one statement in function
--    2 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function remainder return integer is
--            variable var : integer := 2;
--          begin
--            var := var
--                       REM (1
--                       REM (2
--                       REM (3)));
--            return var;
--          end remainder;
--        end test;
--        architecture test of test is
--          signal answer : integer := 0;
--        begin
--          answer <= remainder;
--        end test;
```

```
entity test is
  function remainder return integer is
    variable var : integer := %2%;
  begin
    var := var
#1[                REM (@]#1[~)];
    return var;
  end remainder;
end test;
architecture test of test is
  signal answer : integer := 0;
begin
  answer <= remainder;
end test;
```

```
TEST NUMBER :  254

PATHNAME :  [.BENCH.A.C.I2.P13]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and a remainder statement for each variable.  The factors to be
           varied are the number of variable declarations/remainder statements
           in the function and the value of the right operand in the remainder
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/variable remainder statements in
--        function
--    2 : perform REM with respect to this integer ("b" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function remainder return boolean is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 REM 2;
--            var2 := var2 REM 2;
--            var3 := var3 REM 2;
--            return true;
--          end remainder;
--          signal done : boolean := false;
--        begin
--          done <= remainder;
--        end test;
```

```
entity test is end;
architecture test of test is
  function remainder return boolean is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ REM %2%;]
    return true;
  end remainder;
  signal done : boolean := false;
begin
  done <= remainder;
end test;
```

```
TEST NUMBER :  255

PATHNAME :  [.BENCH.A.C.I2.P13]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           remainder operations in a function; determine the number of
           remainder operations allowed in one variable assignment
           statement of a function.  The model simulated is an architecture
           consisting of a function declaration and a function call; the
           function consists of an integer variable declaration and a variable
           assignment statement containing a number of remainder operations.
           The factors to be varied are the number of remainder operations
           contained in the variable assignment statement and the value of the
           left operand in the remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable remainder operations in one statement in function
--    2 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture test of test is
--          function remainder return integer is
--            variable var : integer := 2;
--          begin
--            var := var
--                       REM (1
--                       REM (2
--                       REM (3)));
--            return var;
--          end remainder;
--          signal answer : integer := 0;
--        begin
--          answer <= remainder;
--        end test;
```

```
entity test is
end test;
architecture test of test is
  function remainder return integer is
    variable var : integer := %2%;
  begin
    var := var
#1[               REM (@]#1[~)];
    return var;
  end remainder;
  signal answer : integer := 0;
begin
  answer <= remainder;
end test;
```

```
TEST NUMBER :  256

PATHNAME :  [.BENCH.A.C.P13]shell0.sh
            (UNIX equivalent : bench/a/c/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute remainder
           operations on signals.  The model simulated consists of a number of
           signal declarations and one remainder signal assignment statement
           for each signal.  The factors to be varied are the number of signal
           declarations/number of remainder signal assignment statements, the
           value of the right operand in the remainder statement, and the length
           of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/remainder statements
--    2 : perform REM with respect to this integer ("b" in "a REM b")
--    3 : length of time (ns) to simulate model (> 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := 1;
--          signal sig2 : integer := 2;
--          signal sig3 : integer := 3;
--        begin
--          sig1 <= sig1 REM 2 after 1 ns;
--          sig2 <= sig2 REM 2 after 1 ns;
--          sig3 <= sig3 REM 2 after 1 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : integer := @;]
```

```
begin
#1[  sig@ <= sig@ REM %2% after 1 ns;]
  stop <= '1' after %3% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  257

PATHNAME :  [.BENCH.A.C.P13]shell1.sh
            (UNIX equivalent : bench/a/c/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform remainder
           operations on a signal; determine the number of remainder operations
           allowed in one signal assignment statement.  The model simulated is
           an architecture consisting of an integer signal declaration and a
           signal assignment statement containing a number of remainder
           operations.  The factors to be varied are the number of remainder
           operations contained in the signal assignment statement, the value
           of the left operand in the remainder statement, and the length of
           time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of remainder operations in one statement
--    2 : find REM of this integer ("a" in "a REM b")
--    3 : length of time (ns) to simulate model (> 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig : integer := 2;
--          signal stop : bit := '0';
--        begin
--          sig <= sig
--                     REM (1
--                     REM (2
--                     REM (3))) after 1 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal sig : integer := %2%;
```

```
  signal stop : bit := '0';
begin
  sig <= sig
#1[             REM (@]#1[~)] after 1 ns;
  stop <= '1' after %3% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

TEST NUMBER : 258

PATHNAME : [.BENCH.B.C.K.L1.P13]shell0.sh
(UNIX equivalent : bench/b/c/k/l1/p13/shell0.sh)

PURPOSE : Determine the simulation CPU time required to execute variable remainder assignment statements. The model simulated is an architecture consisting of a number of processes. Each process consists of an integer array variable declaration and a for-loop. The number of iterations of the loop is equal to the size of the array. The for-loop contains a variable remainder assignment statement. The factors to be varied are the number of processes, the array size/number of loop iterations, and the value of the left operand in the remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of remainder statement iterations per process
--    2 : number of processes
--    3 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,6
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,6)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 3 loop
--              var(i) := 6 REM i;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 3 loop
```

```
--              var(i) := 6 REM i;
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
  begin
    for i in 1 to %1% loop
      var(i) := %3% REM i;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  259

PATHNAME :  [.BENCH.B.C.K.L1.P13]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute remainder
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           for-loop.  The number of iterations of the loop is equal to the
           size of the array.  The for-loop contains a remainder signal
           assignment statement.  The factors to be varied are the array size/
           number of loop iterations and the value of the left operand in the
           remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 6 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of remainder statement iterations in process
--    2 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig : sig_array;
--        begin
--          process
--          begin
--            for i in 1 to 3 loop
--              sig(i) <= 2 REM i;
--            end loop;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
```

```
  signal sig : sig_array;
begin
  process
  begin
    for i in 1 to %1% loop
      sig(i) <= %2% REM i;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  260

PATHNAME :  [.BENCH.B.C.K.L3.P13]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           and remainder variable assignment statements.  The model simulated
           is an architecture consisting of a process.  The process consists of
           two integer variable declarations (one is a loop-counter), one
           integer array variable declaration, and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and a remainder variable assignment
           statement.  The factors to be varied are the array size/number of
           loop iterations and the value of the left operand in the remainder
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : array size/number of remainder statement iterations in process
--    2 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := 2 REM loop_counter;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
begin
  process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := %2% REM loop_counter;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  261

PATHNAME :  [.BENCH.B.C.K.L3.P13]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute remainder
           signal assignment statements.  The model simulated is an
           architecture consisting of a process and two signal declarations,
           one integer and one integer array.  The process consists of a
           while-loop and an integer variable declaration (loop-counter).  The
           number of iterations of the loop is equal to the size of the array.
           The while-loop contains one remainder signal assignment statement
           and one addition variable assignment statement to increment the
           loop-counter.  The factors to be varied are the array size/number of
           loop iterations and the value of the left operand in the remainder
           statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : signal array size/number of remainder statement iterations in process
--    2 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig : sig_array;
--        begin
--          process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig(loop_counter) <= 2 REM loop_counter;
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
  signal sig : sig_array;
begin
  process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig(loop_counter) <= %2% REM loop_counter;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  262

PATHNAME :  [.BENCH.B.C.P13]shell0.sh
            (UNIX equivalent : bench/b/c/p13/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing remainder
           operations on variables.  The model simulated is an architecture
           consisting of a number of processes; each process consists of a
           number of variable declarations and one remainder variable
           assignment statement for each variable.  The factors to be varied
           are the number of processes, the number of variable declarations/
           number of remainder variable assignment statements, and the value of
           the right operand in the remainder statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/remainder statements per process
--    2 : number of processes
--    3 : perform REM with respect to this integer ("b" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 REM 2;
--            var2 := var2 REM 2;
--            var3 := var3 REM 2;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
```

```
--            var1 := var1 REM 2;
--            var2 := var2 REM 2;
--            var3 := var3 REM 2;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ REM %3%;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  263

PATHNAME :  [.BENCH.B.C.P13]shell1.sh
            (UNIX equivalent : bench/b/c/p13/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute remainder
           operations on variables;  determine the number of remainder
           operations allowed in one variable assignment statement.  The model
           simulated is an architecture consisting of a number of processes;
           each process consists of a variable declaration and a variable
           assignment statement containing a number of remainder operations.
           The factors to be varied are the number of processes, the number
           of remainder operations in the variable assignment statement, and the
           value of the left operand in the remainder statement.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of remainder operations in one statement of a process
--    2 : number of processes
--    3 : find REM of this integer ("a" in "a REM b")
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var : integer := 2;
--          begin
--            var := var
--                       REM (1
--                       REM (2
--                       REM (3)));
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var : integer := 2;
--          begin
--            var := var
--                       REM (1
```

```
--                        REM (2
--                        REM (3)));
--             wait;
--           end process pr2;
--         end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    variable var : integer := %3%;
  begin
    var := var
#1[                REM (@]#1[^)];
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  264

PATHNAME :  [.BENCH.A.C.H1.P14]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call.  The procedure consists of a number of integer
           variable declarations and an absolute value statement for each
           variable.  The factor to be varied is the number of variable
           declarations/absolute value statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/absolute value statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure abs_val is
--            variable var1 : integer := -1;
--            variable var2 : integer :=  2;
--            variable var3 : integer := -3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--          end abs_val;
--        end test;
--        architecture test of test is
--        begin
--          abs_val;
--        end test;


entity test is
  procedure abs_val is
#1[    variable var@ : integer := $2$-$ $@;]
```

```
  begin
#1[    var@ := ABS var@;]
  end abs_val;
end test;
architecture test of test is
begin
  abs_val;
end test;
```

```
TEST NUMBER :  265

PATHNAME :  [.BENCH.A.C.H1.P14]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call.  The procedure consists of a number of
           floating-point variable declarations and an absolute value statement
           for each variable.  The factor to be varied is the number of
           variable declarations/absolute value statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure abs_val is
--            variable var1 : real := -1.1;
--            variable var2 : real :=  2.2;
--            variable var3 : real := -3.3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--          end abs_val;
--        end test;
--        architecture test of test is
--        begin
--          abs_val;
--        end test;


entity test is
  procedure abs_val is
#1[    variable var@ : real := $2$-$ $@.@;]
```

```
  begin
#1[    var@ := ABS var@;]
  end abs_val;
end test;
architecture test of test is
begin
  abs_val;
end test;
```

```
TEST NUMBER :  266

PATHNAME :  [.BENCH.A.C.H2.P14]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and an absolute value statement for each variable.  The factor to be
           varied is the number of variable declarations/absolute value
           statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/absolute value statements in
--        procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture test of test is
--          procedure abs_val is
--            variable var1 : integer := -1;
--            variable var2 : integer :=  2;
--            variable var3 : integer := -3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--          end abs_val;
--        begin
--          abs_val;
--        end test;


entity test is
end test;
architecture test of test is
```

```
  procedure abs_val is
#1[    variable var@ : integer := $2$-$ $@;]
  begin
#1[    var@ := ABS var@;]
  end abs_val;
begin
  abs_val;
end test;
```

```
TEST NUMBER :  267

PATHNAME :  [.BENCH.A.C.H2.P14]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of floating-point variable
           declarations and an absolute value statement for each variable.  The
           factor to be varied is the number of variable declarations/absolute
           value statements in the procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements in procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--        end test;
--        architecture test of test is
--          procedure abs_val is
--            variable var1 : real := -1.1;
--            variable var2 : real :=  2.2;
--            variable var3 : real := -3.3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--          end abs_val;
--        begin
--          abs_val;
--        end test;


entity test is
end test;
architecture test of test is
```

```
  procedure abs_val is
#1[    variable var@ : real := $2$-$ $@.@;]
  begin
#1[    var@ := ABS var@;]
  end abs_val;
begin
  abs_val;
end test;
```

```
TEST NUMBER :  268

PATHNAME :  [.BENCH.A.C.I1.P14]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of integer
           variable declarations and an absolute value statement for each
           variable.  The factor to be varied is the number of variable
           declarations/absolute value statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/absolute value statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function abs_val return boolean is
--            variable var1 : integer := -1;
--            variable var2 : integer :=  2;
--            variable var3 : integer .= -3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--            return true;
--          end abs_val;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= abs_val;
--        end test;


entity test is
```

```
  function abs_val return boolean is
#1[    variable var@ : integer := $2$-$ $@;]
  begin
#1[    var@ := ABS var@;]
    return true;
  end abs_val;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= abs_val;
end test;
```

```
TEST NUMBER :  269

PATHNAME :  [.BENCH.A.C.I1.P14]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of
           floating-point variable declarations and an absolute value statement
           for each variable.  The factor to be varied is the number of
           variable declarations/absolute value statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function abs_val return boolean is
--            variable var1 : real := -1.1;
--            variable var2 : real :=  2.2;
--            variable var3 : real := -3.3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--            return true;
--          end abs_val;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= abs_val;
--        end test;


entity test is
```

```
  function abs_val return boolean is
#1[    variable var@ : real := $2$-$ $@.@;]
  begin
#1[    var@ := ABS var@;]
    return true;
  end abs_val;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= abs_val;
end test;
```

```
TEST NUMBER :  270

PATHNAME :  [.BENCH.A.C.I2.P14]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and an absolute value statement for each variable.  The factor to be
           varied is the number of variable declarations/absolute value
           statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/absolute value statements in
--        function
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function abs_val return boolean is
--            variable var1 : integer := -1;
--            variable var2 : integer :=  2;
--            variable var3 : integer := -3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--            return true;
--          end abs_val;
--          signal done : boolean := false;
--        begin
--          done <= abs_val;
--        end test;


entity test is end;
architecture test of test is
```

```
  function abs_val return boolean is
#1[    variable var@ : integer := $2$-$ $@;]
  begin
#1[    var@ := ABS var@;]
    return true;
  end abs_val;
  signal done : boolean := false;
begin
  done <= abs_val;
end test;
```

```
TEST NUMBER :  271

PATHNAME :  [.BENCH.A.C.I2.P14]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of floating-point variable declarations
           and an absolute value statement for each variable.  The factor to be
           varied is the number of variable declarations/absolute value
           statements in the function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements in function
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function abs_val return boolean is
--            variable var1 : real := -1.1;
--            variable var2 : real :=  2.2;
--            variable var3 : real := -3.3;
--          begin
--            var1 := ABS var1;
--            var2 := ABS var2;
--            var3 := ABS var3;
--            return true;
--          end abs_val;
--          signal done : boolean := false;
--        begin
--          done <= abs_val;
--        end test;


entity test is end;
architecture test of test is
```

```
  function abs_val return boolean is
#1[    variable var@ : real := $2$-$ $@.@;]
  begin
#1[    var@ := ABS var@;]
    return true;
  end abs_val;
  signal done : boolean := false;
begin
  done <= abs_val;
end test;
```

```
TEST NUMBER :  272

PATHNAME :  [.BENCH.A.C.P14]shell0.sh
            (UNIX equivalent : bench/a/c/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute absolute value
           operations on signals.  The model simulated consists of a number of
           integer signal declarations and one absolute value signal assignment
           statement for each signal.  The factor to be varied is the number of
           signal declarations/number of absolute value signal assignment
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer signal declarations/absolute value statements
--    2 : length of time (ns) to simulate model (> 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : integer := -1;
--          signal sig2 : integer :=  2;
--          signal sig3 : integer := -3;
--        begin
--          sig1 <= ABS sig1 after 1 ns;
--          sig2 <= ABS sig2 after 1 ns;
--          sig3 <= ABS sig3 after 1 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : integer := $2$-$ $@;]
begin
#1[  sig@ <= ABS sig@ after 1 ns;]
```

```
  stop <= '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  273

PATHNAME :  [.BENCH.A.C.P14]shell1.sh
            (UNIX equivalent : bench/a/c/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute absolute value
           operations on signals.  The model simulated consists of a number of
           floating-point signal declarations and one absolute value signal
           assignment statement for each signal.  The factor to be varied is
           the number of signal declarations/number of absolute value signal
           assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point signal declarations/absolute value statements
--    2 : length of time (ns) to simulate model (> 0)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal sig1 : real := -1.1;
--          signal sig2 : real :=  2.2;
--          signal sig3 : real := -3.3;
--        begin
--          sig1 <= ABS sig1 after 1 ns;
--          sig2 <= ABS sig2 after 1 ns;
--          sig3 <= ABS sig3 after 1 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;


entity test is end;
architecture test of test is
  signal stop : bit := '0';
#1[  signal sig@ : real := $2$-$ $@.@;]
begin
#1[  sig@ <= ABS sig@ after 1 ns;]
```

```
  stop <= '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

TEST NUMBER : 274

PATHNAME : [.BENCH.B.C.K.L1.P14]shell0.sh
(UNIX equivalent : bench/b/c/k/l1/p14/shell0.sh)

PURPOSE : Determine the simulation CPU time required to execute variable absolute value assignment statements. The model simulated is an architecture consisting of a number of processes. Each process consists of an integer array variable declaration and a for-loop. The number of iterations of the loop is equal to the size of the array. The for-loop contains a variable absolute value assignment statement. The factors to be varied are the number of processes and the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 5 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer array size/number of absolute value statement iterations per
--        process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 3 loop
--              var(i) := ABS (-i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--          begin
--            for i in 1 to 3 loop
--              var(i) := ABS (-i);
```

```
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
  begin
    for i in 1 to %1% loop
      var(i) := ABS (-i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  275

PATHNAME :  [.BENCH.B.C.K.L1.P14]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           absolute value assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an floating-point array variable declaration and a
           for-loop.  The number of iterations of the loop is equal to the size
           of the array.  The for-loop contains a variable absolute value
           assignment statement.  The factors to be varied are the number of
           processes and the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point array size/number of absolute value statement
--        iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            constant flt_const : real := - 3.3;
--          begin
--            for i in 1 to 3 loop
--              var(i) := ABS (flt_const);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            constant flt_const : real := - 3.3;
--          begin
```

```
--            for i in 1 to 3 loop
--              var(i) := ABS (flt_const);
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of real;
    variable var : var_array;
    constant flt_const : real := - %1%.%1%;
  begin
    for i in 1 to %1% loop
      var(i) := ABS (flt_const);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  276

PATHNAME :  [.BENCH.B.C.K.L1.P14]shell2.sh
            (UNIX equivalent : bench/b/c/k/l1/p14/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute signal
           absolute value assignment statements.  The model simulated is an
           architecture consisting of a number of integer array signal
           declarations and the same number of processes.  Each process
           consists of a for-loop.  The number of iterations of the loop is
           equal to the size of the array.  The for-loop contains a signal
           absolute value assignment statement.  The factors to be varied are
           the number of processes/number of array declarations and the array
           size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer signal array size/number of absolute value statement
--        iterations per process
--    2 : number of processes/number of array declarations
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--        begin
--          pr1 : process
--          begin
--            for i in 1 to 3 loop
--              sig1(i) <= ABS (-i);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--          begin
--            for i in 1 to 3 loop
--              sig2(i) <= ABS (-i);
```

```
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
#2[  signal sig@ : sig_array;]
begin
#2[  pr@ : process
  begin
    for i in 1 to %1% loop
      sig@(i) <= ABS (-i);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  277

PATHNAME :  [.BENCH.B.C.K.L1.P14]shell3.sh
            (UNIX equivalent : bench/b/c/k/l1/p14/shell3.sh)

PURPOSE :  Determine the simulation CPU time required to execute signal
           absolute value assignment statements.  The model simulated is an
           architecture consisting of a number of floating-point array signal
           declarations and the same number of processes.  Each process
           consists of a for-loop.  The number of iterations of the loop is
           equal to the size of the array.  The for-loop contains a signal
           absolute value assignment statement.  The factors to be varied are
           the number of processes/number of array declarations and the array
           size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point signal array size/number of absolute value statement
--        iterations per process
--    2 : number of processes/number of array declarations
--
--  EXAMPLE :
--    $ sim gen/param="shell3.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell3.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of real;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--        begin
--          pr1 : process
--            constant flt_const : real := - 3.3;
--          begin
--            for i in 1 to 3 loop
--              sig1(i) <= ABS (flt_const);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            constant flt_const : real  = - 3.3;
--          begin
```

```
--            for i in 1 to 3 loop
--              sig2(i) <= ABS (flt_const);
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of real;
#2[  signal sig@ : sig_array;]
begin
#2[  pr@ : process
    constant flt_const : real := - %1%.%1%;
  begin
    for i in 1 to %1% loop
      sig@(i) <= ABS (flt_const);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  278

PATHNAME :  [.BENCH.B.C.K.L3.P14]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           and absolute value variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of one integer variable declaration (loop-counter), one
           integer array variable declaration, and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and an absolute value variable assignment
           statement.  The factors to be varied are the array size/number of
           loop iterations and the number of processes.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer array size/number of absolute value statement iterations in
--        process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := ABS (- loop_counter);
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of integer;
```

```
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := ABS (- loop_counter);
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := ABS (- loop_counter);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  279

PATHNAME :  [.BENCH.B.C.K.L3.P14]shell1.sh
            (UNIX equivalent : bench/b/c/k/l3/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           and absolute value variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of one integer variable declaration (loop-counter), one
           floating-point array variable declaration, one floating-point
           constant, and a while-loop.  The number of iterations of the loop is
           equal to the size of the array.  The while-loop contains an addition
           variable assignment statement to increment the loop-counter and an
           absolute value variable assignment statement.  The factors to be
           varied are the array size/number of loop iterations and the number
           of processes.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point array size/number of absolute value statement
--        iterations per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            constant flt_const : real := - 3.3;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := ABS (flt_const);
--            end loop;
--            wait;
--          end process pr1;
```

```
--          pr2 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            constant flt_const : real := - 3.3;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := ABS (flt_const);
--            end loop;
--            wait;
--          end process pr2;
--        end test;

entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of real;
    variable var : var_array;
    constant flt_const : real := - %1%.%1%;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := ABS (flt_const);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  280

PATHNAME :  [.BENCH.B.C.K.L3.P14]shell2.sh
            (UNIX equivalent : bench/b/c/k/l3/p14/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           variable assignment statements and absolute value signal assignment
           statements.  The model simulated is an architecture consisting of a
           number of processes and the same number of integer array signal
           declarations.  Each process consists of one integer variable
           declaration (loop-counter) and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and an absolute value signal assignment
           statement.  The factors to be varied are the array size/number of
           loop iterations and the number of processes/number of array
           declarations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer signal array size/number of absolute value statement
--        iterations per process
--    2 : number of processes/number of array declarations
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--        begin
--          pr1 : process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig1(loop_counter) <= ABS (-loop_counter);
--            end loop;
--            wait;
```

```
--          end process pr1;
--          pr2 : process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig2(loop_counter) <= ABS (-loop_counter);
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
#2[  signal sig@ : sig_array;]
begin
#2[  pr@ : process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig@(loop_counter) <= ABS (-loop_counter);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  281

PATHNAME :  [.BENCH.B.C.K.L3.P14]shell3.sh
            (UNIX equivalent : bench/b/c/k/l3/p14/shell3.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           variable assignment statements and absolute value signal assignment
           statements.  The model simulated is an architecture consisting of a
           number of processes and the same number of floating-point array
           signal declarations.  Each process consists of one integer variable
           declaration (loop-counter), a floating-point constant, and a
           while-loop.  The number of iterations of the loop is equal to the
           size of the array.  The while-loop contains an addition variable
           assignment statement to increment the loop-counter and an absolute
           value signal assignment statement.  The factors to be varied are the
           array size/number of loop iterations and the number of processes/
           number of array declarations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point signal array size/number of absolute value statement
--        iterations per process
--    2 : number of processes/number of array declarations
--
--  EXAMPLE :
--    $ sim gen/param="shell3.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell3.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of real;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--        begin
--          pr1 : process
--            constant flt_const : real := - 3.3;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig1(loop_counter) <= ABS (flt_const);
--            end loop;
```

```
--            wait;
--          end process pr1;
--          pr2 : process
--            constant flt_const : real := - 3.3;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig2(loop_counter) <= ABS (flt_const);
--            end loop;
--            wait;
--          end process pr2;
--        end test;

entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of real;
#2[  signal sig@ : sig_array;]
begin
#2[  pr@ : process
    constant flt_const : real := - %1%.%1%;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig@(loop_counter) <= ABS (flt_const);
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  282

PATHNAME :  [.BENCH.B.C.P14]shell0.sh
            (UNIX equivalent : bench/b/c/p14/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing absolute
           value operations on variables.  The model simulated is an
           architecture consisting of a number of processes; each process
           consists of a number of integer variable declarations and one
           absolute value variable assignment statement for each variable.  The
           factors to be varied are the number of processes and the number of
           variable declarations/number of absolute value variable assignment
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of variable declarations/absolute value statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : integer := - 1;
--            variable var2 : integer := - 2;
--            variable var3 : integer := - 3;
--          begin
--            var1 := ABS (var1);
--            var2 := ABS (var2);
--            var3 := ABS (var3);
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : integer := - 1;
--            variable var2 : integer := - 2;
--            variable var3 : integer := - 3;
--          begin
--            var1 := ABS (var1);
```

```
--            var2 := ABS (var2);
--            var3 := ABS (var3);
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : integer := - @;]
  begin
#1[    var@ := ABS (var@);]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  283

PATHNAME :  [.BENCH.B.C.P14]shell1.sh
            (UNIX equivalent : bench/b/c/p14/shell1.sh)

PURPOSE :  Determine the simulation CPU time required for executing absolute
           value operations on variables.  The model simulated is an
           architecture consisting of a number of processes; each process
           consists of a number of floating-point variable declarations and one
           absolute value variable assignment statement for each variable.  The
           factors to be varied are the number of processes and the number of
           variable declarations/number of absolute value variable assignment
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements per process
--    2 : number of processes
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : real := - 1.1;
--            variable var2 : real := - 2.2;
--            variable var3 : real := - 3.3;
--          begin
--            var1 := ABS (var1);
--            var2 := ABS (var2);
--            var3 := ABS (var3);
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : real := - 1.1;
--            variable var2 : real := - 2.2;
--            variable var3 : real := - 3.3;
--          begin
```

```
--            var1 := ABS (var1);
--            var2 := ABS (var2);
--            var3 := ABS (var3);
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : real := - @.@;]
  begin
#1[    var@ := ABS (var@);]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  284

PATHNAME :  [.BENCH.B.C.P14]shell2.sh
            (UNIX equivalent : bench/b/c/p14/shell2.sh)

PURPOSE :  Determine the simulation CPU time required for executing absolute
           value operations on signals.  The model simulated is an architecture
           consisting of a process and a number of integer signal declarations;
           the process consists of one absolute value signal assignment
           statement for each signal.  The factor to be varied is the number of
           signal declarations/number of absolute value signal assignment
           statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/number of absolute value statements in
--        the process
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : integer := - 1;
--          signal sig2 : integer := - 2;
--          signal sig3 : integer := - 3;
--        begin
--          process
--          begin
--            sig1 <= ABS (sig1);
--            sig2 <= ABS (sig2);
--            sig3 <= ABS (sig3);
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
#1[  signal sig@ : integer := - @;]
begin
```

```
  process
  begin
#1[    sig@ <= ABS (sig@);]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  285

PATHNAME :  [.BENCH.B.C.P14]shell3.sh
            (UNIX equivalent : bench/b/c/p14/shell3.sh)

PURPOSE :  Determine the simulation CPU time required for executing absolute
           value operations on signals.  The model simulated is an architecture
           consisting of a process and a number of floating-point signal
           declarations; the process consists of one absolute value signal
           assignment statement for each signal.  The factor to be varied is
           the number of signal declarations/number of absolute value signal
           assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating-point signal declarations/number of absolute value
--        statements in the process
--
--  EXAMPLE :
--    $ sim gen/param="shell3.vhd","test.vhd",3
--    (UNIX equivalent : % sim gen -param=\"\shell3.sh\"\,\"\test.vhd\"\,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : real := - 1.1;
--          signal sig2 : real := - 2.2;
--          signal sig3 : real := - 3.3;
--        begin
--          process
--          begin
--            sig1 <= ABS (sig1);
--            sig2 <= ABS (sig2);
--            sig3 <= ABS (sig3);
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
#1[  signal sig@ : real := - @.@;]
begin
```

```
  process
  begin
#1[    sig@ <= ABS (sig@);]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  286

PATHNAME :  [.BENCH.A.C.H1.P15]shell0.sh
            (UNIX equivalent : bench/a/c/h1/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call.  The procedure consists of a number of integer
           variable declarations and an exponentiation statement for each
           variable.  The factors to be varied are the number of variable
           declarations/exponentiation statements in the procedure and the
           value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/exponentiation statements in
--        procedure
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure expo is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--          end expo;
--        end test;
--        architecture test of test is
--        begin
--          expo;
--        end test;


entity test is
```

```
  procedure expo is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ ** %2%;]
  end expo;
end test;
architecture test of test is
begin
  expo;
end test;
```

```
TEST NUMBER :  287

PATHNAME :  [.BENCH.A.C.H1.P15]shell1.sh
            (UNIX equivalent : bench/a/c/h1/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an entity
           consisting of a procedure declaration and an architecture consisting
           of a procedure call.  The procedure consists of a number of
           floating-point variable declaration and an exponentiation statement
           for each variable.  The factors to be varied are the number of
           variable declarations/exponentiation statements in the procedure and
           the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/exponentiation
--        statements in procedure
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          procedure expo is
--            variable var1 : real := 1.1;
--            variable var2 : real := 2.2;
--            variable var3 : real := 3.3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--          end expo;
--        end test;
--        architecture test of test is
--        begin
--          expo;
--        end test;


entity test is
```

```
  procedure expo is
#1[    variable var@ : real := @.@;]
  begin
#1[    var@ := var@ ** %2%;]
  end expo;
end test;
architecture test of test is
begin
  expo;
end test;
```

```
TEST NUMBER :  288

PATHNAME :  [.BENCH.A.C.H2.P15]shell0.sh
            (UNIX equivalent : bench/a/c/h2/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of integer variable declarations
           and an exponentiation statement for each variable.  The factors to
           be varied are the number of variable declarations/exponentiation
           statements in the procedure and the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/exponentiation statements in
--        procedure
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="s ell0.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure expo is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--          end expo;
--        begin
--          expo;
--        end test;


entity test is end;
architecture test of test is
  procedure expo is
```

```
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ ** %2%;]
  end expo;
begin
  expo;
end test;
```

```
TEST NUMBER :  289

PATHNAME :  [.BENCH.A.C.H2.P15]shell1.sh
            (UNIX equivalent : bench/a/c/h2/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an architecture
           consisting of a procedure declaration and a procedure call.  The
           procedure consists of a number of floating-point variable
           declarations and an exponentiation statement for each variable.  The
           factors to be varied are the number of variable declarations/
           exponentiation statements in the procedure and the value of the
           exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/exponentiation
--        statements in procedure
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure expo is
--            variable var1 : real := 1.1;
--            variable var2 : real := 2.2;
--            variable var3 : real := 3.3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--          end expo;
--        begin
--          expo;
--        end test;


entity test is end;
architecture test of test is
```

```
  procedure expo is
#1[    variable var@ : real := @.@;]
  begin
#1[    var@ := var@ ** %2%;]
  end expo;
begin
  expo;
end test;
```

```
TEST NUMBER :  290

PATHNAME :  [.BENCH.A.C.I1.P15]shell0.sh
            (UNIX equivalent : bench/a/c/i1/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of integer
           variable declarations and an exponentiation statement for each
           variable.  The factors to be varied are the number of variable
           declarations/exponentiation statements in the function and the value
           of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/absolute value statements in
--        function
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function expo return boolean is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--            return true;
--          end expo;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= expo;
--        end test;
```

```
entity test is
  function expo return boolean is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ ** %2%;]
    return true;
  end expo;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= expo;
end test;
```

```
TEST NUMBER :  291

PATHNAME :  [.BENCH.A.C.I1.P15]shell1.sh
            (UNIX equivalent : bench/a/c/i1/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an entity
           consisting of a function declaration and an architecture consisting
           of a function call.  The function consists of a number of
           floating-point variable declarations and an exponentiation statement
           for each variable.  The factors to be varied are the number of
           variable declarations/exponentiation statements in the function and
           the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements in function
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is
--          function expo return boolean is
--            variable var1 : real := 1.1;
--            variable var2 : real := 2.2;
--            variable var3 : real := 3.3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--            return true;
--          end expo;
--        end test;
--        architecture test of test is
--          signal done : boolean := false;
--        begin
--          done <= expo;
--        end test;
```

```
entity test is
  function expo return boolean is
#1[    variable var@ : real := @.@;]
  begin
#1[    var@ := var@ ** %2%;]
    return true;
  end expo;
end test;
architecture test of test is
  signal done : boolean := false;
begin
  done <= expo;
end test;
```

```
TEST NUMBER :  292

PATHNAME :  [.BENCH.A.C.I2.P15]shell0.sh
            (UNIX equivalent : bench/a/c/i2/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of integer variable declarations
           and an exponentiation statement for each variable.  The factors to
           be varied are the number of variable declarations/exponentiation
           statements in the function and the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/absolute value statements in
--        function
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function expo return boolean is
--            variable var1 : integer := 1;
--            variable var2 : integer := 2;
--            variable var3 : integer := 3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--            return true;
--          end expo;
--          signal done : boolean := false;
--        begin
--          done <= expo;
--        end test;


entity test is end;
```

```
architecture test of test is
  function expo return boolean is
#1[    variable var@ : integer := @;]
  begin
#1[    var@ := var@ ** %2%;]
    return true;
  end expo;
  signal done : boolean := false;
begin
  done <= expo;
end test;
```

```
TEST NUMBER :  293

PATHNAME :  [.BENCH.A.C.I2.P15]shell1.sh
            (UNIX equivalent : bench/a/c/i2/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation statements.  The model simulated is an architecture
           consisting of a function declaration and a function call.  The
           function consists of a number of floating-point variable declarations
           and an exponentiation statement for each variable.  The factors to
           be varied are the number of variable declarations/exponentiation
           statements in the function and the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 25 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/absolute value
--        statements in function
--    2 : exponent (must be INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function expo return boolean is
--            variable var1 : real := 1.1;
--            variable var2 : real := 2.2;
--            variable var3 : real := 3.3;
--          begin
--            var1 := var1 ** 5;
--            var2 := var2 ** 5;
--            var3 := var3 ** 5;
--            return true;
--          end expo;
--          signal done : boolean := false;
--        begin
--          done <= expo;
--        end test;


entity test is end;
```

```
architecture test of test is
  function expo return boolean is
#1[    variable var@ : real := @.@;]
  begin
#1[    var@ := var@ ** %2%;]
    return true;
  end expo;
  signal done : boolean := false;
begin
  done <= expo;
end test;
```

```
TEST NUMBER :  294

PATHNAME :  [.BENCH.A.C.P15]shell0.sh
            (UNIX equivalent : bench/a/c/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute exponentiation
           operations on signals.  The model simulated consists of a number of
           integer signal declarations and one exponentiation signal assignment
           statement for each signal.  The factors to be varied are the number
           of signal declarations/number of exponentiation signal assignment
           statements and the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer signal declarations/exponentiation statements
--    2 : length of time (ns) to simulate model (> 0)
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,4,5
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,4,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal test_sig : integer := 0;
--          signal sig1 : integer := 1;
--          signal sig2 : integer := 1;
--          signal sig3 : integer := 1;
--        begin
--          sig1 <= test_sig ** 5 after 1 ns;
--          sig2 <= test_sig ** 5 after 1 ns;
--          sig3 <= test_sig ** 5 after 1 ns;
--          test_sig <= 0
--                     ,1 after 1 ns
--                     ,0 after 2 ns
--                     ,1 after 3 ns
--                     ,0 after 4 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal stop : bit := '0';
  signal test_sig : integer := 0;
#1[  signal sig@ : integer := 1;]
begin
#1[  sig@ <= test_sig ** %3% after 1 ns;]
  test_sig <= 0
#2[              ,$2$1$0$ after @ ns];
  stop <= '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  295

PATHNAME :  [.BENCH.A.C.P15]shell1.sh
            (UNIX equivalent : bench/a/c/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute exponentiation
           operations on signals.  The model simulated consists of a number of
           floating-point signal declarations and one exponentiation signal
           assignment statement for each signal   The factors to be varied are
           the number of signal declarations/number of exponentiation signal
           assignment statements and the value of the exponent.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point signal declarations/exponentiation statements
--    2 : length of time (ns) to simulate model (> 0)
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,4,5
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,4,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : bit := '0';
--          signal test_sig : real := 0.0;
--          signal sig1 : real := 1.0;
--          signal sig2 : real := 1.0;
--          signal sig3 : real := 1.0;
--        begin
--          sig1 <= test_sig ** 5 after 1 ns;
--          sig2 <= test_sig ** 5 after 1 ns;
--          sig3 <= test_sig ** 5 after 1 ns;
--          test_sig <= 0.0
--                     ,1.0 after 1 ns
--                     ,0.0 after 2 ns
--                     ,1.0 after 3 ns
--                     ,0.0 after 4 ns;
--          stop <= '1' after 4 ns;
--          assert (stop = '0') report "simulation complete" severity failure;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal stop : bit := '0';
  signal test_sig : real := 0.0;
#1[  signal sig@ : real := 1.0;]
begin
#1[  sig@ <= test_sig ** %3% after 1 ns;]
  test_sig <= 0.0
#2[              ,$2$1.0$0.0$ after @ ns];
  stop <= '1' after %2% ns;
  assert (stop = '0') report "simulation complete" severity failure;
end test;
```

```
TEST NUMBER :  296

PATHNAME :  [.BENCH.B.C.K.L1.P15]shell0.sh
            (UNIX equivalent : bench/b/c/k/l1/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an integer array variable declaration and a for-loop.
           The number of iterations of the loop is equal to the size of the
           array.  The for-loop contains a variable exponentiation assignment
           statement.  The factors to be varied are the number of processes,
           the value of the exponent, and the array size/number of loop
           iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer array size/number of exponentiation statement iterations per
--        process
--    2 : number of processes
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--            variable test_var : integer := 1;
--          begin
--            for i in 1 to 3 loop
--              var(i) := test_var ** 4;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
```

```
--            variable test_var : integer := 1;
--          begin
--            for i in 1 to 3 loop
--              var(i) := test_var ** 4;
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
    variable test_var : integer := 1;
  begin
    for i in 1 to %1% loop
      var(i) := test_var ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  297

PATHNAME :  [.BENCH.B.C.K.L1.P15]shell1.sh
            (UNIX equivalent : bench/b/c/k/l1/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to execute variable
           exponentiation assignment statements.  The model simulated is an
           architecture consisting of a number of processes.  Each process
           consists of an floating-point array variable declaration and a
           for-loop.  The number of iterations of the loop is equal to the size
           of the array.  The for-loop contains a variable exponentiation
           assignment statement.  The factors to be varied are the number of
           processes, the value of the exponent, and the array size/number of
           loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point array size/number of exponentiation statement
--        iterations per process
--    2 : number of processes
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            variable test_var : real := 1.0;
--          begin
--            for i in 1 to 3 loop
--              var(i) := test_var ** 4;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
```

```
--            variable test_var : real := 1.0;
--          begin
--            for i in 1 to 3 loop
--              var(i) := test_var ** 4;
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of real;
    variable var : var_array;
    variable test_var : real := 1.0;
  begin
    for i in 1 to %1% loop
      var(i) := test_var ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  298

PATHNAME :  [.BENCH.B.C.K.L1.P15]shell2.sh
            (UNIX equivalent : bench/b/c/k/l1/p15/shell2.sh)

PURPOSE :  Determine the simulation CPU time required to execute signal
           exponentiation assignment statements.  The model simulated is an
           architecture consisting of a number of integer array signal
           declarations and the same number of processes.  Each process
           consists of a for-loop.  The number of iterations of the loop is
           equal to the size of the array.  The for-loop contains a signal
           exponentiation assignment statement.  The factors to be varied are
           the number of processes/number of array declarations, the value of
           the exponent, and the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer signal array size/number of exponentiation statement
--        iterations per process
--    2 : number of processes/number of array declarations
--    3 : exponent
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--          signal test_sig : integer := 1;
--        begin
--          pr1 : process
--          begin
--            for i in 1 to 3 loop
--              sig1(i) <= test_sig ** 4;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--          begin
```

```
--            for i in 1 to 3 loop
--              sig2(i) <= test_sig ** 4;
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
#2[  signal sig@ : sig_array;]
  signal test_sig : integer := 1;
begin
#2[  pr@ : process
  begin
    for i in 1 to %1% loop
      sig@(i) <= test_sig ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  299

PATHNAME :  [.BENCH.B.C.K.L1.P15]shell3.sh
            (UNIX equivalent : bench/b/c/k/l1/p15/shell3.sh)

PURPOSE :  Determine the simulation CPU time required to execute signal
           exponentiation assignment statements.  The model simulated is an
           architecture consisting of a number of floating-point array signal
           declarations and the same number of processes.  Each process
           consists of a for-loop.  The number of iterations of the loop is
           equal to the size of the array.  The for-loop contains a signal
           exponentiation assignment statement.  The factors to be varied are
           the number of processes/number of array declarations, the value of
           the exponent, and the array size/number of loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer signal array size/number of exponentiation statement
--        iterations per process
--    2 : number of processes/number of array declarations
--    3 : exponent
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--          signal test_sig : integer := 1;
--        begin
--          pr1 : process
--          begin
--            for i in 1 to 3 loop
--              sig1(i) <= test_sig ** 4;
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--          begin
```

```
--            for i in 1 to 3 loop
--              sig2(i) <= test_sig ** 4;
--            end loop;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
#2[  signal sig@ : sig_array;]
  signal test_sig : integer := 1;
begin
#2[  pr@ : process
  begin
    for i in 1 to %1% loop
      sig@(i) <= test_sig ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  300

PATHNAME :  [.BENCH.B.C.K.L3.P15]shell0.sh
            (UNIX equivalent : bench/b/c/k/l3/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           and exponentiation variable assignment statements.  The model
           simulated is an architecture consisting of a process.  The process
           consists of one integer variable declaration (loop-counter), one
           integer array variable declaration, and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and an exponentiation variable assignment
           statement.  The factors to be varied are the array size/number of
           loop iterations, the value of the exponent, and the number of
           processes.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer array size/number of exponentiation statement iterations in
--        process
--    2 : number of processes
--    3 : exponent
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--            variable test_var : integer := 1;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := test_var ** 4;
--            end loop;
--            wait;
```

```
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of integer;
--            variable var : var_array;
--            variable loop_counter : integer := 0;
--            variable test_var : integer := 1;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := test_var ** 4;
--            end loop;
--            wait;
--          end process pr2;
--      .   end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of integer;
    variable var : var_array;
    variable loop_counter : integer := 0;
    variable test_var : integer := 1;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := test_var ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

TEST NUMBER : 301

PATHNAME : [.BENCH.B.C.K.L3.P15]shell1.sh
(UNIX equivalent : bench/b/c/k/l3/p15/shell1.sh)

PURPOSE : Determine the simulation CPU time required to execute addition and exponentiation variable assignment statements. The model simulated is an architecture consisting of a process. The process consists of one integer variable declaration (loop-counter), one floating-point array variable declaration, one floating-point constant, and a while-loop. The number of iterations of the loop is equal to the size of the array. The while-loop contains an addition variable assignment statement to increment the loop-counter and an exponentiation variable assignment statement. The factors to be varied are the array size/number of loop iterations, the value of the exponent, and the number of processes.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point array size/number of exponentiation statement
--        iterations per process
--    2 : number of processes
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            variable test_var : real := 1.0;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := test_var ** 4;
--            end loop;
--            wait;
```

```
--          end process pr1;
--          pr2 : process
--            type var_array is array(1 to 3) of real;
--            variable var : var_array;
--            variable test_var : real := 1.0;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              var(loop_counter) := test_var ** 4;
--            end loop;
--            wait;
--          end process pr2;
--      end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
    type var_array is array(1 to %1%) of real;
    variable var : var_array;
    variable test_var : real := 1.0;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      var(loop_counter) := test_var ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

TEST NUMBER : 302

PATHNAME : [.BENCH.B.C.K.L3.P15]shell2.sh
(UNIX equivalent : bench/b/c/k/l3/p15/shell2.sh)

PURPOSE : Determine the simulation CPU time required to execute addition variable assignment statements and exponentiation signal assignment statements. The model simulated is an architecture consisting of a number of process and the same number of integer array signal declarations. Each process consists of one integer variable declaration (loop-counter) and a while-loop. The number of iterations of the loop is equal to the size of the array. The while-loop contains an addition variable assignment statement to increment the loop-counter and an exponentiation signal assignment statement. The factors to be varied are the array size/number of loop iterations, the value of the exponent, and the number of processes/number of array declarations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : integer signal array size/number of exponentiation statement
--        iterations per process
--    2 : number of processes/number of array declarations
--    3 : exponent
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of integer;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--          signal test_sig : integer := 1;
--        begin
--          pr1 : process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter< 3 loop
--              loop_counter := loop_counter + 1;
--              sig1(loop_counter) <= test_sig ** 4;
```

```
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter< 3 loop
--              loop_counter := loop_counter + 1;
--              sig2(loop_counter) <= test_sig ** 4;
--            end loop;
--            wait;
--          end process pr2;
--        end test;

      .

entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of integer;
#2[  signal sig@ : sig_array;]
  signal test_sig : integer := 1;
begin
#2[  pr@ : process
    variable loop_counter : integer := 0;
  begin
    while loop_counter< %1% loop
      loop_counter := loop_counter + 1;
      sig@(loop_counter) <= test_sig ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  303

PATHNAME :  [.BENCH.B.C.K.L3.P15]shell3.sh
            (UNIX equivalent : bench/b/c/k/l3/p15/shell3.sh)

PURPOSE :  Determine the simulation CPU time required to execute addition
           variable assignment statements and exponentiation signal assignment
           statements.  The model simulated is an architecture consisting of a
           number of processes and the same number of floating-point array
           signal declarations.  Each process consists of one integer variable
           declaration (loop-counter) and a while-loop.  The number of
           iterations of the loop is equal to the size of the array.  The
           while-loop contains an addition variable assignment statement to
           increment the loop-counter and an exponentiation signal assignment
           statement.  The factors to be varied are the array size/number of
           loop iterations, the value of the exponent, and the number of
           processes/number of array declarations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : floating point signal array size/number of exponentiation statement
--        iterations per process
--    2 : number of processes/number of array declarations
--    3 : exponent
--
--  EXAMPLE :
--    $ sim gen/param="shell3.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell3.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type sig_array is array(1 to 3) of real;
--          signal sig1 : sig_array;
--          signal sig2 : sig_array;
--          signal test_sig : real := 1.0;
--        begin
--          pr1 : process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig1(loop_counter) <= test_sig ** 4;
```

```
--            end loop;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 3 loop
--              loop_counter := loop_counter + 1;
--              sig2(loop_counter) <= test_sig ** 4;
--            end loop;
--            wait;
--          end process pr2;
--        end test;

entity test is end;
architecture test of test is
  type sig_array is array(1 to %1%) of real;
#2[  signal sig@ : sig_array;]
  signal test_sig : real := 1.0;
begin
#2[  pr@ : process
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      sig@(loop_counter) <= test_sig ** %3%;
    end loop;
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  304

PATHNAME :  [.BENCH.B.C.P15]shell0.sh
            (UNIX equivalent : bench/b/c/p15/shell0.sh)

PURPOSE :  Determine the simulation CPU time required for executing
           exponentiation operations on variables.  The model simulated is an
           architecture consisting of a number of processes; each process
           consists of a number of integer variable declarations and one
           exponentiation variable assignment statement for each variable.  The
           factors to be varied are the number of processes, the value of the
           exponent, and the number of variable declarations/number of
           exponentiation variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer variable declarations/exponentiation statements per
--        process
--    2 : number of processes
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : integer := 0;
--            variable var2 : integer := 0;
--            variable var3 : integer := 0;
--            variable test_var : integer := 1;
--          begin
--            var1 := test_var ** 4;
--            var2 := test_var ** 4;
--            var3 := test_var ** 4;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : integer := 0;
--            variable var2 : integer := 0;
```

```
--            variable var3 : integer := 0;
--            variable test_var : integer := 1;
--          begin
--            var1 := test_var ** 4;
--            var2 := test_var ** 4;
--            var3 := test_var ** 4;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin .
#2[  pr@ : process
#1[    variable var@ : integer := 0;]
    variable test_var : integer := 1;
  begin
#1[    var@ := test_var ** %3%;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  305

PATHNAME :  [.BENCH.B.C.P15]shell1.sh
            (UNIX equivalent : bench/b/c/p15/shell1.sh)

PURPOSE :  Determine the simulation CPU time required for executing
           exponentiation operations on variables.  The model simulated is an
           architecture consisting of a number of processes; each process
           consists of a number of floating-point variable declarations and one
           exponentiation variable assignment statement for each variable.  The
           factors to be varied are the number of processes, the value of the
           exponent, and the number of variable declarations/number of
           exponentiation variable assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point variable declarations/exponentiation
--        statements per process
--    2 : number of processes
--    3 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.vhd","test.vhd",3,2,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          pr1 : process
--            variable var1 : real := 0.0;
--            variable var2 : real := 0.0;
--            variable var3 : real := 0.0;
--            variable test_var : real := 1.0;
--          begin
--            var1 := test_var ** 4;
--            var2 := test_var ** 4;
--            var3 := test_var ** 4;
--            wait;
--          end process pr1;
--          pr2 : process
--            variable var1 : real := 0.0;
--            variable var2 : real := 0.0;
```

```
--            variable var3 : real := 0.0;
--            variable test_var : real := 1.0;
--          begin
--            var1 := test_var ** 4;
--            var2 := test_var ** 4;
--            var3 := test_var ** 4;
--            wait;
--          end process pr2;
--        end test;


entity test is end;
architecture test of test is
begin
#2[  pr@ : process
#1[    variable var@ : real := 0.0;]
    variable test_var : real := 1.0;
  begin
#1[    var@ := test_var ** %3%;]
    wait;
  end process pr@;]
end test;
```

```
TEST NUMBER :  306

PATHNAME :  [.BENCH.B.C.P15]shell2.sh
            (UNIX equivalent : bench/b/c/p15/shell2.sh)

PURPOSE :  Determine the simulation CPU time required for executing
           exponentiation operations on signals.  The model simulated is an
           architecture consisting of a process and a number of integer signal
           declarations; the process consists of one exponentiation signal
           assignment statement for each signal.  The factors to be varied are
           the value of the exponent and the number of signal declarations/
           number of exponentiation signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of integer signal declarations/number of exponentiation
--        statements in the process
--    2 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell2.vhd","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell2.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : integer := 0;
--          signal sig2 : integer := 0;
--          signal sig3 : integer := 0;
--          signal test_sig : integer := 1;
--        begin
--          process
--          begin
--            sig1 <= test_sig ** 4;
--            sig2 <= test_sig ** 4;
--            sig3 <= test_sig ** 4;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
```

```
#1[  signal sig@ : integer := 0;]
  signal test_sig : integer := 1;
begin
  process
  begin
#1[    sig@ <= test_sig ** %2%;]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  307

PATHNAME :  [.BENCH.B.C.P15]shell3.sh
            (UNIX equivalent : bench/b/c/p15/shell3.sh)

PURPOSE :  Determine the simulation CPU time required for executing
           exponentiation operations on signals.  The model simulated is an
           architecture consisting of a process and a number of floating-point
           signal declarations; the process consists of one exponentiation
           signal assignment statement for each signal.  The factors to be
           varied are the value of the exponent and the number of signal
           declarations/number of exponentiation signal assignment statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 26 July 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of floating point signal declarations/number of exponentiation
--        statements in the process
--    2 : exponent (INTEGER)
--
--  EXAMPLE :
--    $ sim gen/param="shell3.vhd","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell3.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : real := 0.0;
--          signal sig2 : real := 0.0;
--          signal sig3 : real := 0.0;
--          signal test_sig : real := 1.0;
--        begin
--          process
--          begin
--            sig1 <= test_sig ** 4;
--            sig2 <= test_sig ** 4;
--            sig3 <= test_sig ** 4;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
```

```
#1[  signal sig@ : real := 0.0;]
  signal test_sig : real := 1.0;
begin
  process
  begin
#1[    sig@ <= test_sig ** %2%;]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  308

PATHNAME :  [.BENCH.B.C.L1.S1]shell1.sh
            (UNIX equivalent : bench/b/c/l1/s1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform variable
           character read operations from a file.  The model simulated is
           an architecture consisting of a process.  The process contains
           a variable declaration and a for-loop, whose number of iterations
           is equal to the number of characters to read.  The for-loop
           contains a read statement that reads one character from the input
           file.  The factor to be varied is the number of characters to read.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 18 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size (number of characters) of file; the value of this parameter
--        must be less than or equal to the value of the parameter in
--        "shell0.sh"
--
--
--  *** NOTE : The shell in "shell0.sh" must also be expanded (via the "sim gen"
--  ***        command; see comments in "shell0.sh") when using this benchmark.
--
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",15
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,15)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type ifile_type is file of character;
--          file ifile : ifile_type is in "data_file.dat";
--        begin
--          process
--            variable ch : character;
--          begin
--            for i in 1 to 10 loop
--              read(ifile,ch);
--            end loop;
--            wait;
--          end process;
--        end test;
```

```
entity test is end;
architecture test of test is
  type ifile_type is file of character;
  file ifile : ifile_type is in "data_file.dat";
begin
  process
    variable ch : character;
  begin
    for i in 1 to %1% loop
      read(ifile,ch);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  309

PATHNAME :  [.BENCH.B.C.L2.S1]shell1.sh
            (UNIX equivalent : bench/b/c/l2/s1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform variable
           character read operations from a file.  The model simulated is
           an architecture consisting of a process.  The process contains
           a variable declaration and a for-loop, whose iteration scheme is
           integer'high.  The for-loop contains a read statement that reads
           one character from the input file and an exit statement that limits
           the number of iterations to the number of characters desired to be
           read.  The factor to be varied is the number of characters to read.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 21 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size (number of characters) of file; the value of this parameter
--        must be less than or equal to the value of the parameter in
--        "shell0.sh"
--
--
--  *** NOTE : The shell in "shell0.sh" must also be expanded (via the "sim gen"
--  ***        command; see comments in "shell0.sh") when using this benchmark.
--
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",15
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,15)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type ifile_type is file of character;
--          file ifile : ifile_type is in "data_file.dat";
--        begin
--          process
--            variable ch : character;
--          begin
--            for i in 1 to integer'high loop
--              read(ifile,ch);
--              exit when i = 15;
--            end loop;
--            wait;
```

```
--          end process;
--        end test;


entity test is end;
architecture test of test is
  type ifile_type is file of character;
  file ifile : ifile_type is in "data_file.dat";
begin
  process
    variable ch : character;
  begin
    for i in 1 to integer'high loop
      read(ifile,ch);
      exit when i = %1%;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  310

PATHNAME :  [.BENCH.B.C.L3.S1]shell1.sh
            (UNIX equivalent : bench/b/c/l3/s1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform variable
           character read operations from a file.  The model simulated is
           an architecture consisting of a process.  The process contains
           two variable declarations (one is a loop-counter) and a while-loop,
           whose number of iterations is equal to the number of characters to
           read.  The while-loop contains a variable addition statement to
           increment the loop-counter and a read statement that reads one
           character from the input file.  The factor to be varied is the
           number of characters to read.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size (number of characters) of file; the value of this parameter
--        must be less than or equal to the value of the parameter in
--        "shell0.sh"
--
--
--  *** NOTE : The shell in "shell0.sh" must also be expanded (via the "sim gen"
--  ***        command; see comments in "shell0.sh") when using this benchmark.
--
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",15
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,15)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type ifile_type is file of character;
--          file ifile : ifile_type is in "data_file.dat";
--        begin
--          process
--            variable ch : character;
--            variable loop_counter : integer := 0;
--          begin
--            while loop_counter < 15 loop
--              loop_counter := loop_counter + 1;
--              read(ifile,ch);
```

```
--            end loop;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
  type ifile_type is file of character;
  file ifile : ifile_type is in "data_file.dat";
begin
  process
    variable ch : character;
    variable loop_counter : integer := 0;
  begin
    while loop_counter < %1% loop
      loop_counter := loop_counter + 1;
      read(ifile,ch);
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  311

PATHNAME :  [.BENCH.B.C.L4.S1]shell1.sh
            (UNIX equivalent : bench/b/c/l4/s1/shell1.sh)

PURPOSE :  Determine the simulation CPU time required to perform variable
           character read operations from a file.  The model simulated is
           an architecture consisting of a process.  The process contains
           two variable declarations (one is a loop-counter) and a while-loop,
           whose iteration scheme is true.  The while-loop contains a variable
           addition statement to increment the loop-counter, a read statement
           that reads one character from the input file and an exit statement
           that limits the number of iterations to the number of characters
           desired to be read.  The factor to be varied is the number of
           characters to read.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 22 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : size (number of characters) of file; the value of this parameter
--        must be less than or equal to the value of the parameter in
--        "shell0.sh"
--
--
--  *** NOTE : The shell in "shell0.sh" must also be expanded (via the "sim gen"
--  ***        command; see comments in "shell0.sh") when using this benchmark.
--
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",15
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,15)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          type ifile_type is file of character;
--          file ifile : ifile_type is in "data_file.dat";
--        begin
--          process
--            variable ch : character;
--            variable loop_counter : integer := 0;
--          begin
--            while true loop
--              loop_counter := loop_counter + 1;
```

```
--              read(ifile,ch);
--              exit when loop_counter = 15;
--            end loop;
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
  type ifile_type is file of character;
  file ifile : ifile_type is in "data_file.dat";
begin
  process
    variable ch : character;
    variable loop_counter : integer := 0;
  begin
    while true loop
      loop_counter := loop_counter + 1;
      read(ifile,ch);
      exit when loop_counter = %1%;
    end loop;
    wait;
  end process;
end test;
```

```
TEST NUMBER :  312

PATHNAME :  [.BENCH.A.C.D1A]shell1.sh (see readme.txt in this directory)
            (UNIX equivalent : bench/a/c/d1a/shell1.sh)

PURPOSE :  Determine the effect of guarded blocks and resolved signals on CPU
           time.  The model simulated is a package declaration, a package body
           declaration, and an architecture.  The package declaration has two
           type declarations and a resolution function declaration.  The package
           body has the body of the resolution function.  The architecture
           consists of a resolved signal declaration, a number of non-resolved
           signal declarations (guards), a guarded block statement for each
           guard signal, and a signal assignment statement for each guard
           signal.  Each block contains a signal assignment statement for the
           resolved signal.  The factors to be varied are the number of
           non-resolved signal declarations/number of blocks/number of signal
           assignment statements, and the length of time (in ns) to simulate
           the model.  This description is part of an example from Dr Jim
           Armstrong's book.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of guarded blocks/number of guard signals
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        package MVL_package is
--                  .
--                  .
--                  .
--        package body MVL_package is
--                  .
--                  .
--                  .
--          end MVL_package;
--
--
--        use work.MVL_package.all;
--        entity test is end test;
```

```
--        architecture test of test is
--            signal X : MVL_res_func MVL;
--            signal PHASE_1 : boolean;
--            signal PHASE_2 : boolean;
--            signal PHASE_3 : boolean;
--            begin
--            b1 : block(PHASE_1)
--              begin
--              X <= guarded '1' after 500 ps;
--              end block b1;
--            b2 : block(PHASE_2)
--              begin
--              X <= guarded '0' after 500 ps;
--              end block b2;
--            b3 : block(PHASE_3)
--              begin
--              X <= guarded '1' after 500 ps;
--              end block b3;
--            PHASE_1 <= true after 750 ps
--                      ,false after 1 ns
--                      ,true after 2 ns
--                      ,false after 3 ns
--                      ,true after 4 ns;
--            PHASE_2 <= true after 750 ps
--                      ,false after 1 ns
--                      ,true after 2 ns
--                      ,false after 3 ns
--                      ,true after 4 ns;
--            PHASE_3 <= true after 750 ps
--                      ,false after 1 ns
--                      ,true after 2 ns
--                      ,false after 3 ns
--                      ,true after 4 ns;
--            end test;
--    This description is part of an example from Dr Jim Armstrong's book.


package MVL_package is
    type MVL is ('Z','0','1');
    type MVL_res_vec is array (integer range <>) of MVL;
    function MVL_res_func (S : MVL_res_vec) return MVL;
    end MVL_package;
package body MVL_package is
  function MVL_res_func (S : MVL_res_vec) return MVL is
    variable RESOLVED_VALUE : MVL := 'Z';
    begin
      for I in S'RANGE loop
        if S(I) /= 'Z' then
          RESOLVED_VALUE := S(I);
          exit;
        end if;
```

```
      end loop;
      return RESOLVED_VALUE;
    end MVL_res_func;
  end MVL_package;


use work.MVL_package.all;
entity test is end test;
architecture test of test is
    signal X : MVL_res_func MVL;
#1[    signal PHASE_@ : boolean;]
    begin
#1[    b@ : block(PHASE_@)
      begin
      X <= guarded '$2$1$0$' after 500 ps;
      end block b@;]
#1[    PHASE_@ <= true after 750 ps
#2[              ,$2$false$true$ after @ ns];]
    end test;
```

TEST NUMBER : 313

PATHNAME : [.BENCH.A.C.D1C]shell1.sh (see readme.txt in this directory)
(UNIX equivalent : bench/a/c/d1c/shell1.sh)

PURPOSE : Determine the effect of guarded blocks and bus resolved signals on CPU time. The model simulated is a package declaration, a package body declaration, and an architecture. The package declaration has two type declarations and a resolution function declaration. The package body has the body of the resolution function. The architecture consists of a bus resolved signal declaration, a number of non-resolved signal declarations (guards), a guarded block statement for each guard signal, and a signal assignment statement for each guard signal. Each block contains a signal assignment statement for the bus resolved signal. The factors to be varied are the number of non-resolved signal declarations/number of blocks/ number of signal assignment statements, and the length of time (in ns) to simulate the model. This description is part of an example from Dr Jim Armstrong's book.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of guarded blocks/number of guard signals
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        package MVL_package is
--                .
--        package body MVL_package is
--                .
--          end MVL_package;
--        use work.MVL_package.all;
--        entity test is end test;
--        architecture test of test is
--            signal X : MVL_res_func MVL bus;
--            signal PHASE_1 : boolean;
--            signal PHASE_2 : boolean;
--            signal PHASE_3 : boolean;
--            begin
```

```
--            b1 : block(PHASE_1)
--               begin
--               X <= guarded '1' after 500 ps;
--               end block b1;
--            b2 : block(PHASE_2)
--               begin
--               X <= guarded '0' after 500 ps;
--               end block b2;
--            b3 : block(PHASE_3)
--               begin
--               X <= guarded '1' after 500 ps;
--               end block b3;
--            PHASE_1 <= true after 750 ps
--                     ,false after 1 ns
--                     ,true after 2 ns;
--            PHASE_2 <= true after 750 ps
--                     ,false after 1 ns
--                     ,true after 2 ns;
--            PHASE_3 <= true after 750 ps
--                     ,false after 1 ns
--                     ,true after 2 ns;
--            end test;
--    This description is part of an example from Dr Jim Armstrong's book

package MVL_package is
    type MVL is ('Z','0','1');
    type MVL_res_vec is array (integer range <>) of MVL;
    function MVL_res_func (S : MVL_res_vec) return MVL;
    end MVL_package;
package body MVL_package is
  function MVL_res_func (S : MVL_res_vec) return MVL is
    variable RESOLVED_VALUE : MVL := 'Z';
    begin
      for I in S'RANGE loop
        if S(I) /= 'Z' then
          RESOLVED_VALUE := S(I);
          exit;
        end if;
      end loop;
      return RESOLVED_VALUE;
    end MVL_res_func;
  end MVL_package;
use work.MVL_package.all;
entity test is end test;
architecture test of test is
    signal X : MVL_res_func MVL bus;
#1[    signal PHASE_@ : boolean;]
    begin
#1[    b@ : block(PHASE_@)
       begin
       X <= guarded '$2$1$0$' after 500 ps;
```

```
        end block b@;]
#1[     PHASE_@ <= true after 750 ps
#2[                ,$2$false$true$ after @ ns];]
    end test;
```

TEST NUMBER : 314

PATHNAME : [.BENCH.A.C.D1B]shell1.sh (see readme.txt in this directory)
(UNIX equivalent : bench/a/c/d1b/shell1.sh)

PURPOSE : Determine the effect of guarded blocks and register resolved signals on CPU time. The model simulated is a package declaration, a package body declaration, and an architecture. The package declaration has two type declarations and a resolution function declaration. The package body has the body of the resolution function. The architecture consists of a register resolved signal declaration, a number of non-resolved signal declarations (guards), a guarded block statement for each guard signal, and a signal assignment statement for each guard signal. Each block contains a signal assignment statement for the register resolved signal. The factors to be varied are the number of non-resolved signal declarations/number of blocks/number of signal assignment statements, and the length of time (in ns) to simulate the model. This description is part of an example from Dr Jim Armstrong's book.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of guarded blocks/number of guard signals
--    2 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",2,2
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        package MVL_package is
--                .
--        package body MVL_package is
--                .
--          end MVL_package;
--        use work.MVL_package.all;
--        entity test is end test;
--        architecture test of test is
--            signal X : MVL_res_func MVL register;
--            signal PHASE_1 : boolean;
--            signal PHASE_2 : boolean;
--            begin
--            b1 : block(PHASE_1)
```

```
--              begin
--              X <= guarded '1' after 500 ps;
--              end block b1;
--           b2 : block(PHASE_2)
--              begin
--              X <= guarded '0' after 500 ps;
--              end block b2;
--           PHASE_1 <= true after 750 ps
--                    ,false after 1 ns
--                    ,true after 2 ns;
--           PHASE_2 <= true after 750 ps
--                    ,false after 1 ns
--                    ,true after 2 ns;
--           end test;
--    This description is part of an example from Dr Jim Armstrong's book.


package MVL_package is
    type MVL is ('Z','0','1');
    type MVL_res_vec is array (integer range <>) of MVL;
    function MVL_res_func (S : MVL_res_vec) return MVL;
    end MVL_package;
package body MVL_package is
  function MVL_res_func (S : MVL_res_vec) return MVL is
    variable RESOLVED_VALUE : MVL := 'Z';
    begin
      for I in S'RANGE loop
        if S(I) /= 'Z' then
          RESOLVED_VALUE := S(I);
          exit;
        end if;
      end loop;
      return RESOLVED_VALUE;
    end MVL_res_func;
  end MVL_package;
use work.MVL_package.all;
entity test is end test;
architecture test of test is
    signal X : MVL_res_func MVL register;
#1[    signal PHASE_@ : boolean;]
    begin
#1[    b@ : block(PHASE_@)
       begin
       X <= guarded '$2$1$0$' after 500 ps;
       end block b@;]
#1[    PHASE_@ <= true after 750 ps
#2[             ,$2$false$true$ after @ ns];]
    end test;
```

```
TEST NUMBER :  315

PATHNAME :  [.BENCH.A.C.G2]shell.sh
            (UNIX equivalent : bench/a/c/g2/shell.sh)

PURPOSE :  Determine the effect of assertion statements on CPU time.  The model
           simulated is an architecture consisting of a number of signal
           declarations and a number of assertion statements for each signal.
           The factors to be varied are the number of signal declarations, the
           number of assertion statements per signal, and the length of time
           (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 23 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations
--    2 : number of assertion statements per signal declaration
--    3 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",3,2,5
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,3,2,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal sig1 : boolean := true;
--          signal sig2 : boolean := false;
--          signal sig3 : boolean := true;
--          signal stop : boolean := false;
--        begin
--          stop <= false,
--                  true after 5 ns;
--          assert ((sig1 = true) or (sig1 = false)) report "" severity note;
--          assert ((sig2 = true) or (sig2 = false)) report "" severity note;
--          assert ((sig3 = true) or (sig3 = false)) report "" severity note;
--          assert ((sig1 = true) or (sig1 = false)) report "" severity note;
--          assert ((sig2 = true) or (sig2 = false)) report "" severity note;
--          assert ((sig3 = true) or (sig3 = false)) report "" severity note;
--          assert (stop = false) report "simulation complete" severity failure;
--        end test;


entity test is end;
```

```
architecture test of test is
#1[  signal sig@ : boolean := $2$true$false$;]
  signal stop : boolean := false;
begin
  stop <= false,
          true after %3% ns;
#2[#1[  assert ((sig@ = true) or (sig@ = false)) report "" severity note;]]
  assert (stop = false) report "simulation complete" severity failure;
end test;
```

TEST NUMBER : 316

PATHNAME : [.BENCH.A.C.G4]shell.sh
(UNIX equivalent : bench/a/c/g4/shell.sh)

PURPOSE : Determine the effect of block assertion statements on CPU time. The model simulated is an architecture consisting of a number of block declarations. Each block contains a number of signal declarations and a number of assertion statements for each signal. The factors to be varied are the number of block declarations, the number of signal declarations per block, the number of assertion statements per signal, and the length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 24 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of blocks
--    2 : number of signal declarations per block
--    3 : number of assertion statements per signal declaration
--    4 : length of time (ns) to simulate model
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2,2,2,5
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2,2,2,5)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal stop : boolean := false;
--        begin
--          stop <= false,
--                  true after 5 ns;
--          assert (stop = false) report "simulation complete" severity failure;
--          b1 : block
--            signal sig1 : boolean := true;
--            signal sig2 : boolean := false;
--          begin
--            assert ((sig1 = true) or (sig1 = false)) report "" severity note;
--            assert ((sig2 = true) or (sig2 = false)) report "" severity note;
--            assert ((sig1 = true) or (sig1 = false)) report "" severity note;
--            assert ((sig2 = true) or (sig2 = false)) report "" severity note;
--          end block b1;
--          b2 : block
--            signal sig1 : boolean := true;
```

```
--            signal sig2 : boolean := false;
--          begin
--            assert ((sig1 = true) or (sig1 = false)) report "" severity note;
--            assert ((sig2 = true) or (sig2 = false)) report "" severity note;
--            assert ((sig1 = true) or (sig1 = false)) report "" severity note;
--            assert ((sig2 = true) or (sig2 = false)) report "" severity note;
--          end block b2;
--        end test;


entity test is end;
architecture test of test is
  signal stop : boolean := false;
begin
  stop <= false,
          true after %4% ns;
  assert (stop = false) report "simulation complete" severity failure;
#1[  b@ : block
#2[    signal sig@ : boolean := $2$true$false$;]
  begin
#3[#2[    assert ((sig@ = true) or (sig@ = false)) report "" severity note;]]
  end block b@;]
end test;
```

TEST NUMBER : 317

PATHNAME : [.BENCH.B.C.G6.I2.P10]shell.sh
(UNIX equivalent : bench/b/c/g6/i2/p10/shell.sh)

PURPOSE : Determine the effect of function assertion statements on CPU time. The model simulated is an architecture consisting of a number of function declarations and a process. Each function has an assertion statement for each input parameter and a return statement that returns the output of an XOR assignment statement with all the input parameters as arguments. The process has a number of bit variable declarations (the same number as the number of input parameters of each function) and a function call for each function declaration. The factors to be varied are the number of function declarations/ function calls and the number of variable declarations in the process/input parameters in each function/assertion statements in each function.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of function declarations in architecture/number of function
--        calls in process
--    2 : number of variable input parameters (-1) for each function/number
--        of assertion statements (-1) in each function
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2,3
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2,3)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          function xor1(var0 : in bit
--                        ;var1 : in bit
--                        ;var2 : in bit
--                        ;var3 : in bit) return bit is
--          begin
--            assert ((var0 = '0') or (var0 = '1')) report "" severity note;
--            assert ((var1 = '0') or (var1 = '1')) report "" severity note;
--            assert ((var2 = '0') or (var2 = '1')) report "" severity note;
--            assert ((var3 = '0') or (var3 = '1')) report "" severity note;
--            return(var0
--                        XOR var1
```

```
--                      XOR var2
--                      XOR var3);
--         end xor1;
--         function xor2(var0 : in bit
--                     ;var1 : in bit
--                     ;var2 : in bit
--                     ;var3 : in bit) return bit is
--         begin
--           assert ((var0 = '0') or (var0 = '1')) report "" severity note;
--           assert ((var1 = '0') or (var1 = '1')) report "" severity note;
--           assert ((var2 = '0') or (var2 = '1')) report "" severity note;
--           assert ((var3 = '0') or (var3 = '1')) report "" severity note;
--           return(var0
--                      XOR var1
--                      XOR var2
--                      XOR var3);
--         end xor2;
--       begin
--         process
--           variable var0 : bit := '0';
--           variable var1 : bit := '1';
--           variable var2 : bit := '0';
--           variable var3 : bit := '1';
--           variable out1 : bit;
--           variable out2 : bit;
--         begin
--           out1 := xor1(var0
--                      ,var1
--                      ,var2
--                      ,var3);
--           out2 := xor2(var0
--                      ,var1
--                      ,var2
--                      ,var3);
--           wait;
--         end process;
--       end test;


entity test is end;
architecture test of test is
#1[  function xor@(var0 : in bit
#2[              ;var@ : in bit]) return bit is
  begin
    assert ((var0 = '0') or (var0 = '1')) report "" severity note;
#2[    assert ((var@ = '0') or (var@ = '1')) report "" severity note;]
    return(var0
#2[               XOR var@]);
  end xor@;]
begin
  process
```

```
    variable var0 : bit := '0';
#2[    variable var@ : bit := '$2$1$0$';]
#1[    variable out@ : bit;]
  begin
#1[    out@ := xor@(var0
#2[                ,var@]);]
    wait;
  end process;
end test;
```

TEST NUMBER : 318

PATHNAME : [.BENCH.B.C.G5.H2.P10]shell.sh
(UNIX equivalent : bench/b/c/g/h2/p10/shell.sh)

PURPOSE : Determine the effect of procedure assertion statements on CPU time. The model simulated is an architecture consisting of a number of procedure declarations and a process. Each procedure has an assertion statement for each input parameter and an XOR assignment statement with all the input parameters as arguments and the output parameter as the result. The process has a number of bit variable declarations (the same number as the number of input parameters of each procedure) and a procedure call for each procedure declaration. The factors to be varied are the number of procedure declarations/ procedure calls and the number of variable declarations in the process/input parameters in each procedure/assertion statements in each procedure.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 29 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of procedure declarations in architecture/number of procedure
--        calls in process
--    2 : number of variable input parameters (-1) for each procedure/number
--        of assertion statements (-1) in each procedure
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",2,2
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,2,2)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          procedure xor1(var0 : in bit
--                         ;var1 : in bit
--                         ;var2 : in bit
--                         ;out_var : out bit) is
--          begin
--            assert ((var0 = '0') or (var0 = '1')) report "" severity note;
--            assert ((var1 = '0') or (var1 = '1')) report "" severity note;
--            assert ((var2 = '0') or (var2 = '1')) report "" severity note;
--            out_var := var0
--                               XOR var1
--                               XOR var2;
```

```
--          end xor1;
--          procedure xor2(var0 : in bit
--                        ;var1 : in bit
--                        ;var2 : in bit
--                        ;out_var : out bit) is
--          begin
--            assert ((var0 = '0') or (var0 = '1')) report "" severity note;
--            assert ((var1 = '0') or (var1 = '1')) report "" severity note;
--            assert ((var2 = '0') or (var2 = '1')) report "" severity note;
--            out_var := var0
--                               XOR var1
--                               XOR var2;
--          end xor2;
--        begin
--          process
--            variable var0 : bit := '0';
--            variable var1 : bit := '1';
--            variable var2 : bit := '0';
--            variable out1 : bit;
--            variable out2 : bit;
--          begin
--            xor1(var0
--                ,var1
--                ,var2
--                ,out1);
--            xor2(var0
--                ,var1
--                ,var2
--                ,out2);
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
#1[  procedure xor@(var0 : in bit
#2[                ;var@ : in bit]
                  ;out_var : out bit) is
  begin
    assert ((var0 = '0') or (var0 = '1')) report "" severity note;
#2[    assert ((var@ = '0') or (var@ = '1')) report "" severity note;]
    out_var := var0
#2[                     XOR var@];
  end xor@;]
begin
  process
    variable var0 : bit := '0';
#2[    variable var@ : bit := '$2$1$0$';]
#1[    variable out@ : bit;]
  begin
```

```
#1[    xor@(var@
#2[         ,var@]
         ,out@);]
    wait;
  end process;
end test;
```

```
TEST NUMBER :  319

PATHNAME :  [.BENCH.B.C.K.L1.M.O]shell.sh
            (UNIX equivalent : bench/b/c/k/l1/m/o/shell.sh)

PURPOSE :  Determine the effect of arrayed variables and use of the " 'val "
           attribute on CPU time.  The odel simulated is an architecture
           consisting of a process.  The process has a variable character
           array declaration and a for-loop, whose number of iterations is
           equal to the size of the array.  The for-loop contains an if-
           statement, a variable assignment statement using " 'val ", and
           another variable assignment statement.  The if-statement contains
           two variable assignment statements.  The factor to be varied is the
           size of the array/number of for-loop iterations.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :


--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 30 August 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of " 'val " statements to execute/number of variable
--        declarations in process
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",200
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,200)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--        begin
--          process
--            type var_type is array(1 to 200) of character;
--            variable var : var_type;
--            variable corrected_val : integer;
--            variable wrap : integer := 1;
--          begin
--            corrected_val := 32;
--            for i in 1 to 200 loop
--              if i > (95 * wrap) then
--                wrap := wrap + 1;
--                corrected_val := 32;
--              end if;
--              var(i) := character'val(corrected_val);
--              corrected_val := corrected_val + 1;
--            end loop;
```

```
--            wait;
--          end process;
--        end test;


entity test is end;
architecture test of test is
begin
  process
    type var_type is array(1 to %1%) of character;
    variable var : var_type;
    variable corrected_val : integer;
    variable wrap : integer := 1;
  begin
    corrected_val := 32;
    for i in 1 to %1% loop
      if i > (95 * wrap) then
        wrap := wrap + 1;
        corrected_val := 32;
      end if;
      var(i) := character'val(corrected_val);
      corrected_val := corrected_val + 1;
    end loop;
    wait;
  end process;
end test;
```

TEST NUMBER : 320

PATHNAME : [.BENCH.A.C.F1.K.Q]shell.sh
(UNIX equivalent : bench/a/c/f1/k/q/shell.sh)

PURPOSE : Determine the effect of arrayed, aliased signals, and component instantiations on CPU time. The model simulated is a mos logic package (containing various type definitions and function declarations), a clock entity with two port signals (one arrayed) whose architecture has a signal assignment statement for each element of the port signal array, and a top-level entity whose architecture has an arrayed signal declaration whose size is equal to the number of elements in the clock entity's port signal multiplied by the number of aliased signals desired. This architecture also has a number of aliased signals; each one represents part of the arrayed signal declared earlier in the architecture, and is the same size as the clock entity's port signal. The architecture also contains one component instantiation (of the clock entity) for each aliased signal. The factors to be varied are the size of the clock entity's arrayed port signal, the number of aliased signals/component instantiations in the top-level architecture, and the length of time (in ns) to simulate the model.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
--  AUTHOR : Captain Karen M. Serafino; code written by Captain Michael A. Dukes
--
--  Date : 11 September 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : bit_vector size
--    2 : number of aliased signals/component instantiations
--    3 : length of time (ns) to simulate model (must be > 2)
--
--  EXAMPLE :
--    $ sim gen/param="shell.sh","test.vhd",250,150,10
--    (UNIX equivalent : % sim gen -param=\"\shell.sh\"\,\"\test.vhd\"\,250,
--                         150,10)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        package MOS_logic_package2 is
--                    .
--                    .
--                end MOS_logic_package2;
--        use work.mos_logic_package2.all;
--        entity many_clock is
--            port (GO : in mos_node;
--                  clock : inout mos_node_vector (250 downto 1));
```

```
--        end many_clock;
--        architecture proc of many_clock is
--            begin
--            clock(1) <= snand(clock(1),go) after 1 ns;
--                                .
--                                .
--            clock(250) <= snand(clock(250),go) after 1 ns;
--            end proc;
--        use work mos_logic_package2.all;
--        entity top_many is
--        end top_many;
--        architecture struct of top_many is
--            signal go : mos_node;
--            signal massive_clock : mos_node_vector ((250 * 150) downto 1);
--            alias  clock1        : mos_node_vector(250 downto 1)
--                   is massive_clock((1 * 250) downto (((1-1) * 250)+1));
--                         .
--                         .
--            alias  clock150      : mos_node_vector(250 downto 1)
--                   is massive_clock((150 * 250) downto (((150-1) * 250)+1));
--          component many_clock
--            port    (GO : in mos_node;
--                     clock : inout mos_node_vector (250 downto 1));
--            end component;
--          for all : many_clock use entity work.many_clock(proc);
--            begin
--            process
--                begin
--                go.L.S <= 'D' after 1 ns;
--                go.L.V <= '0' after 1 ns, '1' after 2 ns, '0' after 10 ns;
--                assert (go.L.V = '0') severity error;
--                wait;
--                end process;
--          many_clock1: many_clock
--            port map  (GO => GO,
--                       clock => clock1);
--                    .
--                    .
--          many_clock150: many_clock
--            port map  (GO => GO,
--                       clock => clock150);
--            end struct;

package MOS_logic_package2 is

    type trans_type is ('P','N','U','D');

    --  For trans_type the following mean:
    --
    --
    --     P : Optimally sized p-type enhancement MOS.
```

```
--      N : Optimally sized n-type enhancement MOS.
--      U : Resistively sized p-type enhancement MOS (pullUp).
--      D : Resistively sized n-type enhancement MOS (pullDown).

    type strength is ('B', 'C', 'W', 'D');

    type value    is ('B', '0', '1', 'X');

    type capacitance is range 0 to 100000;

    type MOS_logic is record
        S : strength;
        V : value;
        end record;

    type MOS_node_record is record
        L : MOS_logic;
--        C : capacitance;
        end record;

     ne MOS_node_and_time is record
        N : MOS_node_record;
        T : time;
        end record;

    type MOS_node_array is array (Natural range <>) of
            MOS_node_and_time;

    type mos_node_resolution_array is array (integer range <>)
            of MOS_node_record;


    function mos_node_resolution (input : mos_node_resolution_array)
        return mos_node_record;

    subtype mos_node is mos_node_resolution mos_node_record;

    type mos_node_vector is array (Natural range <>) of mos_node;

    function snand (A,B : MOS_node_record) return MOS_node_record;

    function snor  (A,B : MOS_node_record) return MOS_node_record;

    function snot  (A : MOS_node_record)   return MOS_node_record;

    function synor (A,B : MOS_node_record) return MOS_node_record;

    function pnand (A,B : MOS_node_record) return MOS_node_record;

    function nnand (A,B : MOS_node_record) return MOS_node_record;
```

```
    function pnor  (A,B : MOS_node_record) return MOS_node_record;

    function pnot  (A : MOS_node_record)   return MOS_node_record;

    function tmux  (A,B,S,Sbar : MOS_node_record)
                                           return MOS_node_record;

    function dff   (A, PHI, PHI_bar, OUTput : MOS_node_record)
                                           return MOS_node_record;

    function binary_to_multi (A : bit)     return MOS_node_record;

    function multi_to_binary (Sig : bit; A : MOS_node_record)
                                           return bit;


    end MOS_logic_package2;


package body MOS_logic_package2 is

     function mos_node_resolution (input : mos_node_resolution_array)
         return mos_node_record is

         variable output, temp : mos_node_record;

         begin
         output.L.S := 'B';
         output.L.V := 'B';
         for i in input'range loop
             temp := input(i);
             If (temp.L.S > output.L.S) then
                 output := temp;
             elsif ((temp.L.S = output.L.S) and (temp.L.V /= output.L.V)) then
                 output.L.V := 'X';
                 end if;
             end loop;
         return(output);
         end mos_node_resolution;



------------------------------------------------------------------
------------------------------------------------------------------
--
--   Static CMOS functions used with multiple-valued logic
--
------------------------------------------------------------------
------------------------ ------------------------ --------------
```

```
    function snand      (A,B : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If(A.L.V = '1') and (B.L.V = '1') then
            temp.L.S := 'D';
            temp.L.V := '0';
        elsif (B.L.V = '0') or (A.L.V = '0') then
            temp.L.S := 'D';
            temp.L.V := '1';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end snand;



    function snor       (A,B : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If(A.L.V = '0') and (B.L.V = '0') then
            temp.L.S := 'D';
            temp.L.V := '1';
        elsif (B.L.V = '1') or (A.L.V = '1') then
            temp.L.S := 'D';
            temp.L.V := '0';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end snor;
```

```
    function snot       (A : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If (A.L.V = '0') then
            temp.L.S := 'D';
            temp.L.V := '1';
        elsif (A.L.V = '1') then
            temp.L.S := 'D';
            temp.L.V := '0';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end snot;



    function sxnor      (A,B : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If ((A.L.V = '0') and (B.L.V = '0')) or
           ((A.L.V = '1') and (B.L.V = '1')) then
            temp.L.S := 'D';
            temp.L.V := '1';
        elsif ((A.L.V = '1') and (B.L.V = '0')) or
              ((A.L.V = '0') and (B.L.V = '1')) then
            temp.L.S := 'D';
            temp.L.V := '0';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end sxnor;
```

```
------------------------------------------------------------
------------------------------------------------------------
--
--   pnMOS functions used with multiple-valued logic
--
------------------------------------------------------------
------------------------------------------------------------


    function pnand      (A,B : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If (A.L.V = '1') and (B.L.V = '1') then
            temp.L.S := 'D';
            temp.L.V := '0';
        elsif (A.L.V = '0') or (B.L.V = '0') then
            temp.L.S := 'W';
            temp.L.V := '1';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end pnand;



    function nnand      (A,B : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If (A.L.V = '1') and (B.L.V = '1') then
            temp.L.S := 'W';
            temp.L.V := '0';
        elsif (A.L.V = '0') or (B.L.V = '0') then
            temp.L.S := 'D';
            temp.L.V := '1';
```

```
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end nnand;


    function pnor       (A,B : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If (A.L.V = '1') or (B.L.V = '1') then
            temp.L.S := 'D';
            temp.L.V := '0';
        elsif (A.L.V = '0') and (B.L.V = '0') then
            temp.L.S := 'W';
            temp.L.V := '1';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
            end if;
        return(temp);

        end pnor;



    function pnot       (A : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If (A.L.V = '1') then
            temp.L.S := 'D';
            temp.L.V := '0';
        elsif (A.L.V = '0') then
            temp.L.S := 'W';
            temp.L.V := '1';
        else
            temp.L.S := 'W';
            temp.L.V := 'X';
```

```
            end if;
        return(temp);

        end pnot;




------------------------------------------------------------------
------------------------------------------------------------------
--
--   transmission-gate functions used with multiple-valued logic
--
------------------------------------------------------------------
------------------------------------------------------------------


    function tmux       (A,B,S,Sbar : MOS_node_record)
                         return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If ((S.L.V = '1') and (Sbar.L.V = '0')) then
            If not(A.L.S = 'B') then
                temp := A;
                end if;
        elsif ((S.L.V = '0') and (Sbar.L.V = '1')) then
            If not(B.L.S = 'B') then
                temp := B;
                end if;
        elsif (A.L.S = B.L.S) then
            If (A.L.V = B.L.V) then
                temp.L := A.L;
            else
                temp.L.V := 'X';
                temp.L.S := A.L.S;
            end if;
            If (temp.L.S = 'D') then
                If (S.L.V = '0') and (temp.L.V = '0') then
                    temp.L.S := 'W';
                elsif (S.L.V = '1') and (temp.L.V = '1') then
                    temp.L.S := 'W';
                elsif (S.L.V = 'X') then
                    temp.L.S := 'W';
                end if;
            end if;
```

```
        elsif (A.L.S > B.L.S) then
            temp.L := A.L;
            If (temp.L.S = 'D') then
                If (S.L.V = '0') and (temp.L.V = '0') then
                    temp.L.S := 'W';
                elsif (S.L.V = '1') and (temp.L.V = '1') then
                    temp.L.S := 'W';
                elsif (S.L.V = 'X') then
                    temp.L.S := 'W';
                end if;
            end if;
        else
            temp.L := B.L;
            If (temp.L.S = 'D') then
                If (S.L.V = '0') and (temp.L.V = '0') then
                    temp.L.S := 'W';
                elsif (S.L.V = '1') and (temp.L.V = '1') then
                    temp.L.S := 'W';
                elsif (S.L.V = 'X') then
                    temp.L.S := 'W';
                end if;
            end if;
        end if;

        return(temp);

        end tmux;



-------------------------------------------------------------------
-------------------------------------------------------------------
--
--   precharged functions used with multiple-valued logic
--
-------------------------------------------------------------------
-------------------------------------------------------------------




-------------------------------------------------------------------
-------------------------------------------------------------------
--
--   Other logic functions used with multiple-valued logic
--
-------------------------------------------------------------------
-------------------------------------------------------------------

    function dff        (A, PHI, PHI_bar, OUTput : MOS_node_record)
```

```
                        return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        If ((PHI.L.V = '1' and PHI_bar.L.V = '0')) then
            If (A.L.V = '0') then
                temp.L.S := 'D';
                temp.L.V := '1';
            elsif (A.L.V = '1') then
                temp.L.S := 'D';
                temp.L.V := '0';
            else
                temp.L.S := 'W';
                temp.L.V := 'X';
                end if;
        else
            temp.L := OUTput.L;
            end if;
        return(temp);

        end dff;



------------------------------------------------------------------
------------------------------------------------------------------
--
--   Logic translation for binary to multi and multi to binary
--
------------------------------------------------------------------
------------------------------------------------------------------


    function binary_to_multi (A : bit)
                        return MOS_node_record is

        variable temp : MOS_node_record;

        begin

--        temp.C := 0.0;
        temp.L.S := 'D';
        If (A = '1') then
            temp.L.V := '1';
        else
            temp.L.V := '0';
```

```
            end if;
        return(temp);

        end binary_to_multi;


    function multi_to_binary (Sig : bit;
                              A : MOS_node_record)
                        return bit is

        variable temp : bit;

        begin

        If (A.L.V = '1') then
            temp := '1';
        elsif (A.L.V = '0') then
            temp := '0';
        else
            temp := Sig;
            end if;
        return(temp);
        end multi_to_binary;

         end MOS_logic_package2;
use work.mos_logic_package2.all;
entity many_clock is
    port (GO : in mos_node;
          clock : inout mos_node_vector (%1% downto 1));
end many_clock;
architecture proc of many_clock is

    begin

--    process
--
--        begin
--        go.L.S <= 'D' after 1 ns;
--        go.L.V <= '0' after 1 ns, '1' after 2 ns, '0' after 50 ns;
--        wait;
--        end process;

#1[    clock(@) <= snand(clock(@),go) after 1 ns;]

    end proc;
use work.mos_logic_package2.all;
entity top_many is
end top_many;
architecture struct of top_many is

    signal go : mos_node;
```

```
    signal massive_clock : mos_node_vector ((%1% * %2%) downto 1);
#2[    alias  clock@        : mos_node_vector(%1% downto 1)
           is massive_clock((@ * %1%) downto (((@-1) * %1%)+1));]

  component many_clock
    port    (GO : in mos_node;
             clock : inout mos_node_vector (%1% downto 1));
    end component;

  for all : many_clock use entity work.many_clock(proc);

    begin

    process

        begin
        go.L.S <= 'D' after 1 ns;
        go.L.V <= '0' after 1 ns, '1' after 2 ns, '0' after %3% ns;
        assert (go.L.V = '0') severity error;
        wait;
        end process;

#2[  many_clock@: many_clock
    port map  (GO => GO,
               clock => clock@);]

    end struct;
```

TEST NUMBER : 321

PATHNAME : [.BENCH.A.C.F1.J] NO SHELL FILE - SEE
(UNIX equivalent : bench/a/c/f1/j) readme.txt IN THIS
DIRECTORY

PURPOSE : Determine the effect of two levels of component hierarchy
on simulation time. The components are 4-bit multipliers.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
entity two_level is end;
architecture two of two_level is
  signal in1a,in1b,in2a,in2b,in3a,in3b,out1,out2,out3 : integer :=0;
  component test
    port (a_integer : in integer;
  b_integer : in integer;
  result_integer : out integer);
  end component;
  for all : test use entity work.test(proc);
begin
test1 : test
  port map (a_integer => in1a,
    b_integer => in1b,
    result_integer => out1);
test2 : test
  port map (a_integer => in2a,
    b_integer => in2b,
    result_integer => out2);
test3 : test
  port map (a_integer => out1,
    b_integer => out2,
    result_integer => out3);
in1a <= 0,
15 after 100 ns,
5 after 150 ns,
3 after 200 ns,
4 after 250 ns,
2 after 300 ns,
4 after 350 ns,
16 after 400 ns,
2 after 450 ns,
5 after 500 ns;

in1b <= 0,
15 after 50 ns,
0 after 100 ns,
3 after 150 ns,
```

```
5 after 200 ns,
4 after 250 ns,
2 after 350 ns,
16 after 400 ns,
15 after 500 ns;

in2a <= 0,
15 after 100 ns,
5 after 150 ns,
3 after 200 ns,
4 after 250 ns,
2 after 300 ns,
4 after 350 ns,
16 after 400 ns,
2 after 450 ns,
5 after 500 ns;
in2b <= 0,
15 after 50 ns,
0 after 100 ns,
3 after 150 ns,
5 after 200 ns,
4 after 250 ns,
2 after 350 ns,
16 after 400 ns,
15 after 500 ns;
end two;
```

TEST NUMBER : 322

PATHNAME : [.BENCH.A.C.F1.J] NO SHELL FILE - SEE
(UNIX equivalent : bench/a/c/f1/j) readme.txt IN THIS
DIRECTORY

PURPOSE : Determine the effect of five levels of component hierarchy
on simulation time. The components are 4-bit multipliers.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
      .
entity five_level is end;
architecture five of five_level is
  signal in1a,in1b,in2a,in2b,in3a,in3b,in4a,in4b,in5a,in5b,in6a,in6b,
         in7a,in7b,in8a,in8b,in9a,in9b,in10a,in10b,in11a,in11b,in12a,in12b,
         in13a,in13b,in14a,in14b,in15a,in15b,in16a,in16b,out1,out2,out3,out4,
 out5,out6,out7,out8,out9,out10,out11,out12,out13,out14,out15,out16,
         out17,out18,out19,out20,out21,out22,out23,out24,out25,out26,out27,
         out28,out29,out30,out31 : integer :=0;
  component test
    port (a_integer : in integer;
  b_integer : in integer;
  result_integer : out integer);
  end component;
  for all : test use entity work.test(proc);
begin
test1 : test
  port map (a_integer => in1a,
    b_integer => in1b,
    result_integer => out1);
            .
            .
test31 : test
  port map (a_integer => out29,
            b_integer => out30,
            result_integer => out31);
in1a <= 0,
15 after 100  ns,
5 after 150  ns,
3 after 200  ns,
4 after 250  ns,
2 after 300  ns,
4 after 350  ns,
16 after 400  ns,
2 after 450  ns,
5 after 500  ns;
            .
            .
```

```
in16b <= 0,
15 after 50 ns,
0 after 100 ns,
3 after 150 ns,
5 after 200 ns,
4 after 250 ns,
2 after 350 ns,
16 after 400 ns,
15 after 500 ns;
end five;
```

TEST NUMBER : 323

PATHNAME : [.BENCH.A.C.F1.J] NO SHELL FILE - SEE
(UNIX equivalent : bench/a/c/f1/j) readme.txt IN THIS
DIRECTORY

PURPOSE : Determine the effect of ten levels of component hierarchy
on simulation time. The components are 4-bit multipliers.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

```
entity ten_level is end;
architecture ten of ten_level is
  signal in1a,in2a,in3a,in4a,in5a,in6a,in7a,in8a,in9a,in10a,in11a,in12a,in13a,
         in14a,in15a,in16a,in17a,in18a,in19a,in20a,in21a,in22a,in23a,in24a,
 in25a,in26a,in27a,in28a,in29a,in30a,in31a,in32a : integer := 0;
  signal out1,out2,out3,out4,out5,out6,out7,out8,out9,out10,out11,out12,
 out13,out14,out15,out16,out17,out18,out19,out20,out21,out22,out23,
 out24,out25,out26,out27,out28,out29,out30,out31,out32,out33
   : integer := 0;
component five_level
    port (in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,in11,in12,in13,in14,in15,
          in16,in17,in18,in19,in20,in21,in22,in23,in24,in25,in26,in27,in28,
  in29,in30,in31,in32 : in integer;
  result              : out integer);
  end component;
  for all : five_level use entity work.five_level(five);
begin
five1 : five_level
  port map (in1 => in1a,
    in2 => in2a,
    in3 => in3a,
    in4 => in4a,
    in5 => in5a,
    in6 => in6a,
    in7 => in7a,
    in8 => in8a,
    in9 => in9a,
    in10 => in10a,
    in11 => in11a,
    in12 => in12a,
    in13 => in13a,
    in14 => in14a,
    in15 => in15a,
    in16 => in16a,
    in17 => in17a,
    in18 => in18a,
    in19 => in19a,
```

```
    in20 => in20a,
    in21 => in21a,
    in22 => in22a,
    in23 => in23a,
    in24 => in24a,
    in25 => in25a,
    in26 => in26a,
    in27 => in27a,
    in28 => in28a,
    in29 => in29a,
    in30 => in30a,
    in31 => in31a,
    in32 => in32a,
    result => out1);
              .
              .
five33 : five_level
  port map (in1 => out1,
    in2 => out2,
    in3 => out3,
    in4 => out4,
    in5 => out5,
    in6 => out6,
    in7 => out7,
    in8 => out8,
    in9 => out9,
    in10 => out10,
    in11 => out11,
    in12 => out12,
    in13 => out13,
    in14 => out14,
    in15 => out15,
    in16 => out16,
    in17 => out17,
    in18 => out18,
    in19 => out19,
    in20 => out20,
    in21 => out21,
    in22 => out22,
    in23 => out23,
    in24 => out24,
    in25 => out25,
    in26 => out26,
    in27 => out27,
    in28 => out28,
    in29 => out29,
    in30 => out30,
    in31 => out31,
    in32 => out32,
    result => out33);
in1a <= 0,
```

```
15 after 100 ns,
5 after 150 ns,
3 after 200 ns,
4 after 250 ns,
2 after 300 ns,
4 after 350 ns,
16 after 400 ns,
2 after 450 ns,
5 after 500 ns;
              .
              .
in32a <= 0,
15 after 50 ns,
0 after 100 ns,
3 after 150 ns,
5 after 200 ns,
4 after 250 ns,
2 after 350 ns,
16 after 400 ns,
15 after 500 ns;
end ten;
```

```
TEST NUMBER :  324

PATHNAME :  [.BENCH.B.C]shell0.sh
            (UNIX equivalent : bench/b/c/shell0.sh)

PURPOSE :  Use with benchmark # 325 to determine the simulation time
           differences between processes with sensitivity lists versus
           processes with explicit wait statements.  This description
           uses explicit wait statements.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 17 November 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal assignment statements/processes
--    2 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell0.sh","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell0.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal go : bit := '1';
--          signal s1 : bit;
--          signal s2 : bit;
--          signal s3 : bit;
--        begin
--          p1 : process
--               begin
--                 s1 <= s1 nand go after 1 ns;
--                 wait on s1;
--               end process p1;
--          p2 : process
--               begin
--                 s2 <= s2 nand go after 1 ns;
--                 wait on s2;
--               end process p2;
--          p3 : process
--               begin
--                 s3 <= s3 nand go after 1 ns;
--                 wait on s3;
--               end process p3;
--          go <= '1' after 1 ns,
```

```
--                '0' after 4 ns;
--        end test;


entity test is end;
architecture test of test is
  signal go : bit := '1';
#1[  signal s@ : bit;]
begin
#1[  p@ : process
       begin
         s@ <= s@ nand go after 1 ns;
         wait on s@;
       end process p@;]
  go <= '1' after 1 ns,
        '0' after %2% ns;
end test;
```

```
TEST NUMBER :  325

PATHNAME :  [.BENCH.B.C]shell1.sh
            (UNIX equivalent : bench/b/c/shell1.sh)

PURPOSE :  Use with benchmark # 324 to determine the simulation time
           differences between processes with sensitivity lists versus
           processes with explicit wait statements.  This description
           uses sensitivity lists.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

--  AUTHOR : Captain Karen M. Serafino
--
--  Date : 17 November 1989
--
--  PARAMETER NUMBER MEANING :
--    1 : number of signal declarations/signal assignment statements/processes
--    2 : length of time (ns) to simulate model (must be > 1)
--
--  EXAMPLE :
--    $ sim gen/param="shell1.sh","test.vhd",3,4
--    (UNIX equivalent : % sim gen -param=\"\shell1.sh\"\,\"\test.vhd\"\,3,4)
--      will generate a model in file "test.vhd" with an architecture
--      in the form :
--        entity test is end;
--        architecture test of test is
--          signal go : bit := '1';
--          signal s1 : bit;
--          signal s2 : bit;
--          signal s3 : bit;
--        begin
--          p1 : process(s1)
--               begin
--                 s1 <= s1 nand go after 1 ns;
--               end process p1;
--          p2 : process(s2)
--               begin
--                 s2 <= s2 nand go after 1 ns;
--               end process p2;
--          p3 : process(s3)
--               begin
--                 s3 <= s3 nand go after 1 ns;
--               end process p3;
--          go <= '1' after 1 ns,
--                '0' after 4 ns;
--        end test;
```

```
entity test is end;
architecture test of test is
  signal go : bit := '1';
#1[  signal s@ : bit;]
begin
#1[  p@ : process(s@)
       begin
         s@ <= s@ nand go after 1 ns;
       end process p@;]
  go <= '1' after 1 ns,
        '0' after %2% ns;
end test;
```

TEST NUMBER : 326 NO SHELL FILE - SEE "readme.txt" IN THIS DIRECTORY

PATHNAME : [.BENCH.D1A.F1.H.I.K.Q]fpa_file.vhd
(UNIX equivalent : bench/d1a/f1/h/i/k/q/fpa_file.vhd)

PURPOSE : Determine CPU time required to simulate a behavioral description of a 64-bit floating-point adder that handles double precision IEEE standard floating-point notation numbers.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

Too much code for hardcopy.

TEST NUMBER : 327 NO SHELL FILE - SEE "readme.txt" IN THIS DIRECTORY

PATHNAME : [.BENCH.F1.F3.I2.J]
(UNIX equivalent : bench/f1/f3/i2/j)

PURPOSE : Determine CPU time required to simulate a structural description of an 8-bit ALU with two inputs (A and B), three outputs (C, overflow, and zero), and 8 opcodes : output zeroes, output ones, complement A, A + B, A * B, A < B, A < or = B, A - B.

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

Too much code for hardcopy.

TEST NUMBER : 328

PATHNAME : [.BENCH.A.B.C.F1]form9.vhd (no shell file; see
(UNIX equivalent : bench/a/b/c/f1/form9.vhd) readme.txt)

PURPOSE : Determine CPU time required to simulate a structural description of a 9-input parity checker. This model has no parameters to vary, and contains 1275 gates (components).

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

Too much code for hardcopy.

TEST NUMBER : 329

PATHNAME : [.BENCH.A.B.C.F1]form11.vhd (no shell file;see
(UNIX equivalent : bench/a/b/c/f1/form11.vhd) readme.txt)

PURPOSE : Determine CPU time required to simulate a structural
description of a 11-input parity checker. This model
has no parameters to vary, and contains 5115 gates
(components).

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

Too much code for hardcopy.

TEST NUMBER : 330

PATHNAME : [.BENCH.A.B.C.F1]form12.vhd (no shell file;see
(UNIX equivalent : bench/a/b/c/f1/form12.vhd) readme.txt)

PURPOSE : Determine CPU time required to simulate a structural description of a 12-input parity checker. This model has no parameters to vary, and contains 10235 gates (components).

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

Too much code for hardcopy.

TEST NUMBER : 331

PATHNAME : [.BENCH.A.B.C.F1]form14.vhd (no shell file;see
(UNIX equivalent : bench/a/b/c/f1/form14.vhd) readme.txt)

PURPOSE : Determine CPU time required to simulate a structural
description of a 14-input parity checker. This model
has no parameters to vary, and contains 40955 gates
(components).

EXPECTED RESULTS :

UNITS OF MEASUREMENT :

Too much code for hardcopy.

"gen.vhd" :

```
entity gen is
  generic(ifile_name : string(1 to 100);
          ofile_name : string(1 to 100);
          par1 : integer := 0;
          par2 : integer := 0;
          par3 : integer := 0;
          par4 : integer := 0;
          par5 : integer := 0;
          par6 : integer := 0;
          par7 : integer := 0;
          par8 : integer := 0;
          par9 : integer := 0;
          par10 : integer := 0;
          par11 : integer := 0);
end gen;
architecture gen of gen is
  type char_file is file of character;
  subtype para_num_range is integer range 1 to 11;
  type vhdl_line is array(0 to 1000) of character;
  file in_file : char_file is in ifile_name;
  file out_file : char_file is out ofile_name;
  subtype tog_vals is integer range 1 to 10;
  subtype tog_lengths is integer range 0 to 50;
  type tog_val_type is array(tog_vals,tog_lengths) of character;
  type tog_val_length_type is array (tog_vals) of tog_lengths;
  type label_length_type is array(1 to 26) of character;
  constant label_length_test : label_length_type :=
                                          "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

procedure get_seq_num_length(iteration_num : in integer;
                             seq_num_length : out integer) is
  variable power_of_ten : integer := 1;
  begin
    while (iteration_num / (10 ** power_of_ten))  /= 0 loop
      power_of_ten := power_of_ten + 1;
    end loop;
    seq_num_length := power_of_ten;
  end get_seq_num_length;

procedure get_integer(lin : in vhdl_line;
                      ptr : inout integer range vhdl_line'range;
                      delim : in character;
                      out_integer : inout integer) is
  variable power_of_ten : natural := 0;
  begin
    out_integer := 0;
    while lin(ptr) /= delim loop
      out_integer := (out_integer * (10 ** power_of_ten)) +
```

```
                    character'pos(lin(ptr)) - 48;
      power_of_ten := power_of_ten + 1;
      ptr := ptr + 1;
    end loop;
  end get_integer;

procedure file_get_integer(delim : in character;
                           out_integer : inout integer) is
  variable power_of_ten : natural := 0;
  variable ch : character;
  begin
    out_integer := 0;
    read(in_file,ch);
    while ch /= delim loop
      out_integer := (out_integer * (10 ** power_of_ten)) +
                     character'pos(ch) - 48;
      power_of_ten := power_of_ten + 1;
      read(in_file,ch);
    end loop;
  end file_get_integer;

procedure get_toggle_values(lin : in vhdl_line;
                            ptr : inout integer range vhdl_line'range;
                            length : inout tog_val_length_type;
                            val : inout tog_val_type;
                            tog_limit : inout tog_vals) is
  begin
    ptr := ptr + 1;
    get_integer(lin,ptr,'$',tog_limit);
    for i in tog_vals'low to tog_limit loop
      length(i) := 0;
      ptr := ptr + 1;
      while lin(ptr) /= '$' loop
        length(i) := length(i) + 1;
        val(i,length(i)) := lin(ptr);
        ptr := ptr +  1;
      end loop;
    end loop;
  end get_toggle_values;

procedure insert_toggle_value(lin : in vhdl_line;
                              ptr : inout integer range vhdl_line'range;
                              length : inout tog_val_length_type;
                              val : inout tog_val_type;
                              first_toggle : inout boolean;
                              current_toggle : inout tog_vals;
                              tog_limit : inout tog_vals) is
  type toggle_type is array (tog_lengths) of character;
  variable toggle : toggle_type;
  variable toggle_length : tog_lengths;
  variable tog_limit_length : integer;
```

```
  begin
  if first_toggle then
    first_toggle := false;
    get_toggle_values(lin,ptr,length,val,tog_limit);
    toggle_length := length(tog_vals'low);
    for i in 1 to toggle_length loop
      toggle(i) := val(tog_vals'low,i);
    end loop;
    current_toggle := tog_vals'low;
  else
    get_seq_num_length(tog_limit,tog_limit_length);
    ptr := ptr + tog_limit_length + tog_limit + 1;
    for i in tog_vals loop
      ptr := ptr + length(i);
    end loop;
    if current_toggle = tog_limit then
      current_toggle := tog_vals'low;
    else
      current_toggle := current_toggle + 1;
    end if;
    toggle_length := length(current_toggle);
    for i in 1 to toggle_length loop
      toggle(i) := val(current_toggle,i);
    end loop;
  end if;
  for i in 1 to toggle_length loop
    write(out_file,toggle(i));
  end loop;
  end insert_toggle_value;

procedure insert_num(iteration_num : in integer;
                     seq_num_length : in integer) is
  variable cur_seq_num : integer;
  variable cur_digit : integer range 0 to 9;
  type num_string_type is array(1 to seq_num_length) of character;
  variable num_string : num_string_type;
  begin
    cur_seq_num := iteration_num;
    for i in num_string'high downto 1 loop
      cur_digit := cur_seq_num mod 10;
      num_string(i) := character'val(cur_digit + 48);
      cur_seq_num := cur_seq_num/10;
    end loop;
    for i in 1 to num_string'high loop
      write(out_file,num_string(i));
    end loop;
  end insert_num;

procedure get_iteration_num(para_num : in para_num_range;
                            iteration_num : out integer) is
  begin
```

```
    case para_num is
      when 1 => iteration_num := par1;
      when 2 => iteration_num := par2;
      when 3 => iteration_num := par3;
      when 4 => iteration_num := par4;
      when 5 => iteration_num := par5;
      when 6 => iteration_num := par6;
      when 7 => iteration_num := par7;
      when 8 => iteration_num := par8;
      when 9 => iteration_num := par9;
      when 10 => iteration_num := par10;
      when 11 => iteration_num := par11;
    end case;
  end get_iteration_num;


procedure same_line(lin : in vhdl_line;
                    ptr : inout integer range vhdl_line'range;
                    file_io : in boolean) is
  variable para_num : para_num_range;
  variable label_length : integer;
  variable label_ptr : integer range label_length_type'range;
begin
  if file_io then
    file_get_integer('?',para_num);
  else
    get_integer(lin,ptr,'?',para_num);
  end if;
  get_iteration_num(para_num,label_length);
  label_ptr := label_length_type'low;
  for i in 1 to label_length loop
    write(out_file,label_length_test(label_ptr));
    if label_ptr = label_length_type'high then
      label_ptr := label_length_type'low;
    else
      label_ptr := label_ptr + 1;
    end if;
  end loop;
end same_line;
procedure process_block(iteration_num : in integer;
                        lin : inout vhdl_line;
                        char_count : inout integer range vhdl_line'range;
                        seq_num_length : in integer;
                        indent : in integer range 0 to vhdl_line'high;
                        first_iter : in boolean;
                        length : inout tog_val_length_type;
                        val : inout tog_val_type;
                        first_toggle : in boolean;
                        current_toggle : inout tog_vals;
                        tog_limit : inout tog_vals;
                        same_line_flag : out boolean) is
```

```
procedure save_sub(blk : in vhdl_line;
                   char_count : inout integer range vhdl_line'range;
                   indent : in integer range 0 to vhdl_line'high;
                   para_num_length : out integer) is
  variable ch : character;
  variable seq_num_length : integer;
  variable lin : vhdl_line;
  variable iteration_num : integer;
  variable block_count : integer;
  variable blk_counter : integer;
  variable cur_indent : integer range 0 to vhdl_line'high := indent;
  variable first_iter : boolean := false;
  variable para_num : para_num_range;
  variable first_toggle : boolean;
  variable length : tog_val_length_type;
  variable val : tog_val_type;
  variable current_toggle : tog_vals;
  variable tog_limit : tog_vals;
  variable same_line_flag : boolean;
  begin
    blk_counter := 1;
    get_integer(blk,blk_counter,'[',para_num);
    get_seq_num_length(para_num,para_num_length);
    get_iteration_num(para_num,iteration_num);
    assert (blk(blk_counter) = '[') report "syntax error" severity error;
    blk_counter := blk_counter + 1;
    char_count := 0;
    block_count := 0;
    ch := blk(blk_counter);
    blk_counter := blk_counter + 1;
    while (ch /= ']') or (block_count /=0) loop
      char_count := char_count + 1;
      lin(char_count) := ch;
      if ch = '#' then
        block_count := block_count + 1;
      elsif ch = ']' then
        block_count := block_count - 1;
      end if;
      ch := blk(blk_counter);
      blk_counter := blk_counter + 1;
    end loop;
    if cur_indent = 0 then
      first_iter := true;
    end if;
    first_toggle := true;
    same_line_flag := false;
    for i in 1 to iteration_num loop
      get_seq_num_length(i,seq_num_length);
      process_block(i,lin,char_count,seq_num_length,cur_indent,first_iter,
                    length,val,first_toggle,current_toggle,tog_limit,
                    same_line_flag);
```

```
    first_toggle := false;
    first_iter := false;
  end loop;
end save_sub;
variable counter : integer;
variable temp_count : integer range 0 to char_count;
variable temp_block : vhdl_line;
variable cur_indent : integer range 0 to vhdl_line'high := indent;
variable para_num : para_num_range;
variable limit : integer;
variable limit_length : integer;
variable local_first_toggle : boolean := first_toggle;
variable para_num_length : integer;
begin
  counter := 0;
  temp_count := 0;
  if lin(1) = '~' then
    counter := counter + 1;
    same_line_flag := true;
  else
    same_line_flag := false;
    if not first_iter then
      write(out_file,lf);
    end if;
    for i in 1 to indent loop
      write(out_file,' ');
    end loop;
  end if;
  while counter < char_count loop
    counter := counter + 1;
    if lin(counter) = '#' then
      if counter > 1 then
        if lin(counter - 1) /= lf then
          cur_indent := indent + counter;
        end if;
      end if;
      for i in (counter + 1) to char_count loop
        temp_block(i - counter) := lin(i);
      end loop;
      save_sub(temp_block,temp_count,cur_indent,para_num_length);
      counter := counter + 2 + temp_count + para_num_length;
    elsif lin(counter) = '@' then
      insert_num(iteration_num,seq_num_length);
    elsif lin(counter) = '%' then
      counter := counter + 1;
      get_integer(lin,counter,'%',para_num);
      get_iteration_num(para_num,limit);
      get_seq_num_length(limit,limit_length);
      insert_num(limit,limit_length);
    elsif lin(counter) = '$' then
      insert_toggle_value(lin,counter,length,val,local_first_toggle,
```

```
                              current_toggle,tog_limit);
      elsif lin(counter) = '?' then
        counter := counter + 1;
        same_line(lin,counter,false);
      else
        write(out_file,lin(counter));
      end if;
    end loop;
  end process_block;


procedure save_block(char_count : inout integer range vhdl_line'range;
                     indent : in integer range 0 to vhdl_line'high) is
  variable ch : character;
  variable lin : vhdl_line;
  variable para_num : para_num_range;
  variable seq_num_length : integer;
  variable iteration_num : integer;
  variable block_count : integer;
  variable cur_indent : integer range 0 to vhdl_line'high := indent;
  variable first_iter : boolean := false;
  variable first_toggle : boolean;
  variable length : tog_val_length_type;
  variable val : tog_val_type;
  variable current_toggle : tog_vals;
  variable tog_limit : tog_vals;
  variable same_line_flag : boolean;
  begin
    file_get_integer('[',para_num);
    get_iteration_num(para_num,iteration_num);
    char_count := 0;
    block_count := 0;
    read(in_file,ch);
    while (ch /= ']') or (block_count /=0) loop
      char_count := char_count + 1;
      lin(char_count) := ch;
      if ch = '#' then
        block_count := block_count + 1;
      elsif ch = ']' then
        block_count := block_count - 1;
      end if;
      read(in_file,ch);
    end loop;
    if cur_indent = 0 then
      first_iter := true;
    end if;
    first_toggle := true;
    same_line_flag := false;
    for i in 1 to iteration_num loop
      get_seq_num_length(i,seq_num_length);
      process_block(i,lin,char_count,seq_num_length,cur_indent,first_iter,
```

```
                    length,val,first_toggle,current_toggle,tog_limit,
                    same_line_flag);
      first_toggle := false;
      first_iter := false;
    end loop;
  end save_block;


begin
  process
  variable indent : integer range 0 to vhdl_line'high := 0;
  variable ch : character;
  variable char_count : integer range vhdl_line'range;
  variable para_num : para_num_range;
  variable limit : integer;
  variable limit_length : integer;
  variable already_read : boolean := false;
  variable dummy_lin : vhdl_line;
  variable dummy_ptr : integer range vhdl_line'range;
  begin
    if not endfile(in_file) then
      if not already_read then
        read(in_file,ch);
      end if;
      already_read := false;
      if ch = '-' then
        read(in_file,ch);
        if ch = '-' then
          while ch /= lf loop
            read(in_file,ch);
          end loop;
          if indent /= 0 then
            write(out_file,lf);
          end if;
          indent := 0;
        else
          write(out_file,'-');
          indent := indent + 1;
          already_read := true;
        end if;
      elsif ch = '#' then
        save_block(char_count,indent);
        indent := indent + char_count;
      elsif ch = '%' then
        file_get_integer('%',para_num);
        get_iteration_num(para_num,limit);
        get_seq_num_length(limit,limit_length);
        insert_num(limit,limit_length);
        indent := indent + limit_length;
      elsif ch = '?' then
        same_line(dummy_lin,dummy_ptr,true);
```

```
      else
        write(out_file,ch);
        indent := indent + 1;
        if ch = lf then
          indent := 0;
        end if;
      end if;
    assert not endfile(in_file) report "done" severity failure;
    end if;
  end process;
end gen;


"front_end.vhd" :

entity front_end is end;
architecture front_end of front_end is
  component gen
    generic(ifile_name : string(1 to 46);
            ofile_name : string(1 to 8);
            par1,par2,par3,par4,par5,
            par6,par7,par8,par9,par10,par11 : integer := 0);
  end component;
  for all : gen use entity work.gen(gen);
  constant ifile : string := "/home/elsun1/serafino/suite/bench/a/c/shell.sh";
  constant ofile : string := "test.vhd";
  constant p1 : integer := 3;
  constant p2 : integer := 2;
  constant p3 : integer := 0;
  constant p4 : integer := 0;
  constant p5 : integer := 0;
  constant p6 : integer := 0;
  constant p7 : integer := 0;
  constant p8 : integer := 0;
  constant p9 : integer := 0;
  constant p10 : integer := 0;
  constant p11 : integer := 0;
begin
  gen_comp : gen generic map(ifile,ofile,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11);
end front_end;



"alternate_gen.vhd" :

entity gen is
  generic(ifile_name : string(1 to 46);
          ofile_name : string(1 to 8);
          par1 : integer := 3;
          par2 : integer := 2;
          par3 : integer := 0;
```

```
          par4 : integer := 0;
          par5 : integer := 0;
          par6 : integer := 0;
          par7 : integer := 0;
          par8 : integer := 0;
          par9 : integer := 0;
          par10 : integer := 0;
          par11 : integer := 0);
end gen;
architecture gen of gen is
  type char_file is file of character;
  subtype para_num_range is integer range 1 to 11;
  type vhdl_line is array(0 to 1000) of character;
  file in_file : char_file is in ifile_name;
  file out_file : char_file is out ofile_name;
  subtype tog_vals is integer range 1 to 10;
  subtype tog_lengths is integer range 0 to 50;
  type tog_val_type is array(tog_vals,tog_lengths) of character;
  type tog_val_length_type is array (tog_vals) of tog_lengths;
  type label_length_type is array(1 to 26) of character;
  constant label_length_test : label_length_type :=
                                          "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

procedure get_seq_num_length(iteration_num : in integer;
                             seq_num_length : out integer) is
  variable power_of_ten : integer := 1;
  begin
    while (iteration_num / (10 ** power_of_ten))  /= 0 loop
      power_of_ten := power_of_ten + 1;
    end loop;
    seq_num_length := power_of_ten;
  end get_seq_num_length;

procedure get_integer(lin : in vhdl_line;
                      ptr : inout integer range vhdl_line'range;
                      delim : in character;
                      out_integer : inout integer) is
  variable power_of_ten : natural := 0;
  begin
    out_integer := 0;
    while lin(ptr) /= delim loop
      out_integer := (out_integer * (10 ** power_of_ten)) +
                      character'pos(lin(ptr)) - 48;
      power_of_ten := power_of_ten + 1;
      ptr := ptr + 1;
    end loop;
  end get_integer;

procedure file_get_integer(delim : in character;
                           out_integer : inout integer) is
  variable power_of_ten : natural := 0;
```

```
  variable ch : character;
  begin
    out_integer := 0;
    read(in_file,ch);
    while ch /= delim loop
      out_integer := (out_integer * (10 ** power_of_ten)) +
                     character'pos(ch) - 48;
      power_of_ten := power_of_ten + 1;
      read(in_file,ch);
    end loop;
  end file_get_integer;

procedure get_toggle_values(lin : in vhdl_line;
                            ptr : inout integer range vhdl_line'range;
                            length : inout tog_val_length_type;
                            val : inout tog_val_type;
                            tog_limit : inout tog_vals) is
  begin
    ptr := ptr + 1;
    get_integer(lin,ptr,'$',tog_limit);
    for i in tog_vals'low to tog_limit loop
      length(i) := 0;
      ptr := ptr + 1;
      while lin(ptr) /= '$' loop
        length(i) := length(i) + 1;
        val(i,length(i)) := lin(ptr);
        ptr := ptr + 1;
      end loop;
    end loop;
  end get_toggle_values;

procedure insert_toggle_value(lin : in vhdl_line;
                              ptr : inout integer range vhdl_line'range;
                              length : inout tog_val_length_type;
                              val : inout tog_val_type;
                              first_toggle : inout boolean;
                              current_toggle : inout tog_vals;
                              tog_limit : inout tog_vals) is
  type toggle_type is array (tog_lengths) of character;
  variable toggle : toggle_type;
  variable toggle_length : tog_lengths;
  variable tog_limit_length : integer;
  begin
  if first_toggle then
    first_toggle := false;
    get_toggle_values(lin,ptr,length,val,tog_limit);
    toggle_length := length(tog_vals'low);
    for i in 1 to toggle_length loop
      toggle(i) := val(tog_vals'low,i);
    end loop;
    current_toggle := tog_vals'low;
```

```
    else
      get_seq_num_length(tog_limit,tog_limit_length);
      ptr := ptr + tog_limit_length + tog_limit + 1;
      for i in tog_vals loop
        ptr := ptr + length(i);
      end loop;
      if current_toggle = tog_limit then
        current_toggle := tog_vals'low;
      else
        current_toggle := current_toggle + 1;
      end if;
      toggle_length := length(current_toggle);
      for i in 1 to toggle_length loop
        toggle(i) := val(current_toggle,1);
      end loop;
    end if;
    for i in 1 to toggle_length loop
      write(out_file,toggle(i));
    end loop;
    end insert_toggle_value;

  procedure insert_num(iteration_num : in integer;
                       seq_num_length : in integer) is
    variable cur_seq_num : integer;
    variable cur_digit : integer range 0 to 9;
    type num_string_type is array(1 to seq_num_length) of character;
    variable num_string : num_string_type;
    begin
      cur_seq_num := iteration_num;
      for i in num_string'high downto 1 loop
        cur_digit := cur_seq_num mod 10;
        num_string(i) := character'val(cur_digit + 48);
        cur_seq_num := cur_seq_num/10;
      end loop;
      for i in 1 to num_string'high loop
        write(out_file,num_string(i));
      end loop;
    end insert_num;

  procedure get_iteration_num(para_num : in para_num_range;
                              iteration_num : out integer) is
    begin
      case para_num is
        when 1 => iteration_num := par1;
        when 2 => iteration_num := par2;
        when 3 => iteration_num := par3;
        when 4 => iteration_num := par4;
        when 5 => iteration_num := par5;
        when 6 => iteration_num := par6;
        when 7 => iteration_num := par7;
        when 8 => iteration_num := par8;
```

```
      when 9 => iteration_num := par9;
      when 10 => iteration_num := par10;
      when 11 => iteration_num := par11;
    end case;
  end get_iteration_num;


procedure same_line(lin : in vhdl_line;
                    ptr : inout integer range vhdl_line'range;
                    file_io : in boolean) is
  variable para_num : para_num_range;
  variable label_length : integer;
  variable label_ptr : integer range label_length_type'range;
begin
  if file_io then
    file_get_integer('?',para_num);
  else
    get_integer(lin,ptr,'?',para_num);
  end if;
  get_iteration_num(para_num,label_length);
  label_ptr := label_length_type'low;
  for i in 1 to label_length loop
    write(out_file,label_length_test(label_ptr));
    if label_ptr = label_length_type'high then
      label_ptr := label_length_type'low;
    else
      label_ptr := label_ptr + 1;
    end if;
  end loop;
end same_line;
procedure process_block(iteration_num : in integer;
                        lin : inout vhdl_line;
                        char_count : inout integer range vhdl_line'range;
                        seq_num_length : in integer;
                        indent : in integer range 0 to vhdl_line'high;
                        first_iter : in boolean;
                        length : inout tog_val_length_type;
                        val : inout tog_val_type;
                        first_toggle : in boolean;
                        current_toggle : inout tog_vals;
                        tog_limit : inout tog_vals;
                        same_line_flag : out boolean) is
procedure save_sub(blk : in vhdl_line;
                   char_count : inout integer range vhdl_line'range;
                   indent : in integer range 0 to vhdl_line'high;
                   para_num_length : out integer) is
  variable ch : character;
  variable seq_num_length : integer;
  variable lin : vhdl_line;
  variable iteration_num : integer;
  variable block_count : integer;
```

```
variable blk_counter : integer;
variable cur_indent : integer range 0 to vhdl_line'high := indent;
variable first_iter : boolean := false;
variable para_num : para_num_range;
variable first_toggle : boolean;
variable length : tog_val_length_type;
variable val : tog_val_type;
variable current_toggle : tog_vals;
variable tog_limit : tog_vals;
variable same_line_flag : boolean;
begin
  blk_counter := 1;
  get_integer(blk,blk_counter,'[',para_num);
  get_seq_num_length(para_num,para_num_length);
  get_iteration_num(para_num,iteration_num);
  assert (blk(blk_counter) = '[') report "syntax error" severity error;
  blk_counter := blk_counter + 1;
  char_count := 0;
  block_count := 0;
  ch := blk(blk_counter);
  blk_counter := blk_counter + 1;
  while (ch /= ']') or (block_count /=0) loop
    char_count := char_count + 1;
    lin(char_count) := ch;
    if ch = '#' then
      block_count := block_count + 1;
    elsif ch = ']' then
      block_count := block_count - 1;
    end if;
    ch := blk(blk_counter);
    blk_counter := blk_counter + 1;
  end loop;
  if cur_indent = 0 then
    first_iter := true;
  end if;
  first_toggle := true;
  same_line_flag := false;
  for i in 1 to iteration_num loop
    get_seq_num_length(i,seq_num_length);
    process_block(i,lin,char_count,seq_num_length,cur_indent,first_iter,
                  length,val,first_toggle,current_toggle,tog_limit,
                  same_line_flag);
    first_toggle := false;
    first_iter := false;
  end loop;
end save_sub;
variable counter : integer;
variable temp_count : integer range 0 to char_count;
variable temp_block : vhdl_line;
variable cur_indent : integer range 0 to vhdl_line'high := indent;
variable para_num : para_num_range;
```

```
variable limit : integer;
variable limit_length : integer;
variable local_first_toggle : boolean := first_toggle;
variable para_num_length : integer;
begin
  counter := 0;
  temp_count := 0;
  if lin(1) = '-' then
    counter := counter + 1;
    same_line_flag := true;
  else
    same_line_flag := false;
    if not first_iter then
    - write(out_file,lf);
    end if;
    for i in 1 to indent loop
      write(out_file,' ');
    end loop;
  end if;
  while counter < char_count loop
    counter := counter + 1;
    if lin(counter) = '#' then
      if counter > 1 then
        if lin(counter - 1) /= lf then
          cur_indent := indent + counter;
        end if;
      end if;
      for i in (counter + 1) to char_count loop
        temp_block(i - counter) := lin(i);
      end loop;
      save_sub(temp_block,temp_count,cur_indent,para_num_length);
      counter := counter + 2 + temp_count + para_num_length;
    elsif lin(counter) = '@' then
      insert_num(iteration_num,seq_num_length);
    elsif lin(counter) = '%' the
      counter := counter + 1;
      get_integer(lin,counter,'%',para_num);
      get_iteration_num(para_num,limit);
      get_seq_num_length(limit,limit_length);
      insert_num(limit,limit_length);
    elsif lin(counter) = '$' then
      insert_toggle_value(lin,counter,length,val,local_first_toggle,
                          current_toggle,tog_limit);
    elsif lin(counter) = '?' then
      counter := counter + 1;
      same_line(lin,counter,false);
    else
      write(out_file,lin(counter));
    end if;
  end loop;
end process_block;
```

```
procedure save_block(char_count : inout integer range vhdl_line'range;
                     indent : in integer range 0 to vhdl_line'high) is
  variable ch : character;
  variable lin : vhdl_line;
  variable para_num : para_num_range;
  variable seq_num_length : integer;
  variable iteration_num : integer;
  variable block_count : integer;
  variable cur_indent : integer range 0 to vhdl_line'high := indent;
  variable first_iter : boolean := false;
  variable first_toggle : boolean;
  variable length : tog_val_length_type;
  variable val : tog_val_type;
  variable current_toggle : tog_vals;
  variable tog_limit : tog_vals;
  variable same_line_flag : boolean;
  begin
    file_get_integer('[',para_num);
    get_iteration_num(para_num,iteration_num);
    char_count := 0;
    block_count := 0;
    read(in_file,ch);
    while (ch /= ']') or (block_count /=0) loop
      char_count := char_count + 1;
      lin(char_count) := ch;
      if ch = '#' then
        block_count := block_count + 1;
      elsif ch = ']' then
        block_count := block_count - 1;
      end if;
      read(in_file,ch);
    end loop;
    if cur_indent = 0 then
      first_iter := true;
    end if;
    first_toggle := true;
    same_line_flag := false;
    for i in 1 to iteration_num loop
      get_seq_num_length(i,seq_num_length);
      process_block(i,lin,char_count,seq_num_length,cur_indent,first_iter,
                    length,val,first_toggle,current_toggle,tog_limit,
                    same_line_flag);
      first_toggle := false;
      first_iter := false;
    end loop;
  end save_block;


begin
```

```
process
variable indent : integer range 0 to vhdl_line'high := 0;
variable ch : character;
variable char_count : integer range vhdl_line'range;
variable para_num : para_num_range;
variable limit : integer;
variable limit_length : integer;
variable already_read : boolean := false;
variable dummy_lin : vhdl_line;
variable dummy_ptr : integer range vhdl_line'range;
begin
  if not endfile(in_file) then
    if not already_read then
    - read(in_file,ch);
    end if;
    already_read := false;
    if ch = '-' then
      read(in_file,ch);
      if ch = '-' then
        while ch /= lf loop
          read(in_file,ch);
        end loop;
        if indent /= 0 then
          write(out_file,lf);
        end if;
        indent := 0;
      else
        write(out_file,'-');
        indent := indent + 1;
        already_read := true;
      end if;
    elsif ch = '#' then
      save_block(char_count,indent);
      indent := indent + char_count;
    elsif ch = '%' then
      file_get_integer('%',para_num);
      get_iteration_num(para_num,limit);
      get_seq_num_length(limit,limit_length);
      insert_num(limit,limit_length);
      indent := indent + limit_length;
    elsif ch = '?' then
      same_line(dummy_lin,dummy_ptr,true);
    else
      write(out_file,ch);
      indent := indent + 1;
      if ch = lf then
        indent := 0;
      end if;
    end if;
  assert not endfile(in_file) report "done" severity failure;
  end if;
```

```
  end process;
end gen;
```

"help.txt" :

Benchmark Executors :

Before starting to run any benchmarks, you need to read this note, along with "unixinfo.txt" in the "readme" directory.

Make sure you "copy" (VMS) or "tar" (UNIX) the subdirectories "bench" and "coms" from your top-level or home directory, so the command files will work properly. The subdirectories "bench" and "coms" do not have to exist before you enter the command if you are using UNIX; "tar" will create it.

The general format for running a VMS batch job is :

```
$ submit command_file_name.com
```

The "/que=que_name" and "/log=log_file_name" qualifiers may be used to control which batch queue the job is submitted to, and the pathname of the log_file. See "unixinfo.txt" for a description of the UNIX equivalent.

The "bench" subdirectory contains the VHDL benchmark "shells". Also in this directory is the utility that takes the shells and your input parameters and generates proper VHDL source code to run for timing data collection. The program that generates VHDL source from shells is "gen.vhd". It is written in VHDL and the command files to analyze, model generate, and build it are "generate.com" and "generate.unix" for VMS and UNIX, respectively, in the "coms" subdirectory.

The "coms" subdirectory contains the same directory structure as "bench". For each "bench" directory with a shell file, there are command files (VMS - test.com and UNIX - test.unix) in "coms" to analyze, model generate, build, and simulate the benchmark. These command files were designed for numerous consecutive analyze/model generate/build/sim cycles for the same model, so after the "sim" command, the run, kernel, entity, and architecture files are deleted from the VLS library. If this is not desired, the command files must be edited to remove these "vls del" commands and to add the "/replace" (VMS) or "-replace" (UNIX) option to the "build" command.

The "readme" subdirectory contains documentation. File "test.edt" has the TEST NUMBER, PATHNAME, and PURPOSE (description) of each test. The tests are organized into subdirectories by the VHDL features they test. Each feature category has its own subdirectory. File "matrices.edt" lists all the feature categories in column format, along with each test number in row format. The categories tested in each benchmark are then "X"-ed off in the appropriate column and rows. It would probably be helpful to print both "test.edt" and "matrices.edt". File matrices_132col.edt is the 132-column version of matrices.edt, which is 80 columns wide.

When you are ready to start running the benchmarks, first analyze, model generate, and build "gen", using the command file "generate.com" (VMS), or

"generate.unix" (UNIX). Then choose a test, change directories (set def for VMS), (cd for UNIX) to that subdirectory, and print the shell either to the screen or a printer, because you need to read the comments to see what the parameters are. Look at the EXAMPLE section, and use the exact same file names in your "sim gen..." command. After you have decided what parameter values to use, enter the "sim gen..." command. You will then have a (hopefully) syntactically correct VHDL model named "test.vhd', or something very similar. When "gen" has successfully completed generating a description, you will get an assertion violation, along with the message "done". You then can use the "test.com" (VMS) or "test.unix" (UNIX) command file in the corresponding subdirectory of "coms" to run the job in the non-interactive mode. For UNIX users, see "unixinfo.txt" to set up your .cshrc file to collect timing data, if you want it. Your command to start the job will be in the format

```
% csh test.unix >& log_file_name &
```

This will cause all messages related to this job to be written to the file "log_file_name" (you can name this whatever you want). It also runs the job in the background, instead of in interactive mode. When your job is reported "done", check "log_file_name" to see the timing data or any errors that might have occurred. You can also "more" or "cat" this file while the job is running to check which command is executing. "unixinfo.txt" explains how to interpret the timing data that will be in "log_file_name". You can then re-run the test, varying the parameters.

*** NOTE ***
****
In order for the command files to work, you must either have your default directory set to where the shell resides when executing the "sim gen..." command OR specify the complete pathname for the output file in the "sim gen..." command so that the output file ends up in the same directory as its shell. If this is not desirable, the command files must be edited to remove the "set def" (VMS) or "cd" (UNIX) lines.
****
****
Users with toolsets that do not support generic parameters in top-level entities must use front_end.vhd and alternate_gen.vhd instead of gen.vhd. Instead of the "sim gen/param=..." command, you must edit front_end.vhd and set the constants ifile, ofile, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, and p11 to the appropriate parameter values. "ifile" corresponds to the input file name (the shell file name), "ofile" corresponds to the output file name (usually "test.vhd"), and p1 through p11 correspond to parameters 1 through 11 as defined in the shell file comments. You must count the number of characters in your input and output files and set the string lengths for ifile_name and ofile_name accordingly. Then you must edit alternate_gen.vhd and set the string lengths for the input and output files to the same values as in front_end.vhd. After editing, you need to analyze and model generate alternate_gen.vhd (entity : gen; architecture : gen), then analyze, model generate, build, and simulate front_end.vhd (entity : front_end; architecture : front_end). Simulation of front_end causes the VHDL source file to be generated. Each time new

parameter values are desired, the edit, analyze, model generate, build, sim cycle must be done. A command file (front_end.com (VMS) or front_end.unix (UNIX)) resides in [.coms] to do all but the editing.
****

For VMS users, the timing collection commands are already in the command files; however, these commands do not provide timing data for any subprocesses spawned (as in model generation, build, and simulation). To get this data, you need to check with your system manager to see what kind of system accounting, if any, is being done. If "process" accounting is done, ask the system manager to execute a command in the following general form AFTER you have run some benchmarks

```
$ account/binary/output=DATA_FILE.DAT/since=DD-MM-YYYY/user=YOUR_USER_ID
```

where DATA_FILE.DAT is any name you choose for the output file, DD-MM-YYYY is the date you started running the benchmarks (check with the manager to see how far back the accounting files go), and YOUR_USER_ID is your login name on the system. When you receive the DATA_FILE.DAT file, use the following command to look at it

```
$ acc/full DATA_FILE.DAT
```

This displays one subprocess's data per screen. You can use the date/time data in the log_file to help identify which subprocess corresponds to which command. Only model generate, build, and sim commands spawn subprocess, so nothing in the accounting file will correspond to the analysis commands. When you have matched a command with a subprocess, record the "CPU time" data for the command. When you have done this for each command in a log_file, go through the log_file and subtract the "Elapsed CPU Time" number above each command from the one below it to get the remaining CPU time. Add this time to the subprocess CPU time from the accounting file. This is the total CPU time for the command. I am not familiar with the other types of accounting, although I know that "image accounting" can also be used.

If you have any questions or problems understanding the system, or if you encounter errors, please phone me :

Capt Karen Serafino
(513)255-8635

"unixinfo.txt" :

In order to collect timing data while running UNIX batch jobs, add the following command to your .cshrc file :

```
set time = (0 "u=%U s=%S r=%E C=%P P=%C s=%W m=%X d=%D T=%K m=%M pi=%F p=%R Bi=
%I Bo=%O")
```

This will cause a timing line to be output for each command. Commands that

spawn subprocesses will have multiple timing lines. The last line is a total of the preceding ones for such commands.

Abbreviation meanings are as follows : (found in UNIX Programmer's Manual)

%U : amount of time spent executing in user mode

%S : amount of time spent in the system executing in behalf of the process (in seconds)

%E : real (elapsed) time in minutes and seconds

%P : percentage of CPU utilization (ratio of user plus system times to real time, scaled by the number of processors on the system)

%C : CPU parallelization factor or concurrency level (somewhere between one and the number of processors on the system); this value is designed to be used when timing a particular process and so will display N/A when "time" is issued as a command with no arguments

%W : number of process swaps out of main memory

%X : amount of memory shared among other processes (in kilobytes)

%D : combined size of unshared data and stack segments (in kilobytes)

%K : total size of shared memory, unshared data, and unshared stack sizes

%M : maximum resident set size utilized (in kilobytes)

%F : number of page faults serviced which required I/O activity

%R : number of page faults serviced without I/O activity; here, I/O activity is avoided by reclaiming a page from the list of pages awaiting reallocation

%I : number of times the file system had to perform block input

%O : number of times the file system had to perform block output

This command can be tailored to your interests by including only those abbreviations that record the data you want and omitting the rest. The indicators preceding the equal signs and the equal signs serve as labels and are printed to standard output, followed by the actual corresponding timing data. You can change the labels to suit your own formatting style.

The command format to run the batch jobs in the background and record the commands and timing data in a file is :

```
% csh command_file_name >& log_file_name &
```


☆U.S.Government Printing Office. 1991—548-076/44093