AFRL-IF-WP-TR-1998-1507

VHDL MODELING AND BENCHMARKS

MISSISSIPPI STATE UNIVERSITY
MSU ENGINEERING RESEARCH CENTER
P.O. BOX 9627
MISSISSIPPI STATE, MS 39759

MARCH 1998

FINAL REPORT FOR 04/11/1994 - 12/31/1997

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

Information Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson Air Force Base, OH 45433-7334

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MAR 1998	FINAL 04/11/1994--12/31/1997

4. TITLE AND SUBTITLE VHDL MODELING AND BENCHMARKS

5. FUNDING NUMBERS
C F33615-94-C-1494
PE 63739
PR A268
TA 02
WU 15

6. AUTHOR(S)

Robert B. Reese

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MISSISSIPPI STATE UNIVERSITY
MSU ENGINEERING RESEARCH CENTER
P.O. BOX 9627
MISSISSIPPI STATE, MS 39759

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7334
POC: KERRY HILL, AFRL/IFTA (937) 255-7698 ext. 3604

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-WP-TR-1998-1507

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The MPL developed VHDL models for several commercial SRAMS, PROMS, and PLDs. A VHDL component library for use in simulating an X4000 FPGA implementation was created; a netlist conversion tool is used for converting from a Xilinx LCA netlist format to a VHDL structural model. A SRAM VHDL model generator was created with an associated web–browser interface that allowed SRAM VHDL models to be generated via the WWW. A VHDL intelligent tutor demonstration was done using Java and ActiveX capabilities coupled to the Microsoft Internet explorer Web Browser. The demonstration utilized materials from the SCRA VHDL Interactive Tutorial CDROM. The MPL developed a synthesizeable 1750A VHDL model based on the Fairchild F9450 implementation. The 1750A model included all 1750A instructions except for floating point, optional IO operations, and most of the console mode operations. The model was synthesized to both a standard cell netlist and a Xilinx X4000 netlist. All of the models are available for download at the URL http://www.erc.msstate.edu/mpl (follow the *distributions* link).

14. SUBJECT TERMS

15. NUMBER OF PAGES 24

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Table Of Contents

List of Illustrations/Tables

1. SUMMARY OF WORK PERFORMED

The MPL developed VHDL models for several commercial SRAMS, PROMS, and PLDs. A VHDL component library for use in simulating an X4000 FPGA implementation was created; a netlist conversion tool is used for converting from a Xilinx LCA netlist format to a VHDL structural model. A SRAM VHDL model generator was created with an associated web–browser interface that allowed SRAM VHDL models to be generated via the WWW. A VHDL intelligent tutor demonstration was done via Java and ActiveX capabilities coupled to the Microsoft Internet explorer Web Browser. The demonstration utilized materials from the SCRA VHDL Interactive Tutorial CDROM. The MPL developed a synthesizeable 1750A VHDL model based on the Fairchild F9450 implementation. The 1750A model included all 1750A instructions except for floating point, optional IO operations, and most of the console mode operations. The model was synthesized to both a standard cell netlist and a Xilinx X4000 netlist. All of the models are available for download at the URL http://www.erc.msstate.edu/mpl (follow the 'distributions link).

2. VHDL MODEL DEVELOPMENT

2.1 RAM, PROM, PLD Models

One of the goals of this contract was to provide example VHDL models for common COTS devices; specifically RAMS, ROMS and simple PLDS. The VHDL model list produced under this contract for these device types is shown below:

cy7b134 – 4K x 8 Dual Port SRAM w/ OE, CE, R/W, non–separate I/O, and semaphores

cy7b138 – 4K x 8 Dual Port SRAM w/ OE, CE, R/W, non–separate I/O, Int, Busy, and Semaphores

cy7b195 – 64K x 4 BiCMOS SRAM w/ CE, OE, WE, and non–separate I/O

cy7b199 – 32K x 8 BiCMOS SRAM w/ CE, OE, WE, and non–separate I/O

cy7c1006 – 256K x 4 SRAM w/ CE, OE, WE, and non–sparate I/O

cy7c1007 – 1M x 1 SRAM w/ CE, WE, and separate I/O

cy7c195 – 64K x 4 SRAM w/ CE, OE, WE, and non–separate I/O

cy7c199 – 32K x 8 SRAM w/ CE, WE, OE, and non–separate I/O

cy7c256 – 32K x 8 PROM w/ CE and OE, Switched and Reprogrammable

cy7c266 – 8K x 8 PROM w/ OE and CE, Switched and Programmable

cy7c276 – 16K x 16 PROM w/ CS0, CS1, CS2, and OE; Reprogrammable

cy7c285 – 64K x 8 PROM w/ CS; Reprogrammable Fast Column Access

cy7c401 – 64 x 4 FIFO w/ MR, OE, SI, and SO; Cascadable

cy7c402 – 64 x 5 FIFO w/ MR, OE, SI, and SO; Cascadable

cy7c403 – 64 x 4 FIFO w/ MR, OE, SI, and SO; Cascadable

cy7c404 – 64 x 5 FIFO w/ MR, OE, SI, and SO; Cascadable

cy7c429 – 2K x 9 FIFO w/ MR, W, R, XI, and FL/RT; Cascadable

pal16l8 – Industry Standard PLD

pal16r4 – Industry Standard PLD

pal16r6 – Industry Standard PLD

pal16r8 – Industry Standard PLD

palc16l8 – Industry Standard PLD

palc16r4 – Industry Standard PLD

palc16r6 – Industry Standard PLD

palc16r8 – Industry Standard PLD

palc22v10d – Industry Standard PLD

dtXXfct543 – Octal Latched Transceiver w/ Latch Enable, Chip Enable, and Output Enable

idtXXfct646 – Octal Transceiver/Register w/ Enable and Direction Control

idtXXfct841 – Bus Interface Latches w/ PRE, CLR, LE, and OELibrary Xilinx

Four files were associated with each model:

- *modelname*_.vhd : Entity definition
- *modelname*_behavioral.vhd : Architecture definition
- *modelname*_tv_.vhd : Timing view package declaration
- *modelname*_tv.vhd : Timing view package body

The databook timing data was arranged in a two dimensional array where one axis was operating point (minimum, nominal, maximum) and the other axis was speed grade (mil_spec or commercial). Generics were defined in each model entity so that individual timing package values could be overridden by the user.

Several VHDL packages were written to support these models. These packages were used to provide functionality shared by similar models. Examples of shared functionality JEDEC file parsing used by the PLD models and memory allocation/initialization/dumping used by the RAM/PROM models.

2.2 Xilinx X4000 FPGA Model

One of the goals of this contract was to provide an example VHDL model for a complex programmable device such as an FPGA. We chose the Xilinx X4000 family since it was one of the more popular FPGA families and the Lockheed–Sanders RASSP team was using the X4000 for the RASSP demo development.

The Xilinx X4000 family is a static RAM based FPGA. The basic logic cell is called a Configurable Logic Block (CLB) and contains two 4–input lookup tables, two D flip–flops, and dedicated carry logic. The lookup tables can implement two separate 4–variable functions; the output of the tables can also be combined to form a third logic function. The lookup tables can also be used as an asynchronous SRAM, synchronous SRAM (X4000E) or dual port SRAM (X4000E). Combinational outputs can be registered via the flip–flops if desired. IO is handled via a versatile Input Output Block (IOB) which has several configuration options such as registered/non–registered on the input or output signals, tri–state output, programmable output slew rate, pullup/pulldown on output, and output polarity. There are several other logic resources on the chip as well – an on–chip oscillator, fast decoders, tri–state buffers, pullups, high–drive buffers, startup logic, and boundary scan capability.

At the outset we decided that the X4000 model would be a static model; i.e., the model behavior would not be dynamically changeable during simulation even though the X4000 FPGA supports dynamic reconfiguration. Dynamic reconfiguration is rarely used by designers and including this feature would have significantly impacted model performance. This decision allowed the modeling methodology to follow the traditional path of generating a structural model of primitive components based on the initial programming of the device. A Logic Cell Array (LCA) file is generated as part of the normal Xilinx FPGA mapping procedure; this file is a netlist of the actual on–chip logic resources (CLBs, IOBs, fast decoders, on–chip oscillators, etc.). The LCA

file was used as the input for the VHDL model generator script which produced a structural VHDL description of the netlist.

A X4000 VHDL component library was created to model the onchip resources. The consists of:

- 4000clb the configurable logic block
- x4000iob the input/output block
- x4000bufgp primary global buffer to implement high drive nets
- x4000bufgs secondary global buffer to implement high drive nets
- x4000tbuf tristate buffer
- x4000startup component for startup sequence emulation
- x4000osc internal oscillator — 8MHz, 500kHz, 16kHz, 490Hz available

There were some additional logic resources (wide decoders, pullups, pulldowns) which are implemented directly in the structural model without reference to a component model. The CLB and IOB components are further subdivided into functional blocks which implement the lookup table/SRAM functionality, D flip flops, dedicated carry logic, and IOB latching capability. VHDL *generate* blocks are used to implement only the logic which is actually required inside of a CLB or IOB (i.e., if a CLB does not make use of the D flip flops, then the D flip–flop components are not instantiated).

A Perl5 script called *lca2vhd* is used to convert an LCA file to a VHDL structural model. In addition to reading the LCA file, the script also reads package and pin files within the Xilinx software distribution to determine various features which are package and pin dependent. A design specific timing file ('design–name'.spc) will be read if present; this file can be optionally generated during the FPGA mapping process and contains package and speed grade specific tim-

ing information. It should be noted that the LCA file can contain back–annotated net delays; these net delays are included in the VHDL structural model. A WWW interface was also created for the l*ca2vhd* script for purposes of remote execution. The user fills out a form selecting various *lca2vhd* options, uploads their LCA file, and triggers execution of *lca2vhd*. The user then receives a compressed tar file which contains the *lca2vhd* log file and the generated VHDL model.

2.3 SRAM Model Generator

Using experience gained by writing the standalone SRAM/PROM models, an SRAM VHDL model was created. This model generator is accessed via a WWW–interface which allows the user to enter both architectural and timing parameters. The architectural parameters allow the user to specify:

- address and data bus width
- separate or shared data I/O
- Chip select and OE logic functions

The timing parameters are industry–standard SRAM timing parameters such as data valid from OE valid, data valid from address valid, etc. The WWW interface provided sample parameter sets for several industry standard RAMS.

2.4 1750A Synthesizeable VHDL Model

In the last 4 months of the contract, a 1750A Synthesizeable VHDL model based upon the Fairchild F9450 implementation was created. The Mil Std 1750A document defines a 16–bit, CISC style instruction set architecture. The instruction set includes a full set of integer, floating point, and IO instructions with a large set of addressing modes. The MPL model implements all of the Fairchild F9450 functionality except for floating point, optional IO instructions, and most of the console mode operations.

The model was a large effort; the model is currently at approximately 100,000 lines of code. Figure 1 shows the VHDL model hierarchy for the 1750A model.

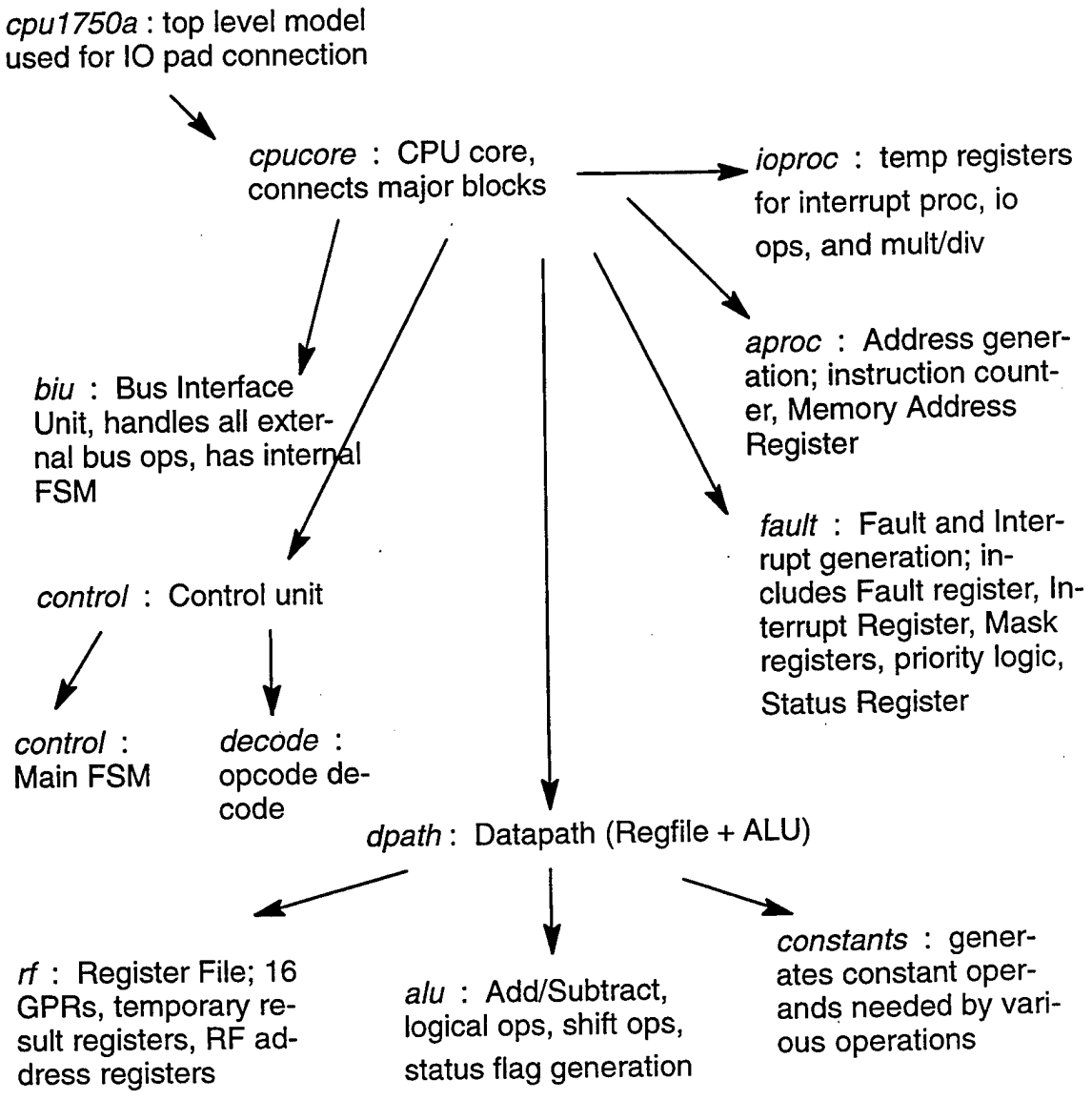

Figure 1: 1750A Model Hierarchy

The model was verified using some legacy 1750A tests provided by K. Hill of WPAFB. A Perl–based regression system was created that:

- Would take the original ASM test files and convert the source to be compatible with the Unix–based 1750a simulator that was used for verification
- Run the 1750a simulator to produce a golden result
- Run either the gate–level or behavioral VHDL model and produce a test result
- Compare the two results for pass/fail. Test logs were produced that showed the exact problem in case of a failure

A total of 272 tests were provided (excluding floating point tests). Of these, 220 tests passed, while 48 tests were not used because either the test was different enough from the others that the automated conversion did not work or our simulator did not support executing this code. An example of the former are all the jump tests; we tested these with some local tests instead. An example of the latter was the extended address space tests; our UNIX 1750A simulator was not set up for MMU operations. Four tests produced incorrect results in the UNIX 1750A simulator (the Carry bit was set for some addressing modes in the logical operations which is incorrect) but correct results in the VHDL simulation. The tests that passed were simulated using both the behavioral model and gate–level models.

The model was synthesized to a standard cell library and the Xilinx X4000 FPGA in order to demonstrate synthesis compatibility. Because of time constraints, no attempt was made to tune the synthesis process to make use of special hardware features of the target technology that would have lowered the gate count. The final Xilinx CLB count was 5500 CLBS and the standard cell count was 7200. Both of these counts are higher than they should be and could easily be reduced via extra effort at the hardware mapping level. For example, the core of the register

file took 1150 CLBs; if the SRAM capabilities of the Xilinx CLBs are used this could be reduced to less than 100 CLBs. The standard cell model was used in the gate level verification tests.

3. INTELLIGENT TUTORING DEMONSTRATION

A project effort directed by Scott Calhoun of Web Services, Inc. concerned an Internet–based Intelligent VHDL Tutoring demonstration. Scott Calhoun had been the PI on this contract until December 1996, at which time he formed Web Services, Inc. and Robert Reese (MSU MPL) assumed PI responsibilities. The work for the tutoring demonstration was performed by Web Services for MPL during the period of January 1997 to July 1997.

The Intelligent VHDL Tutoring demonstration was developed to accompany the training materials on the RASSP E&F VHDL CD–ROM created by South Carolina Research Authority (SCRA). The software platform developed for this effort is called Internet–based Intelligent Tutoring for Tools (I2T2).The I2T2 platform consists of a Java server and several Java services that enable portability and wide–area delivery of instructional material. A web browser is needed on the client side to drive the Tutor Console.The I2T2 architecture supports several I2T2 services available which may be distributed or local to the server. The Accounts Database service provides a means for maintaining student records and performance levels. The Test/Evaluation Service provides a means for testing and evaluating students on the different knowledge areas covered by a tutor. Test mechanisms include the common multiple choice, fill–in–the–blank, and matching test questions in addition to application sequence comparisons and evaluations. The Multi–Media Resource (MMR) Service provides several different mediums of communication with the student including animated application sequences, pop–up messages, graphics displays, and audio streams. Any registered Windows 95 application may be monitored and animated unobtrusively by the ActiveX I2T2 Application Player–Recorder (APR) that is available through our Multi–Media Service.

Tut is a quasi object–oriented scripting language used for constructing I2T2 tutors. Close comparisons can be drawn between the syntax of this language and those of C and Java. Although this language was developed to allow for the construction of intelligent tutors, it was designed

so that eventually an authoring environment for intelligent tutors can generate Tut scripts. The Tut script is instantiated by a Java applet call from HTML, and there are no limitations on the size or complexity of the script.

The I2T2 demonstration is available on the CDROM accompanying this final report. In order to effectively run these tutorials, one will need to download and install PeakVHDL from Accolade's web site (http://www.acc–eda.com/demo.htm). Tut is not dependent upon a specific web browser, but its Java and ActiveX presentations are directly influenced by the capabilities of the browser. Windows Explorer 3.0 or later is recommended for Tutors that use the APR. Otherwise, any Java–enabled web browser is compatible. A personal web server will also need to be running for stand–alone tutors like the ones we created for VHDL.

The SCRA VHDL Interactive Tutorial CD–ROM contains 3 modules: Basic VHDL, Structural VHDL, and Behavioral VHDL. When possible, the I2T2 environment was incorporated into the already–existing HTML tutorials by inserting Java applet calls which initiate Tut scripts customized for each lesson (Figure 2). This method of integration demonstrates the usefulness of the I2T2 platform as an upgrade to pre–existing HTML material.

The first I2T2 tutor was tightly incorporated into the existing Basic VHDL Tutorial. This module introduces the student to VHDL and provides a familiarity with its evolution and history. While the tutorial presents information, the I2T2 APR service is used to illustrate some of the examples in the PeakVHDL Analyzer and Simulator. This tutor also evaluates the student's understanding of the module by posing exercises and questions for the student. The tutor determines when the student needs to review a concept by evaluating the student's test results. In this tutor, the review is conducted by forcing the student to repeat the HTML lesson pages.

The second I2T2 tutor focuses on the Accolade tools used for analysis and simulation. The Accolade PeakVHDL Analyzer and Simulator have been chosen as the tools to be used because they

are free to the students. This tutor will describe and demonstrate the features of PeakVHDL to the student. The tutor guides the student through several typical analysis/simulation scenarios offering explanations of common problems, misunderstandings, and pitfalls. The student's understanding of the use of PeakVHDL will be automatically evaluated using I2T2 Test/Eval Service mechanisms. Although this tutor will still need to be initiated by the web browser, portions of the material will be covered in completely separate consoles. This method of presentation demonstrates the I2T2 platform in a virtually stand–alone technique.The third I2T2 tutor was incorporated into the Behavioral VHDL module. The tutor helps to explain concepts taught in this module by providing audio, demonstrations and slide presentation sequences to further illustrate the theories. This tutor is also able to evaluate the student's performance based upon tests where needed.

A demonstration of the tutoring system was given by Scott Calhoun at the final RASSP PI meeting held on October 23rd, 1997 in Arlington, VA.

4. VHDL MODEL DISTRIBUTION / USER FEEDBACK.

A WWW distribution site has been maintained by MPL since January 1996. The URL for the site is http://www.erc.msstate.edu/mpl (follow the DISTRIBUTIONS link). The site has been very active; the site has averaged approximately 1600 file downloads from 250 unique sites each month. The WWW Xilinx FPGA model generator and SRAM generator are also accessible from this site. The most popular models in terms of downloads each month have been the SRAM and PLD models. One puzzling aspect is that despite averaging such high monthly activity we have received very little feedback on the models themselves (the download site has an easily accessible email comment point). We feel that one reason we are not getting much feedback is because users are taking the model sources and modifying it to suit their particular needs.

5. PAPERS

1. McCloskey,C., V. Sanders, R. Reese, "Redesign of a Generic VHDL Model Template for SRAM", Rapid Systems Prototyping with VHDL, VIUF Fall 1997 Conference Proceedings,Washington D.C, pp 122–124.
2. Brown,D. R. Reese, "VHDL Modeling and Tutoring Efforts", Rapid Systems Prototyping with VHDL, VIUF Fall 1997 Conference Proceedings,Washington D.C, pp 179–182.
3. Calhoun J.S., V. K. Madisetti, R. B. Reese, T. Egolf., "Developing and Distributing Component–Level VHDL Models", Journal of VLSI Signal Processing, Vol 15 (1996), pp 111–126.
4. Reese, R., Vince Sanders., "A VHDL Modeling Approach to the Xilinx 4000 Series FPGA", VHDL International User's Forum, October 27–30, 1996, Durham, NC.
5. Reese, R. and J. Scott Calhoun " Mississippi State Develops on–Line FPGA VHDL Model Generator", RASSP Digest, Vol 3., September 1996, pp 39–41.
6. Reese, R. and J. Scott Calhoun " VHDL Component Modeling: Impact on the RASSP Program", RASSP Digest, Vol 2, No 1., 1st Qtr 1995, pp 18–19.

6. APPENDIX

6.1 SEAFAC Test Status for 1750A VHDL Model

Tests marked as F–CNC indicate that the test was not used by the regression test system because it could not be automatically converted. Tests marked as F–SD indicate that the VHDL results were correct; the Unix 1750A simulator results were incorrect.

absqc110 – Passed: single precision absolute value instruction
absqc111 – Passed: single precision absolute value instruction using the same register for both source and destination
abxq7190 – Passed: single precision integer add, base relative indexed mode
aimq7140 – Passed: single precision add instruction, immediate long mode
aisp7150 – Passed: single precision integer "immediate short add" instruction
andbb180 – Passed: "logical and" instruction, base relative mode
andmb140 – Passed: logical and instruction, immediate long mode
andqb120 – Passed: "logical and" instruction
andrb110 – Passed: "logical and" instruction
andrb111 – Passed: logical and" instruction
andxb120 – Passed: logical and" instruction
andxb190 – F–SD : logical and instruction, base relative indexed mode
aqqq7120 – Passed: single precision add instruction,memory direct mode
arqq7110 – Passed: single precision integer add
arqq7111 – Passed: single precision integer add
axqq7120 – Passed: single precision add instruction, memory direct–indexed mode
bexqh1a0 – F–CNC : Branch to Executive instruction
bezq4170 – Passed: branch if equal to (zero), IC relative mode
bgeq4170 – Passed: branch greater than or equal to (zero), IC relative mode
bgtq4170 – Passed: branch if greater than (zero), IC relative mode
bleq4170 – Passed: branch if less than or equal to (zero),IC relative mode
bltq4170 – Passed: branch if less than (zero), IC relative mode
bnzq4170 – Passed: branch if not equal to (zero), IC relative mode
brqq4170 – Passed: branch unconditionally, IC relative mode
catch1a0 – F–CNC : attempts to test whether there is interference between instruction fetching and data addressing
cblqd120 – Passed: exercise the "COMPARE BETWEEN LIMITS – MEMORY DIRECT MODE" instruction
cblxd120 – Passed: exercise the "compare between limits – memory direct, indexed mode
cbqqd180 – Passed: single precision compare, base relative mode
cbxqd190 – Passed: single precision compare, base relative indexed mode
cimqd140 – Passed: single precision compare – immediate long mode
cisnd160 – Passed: single precision integer "immediate short negative compare
cispd150 – Passed: single precision integer "immediate short positive compare
clir1140 – F–CNC : Clear Interrupt Request XIO
clkqh1a0 – F–CNC : clocks are within a reasonable deviation of the expected ratio of 10:1 for timers A and B

cqqqd120 – Passed: single precision compare – memory direct mode" instruction
crqqd110 – Passed: single precision compare – register mode" instruction
crqqd111 – Passed: single precision compare – register mode" instruction
cxqqd120 – Passed: single precision compare – memory direct, indexed mode
dabsc210 – Passed: double precision absolute value instruction
dabsc211 – Passed: double precision absolute value instruction using the same registers for both source and destination
daqq7220 – Passed: double precision add instruction memory direct mode
darq7210 – Passed: double precision integer add" instruction
darq7211 – Passed: double precision integer add" instruction
daxq7220 – Passed: double precision add instruction memory direct–indexed mode
dbqqa180 – Passed: single precision integer divide base relative mode
dbxqa190 – Passed: single precision integer divide base relative indexed mode with 32–bit dividend
dcqqd220 – Passed: double precision compare" instruction
dcrqd210 – Passed: double precision compare" instruction
dcrqd211 – Passed: double precision compare" instruction
dcxqd220 – Passed: double precision compare instruction memory direct–indexed mode
ddqqa220 – Passed: double precision integer divide" instruction
ddrqa210 – Passed: double precision integer divide
ddrqa211 – Passed: double precision integer divide" instruction
ddxqa220 – Passed: double precision divide instruction memory direct–indexed mode
decm8120 – Passed: decrement memory by a positive integer direct mode instruction
decm8121 – Passed: decrement memory by a positive integer direct indexed mode instruction
dimqa140 – Passed: single precision divide instruction, immediate long mode with 32–bit dividend
disna160 – Passed: single precision integer "immediate short negative divide" with 16–bit product instruction
dispa150 – Passed: single precision integer immediate short positive deivide" with 16–bit result
dlbq5280 – Passed: double precision load instruction base relative mode
dlbx5290 – F–SD : double precision load instruction base relative mode with indexing
dliq5230 – Passed: double precision load – memory indirect
dlix5230 – Passed: double precision load – memory indirect indexed
dlqq5220 – Passed: double precision load – memory direct
dlrq5210 – Passed: double precision load – register mode
dlrq5211 – Passed: double precision load – register mode
dlxq5220 – Passed: double precision load – memory direct indexed
dmqq9220 – Passed: double precision integer multiply
dmrq9210 – Passed: double precision integer multiply" instruction
dmrq9211 – Passed: double precision integer multiply" instruction
dmxq9220 – Passed: double precision multiply instruction memory direct–indexed mode
dnegc210 – Passed: double precision negate register
dnegc211 – Passed: double precision negate register
dqqqa120 – Passed: single precision integer divide w/ 32–bit dividend
drqqa110 – Passed: single precision integer divide instruction with 32–bit dividend

drqqa111 – Passed: single precision integer divide instruction with 32–bit dividend
dsar3210 – Passed: double shift arithmetic – count in register
dsar3211 – Passed: shift arithmetic – count in register
dscr3210 – Passed: double shift cyclic – count in register
dscr3211 – Passed: shift cyclic – count in register instruction
dslc3210 – Passed: double shift left cyclic instruction, register mode
dsll3210 – Passed: double shift left logical instruction register mode
dslr3210 – Passed: double shift logical – count in register
dslr3211 – Passed: shift logical – count in register instruction
dsqq8220 – Passed: double precision integer subtract direct non–indexed" instruction
dsra3210 – Passed: double shift right arithmetic instruction
dsrl3210 – Passed: double shift right logical instruction register mode
dsrq8210 – Passed: double precision integer subtract register mode
dsrq8211 – Passed: double precision integer subtract register mode
dstb6280 – Passed: double precision store – based
dsti6230 – Passed: double precision store – memory indirect
dsti6231 – Passed: double precision store – memory indirect indexed
dstq6220 – Passed: double precision store – direct
dstx6220 – Passed: double precision store – direct indexed
dstx6290 – Passed: double precision store – based indexed
dsxq8220 – Passed: double precision integer subtract
dvima140 – Passed: single precision divide instruction, immediate long mode with 16–bit dividend
dvqqa120 – Passed: single precision integer divide w/ 16–bit dividend
dvrqa110 – Passed: single precision integer divide w/ 16–bit dividend" instruction
dvrqa111 – Passed: single precision integer divide w/ 16–bit dividend
dvxqa120 – Passed: single precision divide instruction memory direct–indexed mode with 16–bit dividend
dxqqa120 – Passed: single precision divide instruction memory direct–indexed mode with 32–bit dividend
getptx – F–CNC :
getput – F–CNC :
grtqh1a0 – F–CNC : The purpose of this test is to verify the operation of the general registers
iloph1a0 – F–CNC :
incm7120 – Passed: increment memory by a positive integer direct mode
incm7121 – Passed: increment memory by a positive integer direct indexed mode instruction
indx11a0 – F–CNC : indexed access to memory is done
intr15a0 – F–CNC : interrupt structure of a MS1750 machine
iopr1140 – F–CNC : test the XIO commands RIPR, ROPR, WIPR, WOPR
ixio1140 – F–CNC : Reserved and unused Spare XIOs return a Machine Error when executed
jciq4130 – F–CNC : jump on condition instruction, memory indirect mode without indexing
jcix4130 – F–CNC : jump on condition instruction, memory direct mode with indexing
jcqq4120 – F–CNC : jump on condition instruction, memory direct mode without indexing
jcxq4120 – F–CNC : jump on condition instruction, memory direct mode with indexing
jsqq4120 – F–CNC : jump to subroutine instruction, memory direct mode without indexing

jsxq4120 – F–CNC : jump to subroutine instruction, memory direct mode with indexing
jsxq4121 – F–CNC : jump to subroutine instruction, memory direct mode with indexing
lbqq5180 – Passed: single precision load instruction base relative mode
lbxq5190 – F–SD : single precision load instruction base relative mode with indexing
limq5140 – Passed: single precision load – immediate instruction
limx5140 – Passed: single precision load – immediate indexed
liqq5130 – Passed: single precision load – memory indirect
lisn5160 – Passed: single precision load – immediate short negative
lisp5150 – Passed: single precision load – immediate short positive
lixq5130 – Passed: single precision load – memory indirect indexed
llbi5130 – Passed: load from lower byte – memory indirect
llbi5131 – Passed: load from lower byte – memory indirect indexed
llbq5120 – Passed: load from lower byte – memory direct
llbx5120 – Passed: load from lower byte – memory direct indexed
lmqq5120 – F–CNC : load multiple registers – memory direct
lmxq5120 – F–CNC : load multiple registers – memory direct indexed
lqqq5120 – Passed: single precision load – memory direct
lrqq5110 – Passed: single precision load – register mode
lrqq5111 – Passed: single precision load – register mode
lsti5130 – F–CNC : load status – memory indirect
lsti5131 – F–CNC : load status – memory indirect
lstq5120 – F–CNC : load status – memory direct
lstx5120 – F–CNC : load status – memory direct
lubi5130 – Passed: load from upper byte – memory indirect
lubi5131 – Passed: load from upper byte – memory indirect indexed
lubq5120 – Passed: load from upper byte – memory direct
lubx5120 – Passed: load from lower byte – memory direct indexed
lxqq5120 – Passed: single precision load – memory direct indexed
mbqq9180 – Passed: single precision integer multiply base relative mode
mbxq9190 – Passed: single precision integer multiply base relative indexed mode
mimq9140 – Passed: single precision multiply instruction, immediate long mode.
misn9160 – Passed: single precision integer "immediate short negative multiply" instruction
misp9150 – Passed: single precision integer "immediate short positive multiply" with 16–bit product instruction
mpra1140 – F–CNC : correct operation of the XIO commands MPEN, LMP and RMP
mqqq9120 – Passed: single precision integer multiply direct non–indexed
mrqq9110 – Passed: single precision integer multiply" instruction with 32–bit result
mrqq9111 – Passed: single precision integer multiply" instruction with 32–bit result using the same register
msim9140 – Passed: single precision multiply instruction, immediate long mode with 16–bit product
msqq9120 – Passed: single precision integer multiply" instruction with 16–bit result
msrq9110 – Passed: single precision integer multiply" instruction with 16–bit result
msrq9111 – Passed: single precision integer multiply" instruction with 16–bit result using the same register

msxq9120 – Passed: single precision integer multiply, indexed mode" instruction with 16–bit result
mxqq9120 – Passed: single precision multiply instruction memory direct–indexed mode
negqc110 – Passed: single precision negate register
negqc111 – Passed: single precision negate register instruction using the same register
nimqb140 – Passed: logical nand instruction, immediate mode
nopqf1a0 – Passed: NOP
notqb110 – Passed: logical not instruction, register–to register
nqqqb120 – Passed: logical nand direct
nrqqb110 – Passed: logical nand instruction, register–to register
orbqb180 – Passed: inclusive or based relative mode without indexing
orbxb190 – F–SD : inclusive–or instruction, base relative mode with indexing
orqqb120 – Passed: inclusive or direct mode
orrqb110 – Passed: inclusive logical or" instruction
orrqb111 – Passed: inclusive logical or" instruction
orxqb120 – Passed: inclusive–or instruction, memory direct mode with indexing
popm51a0 – Passed: POP multiple registers – Special mode
pshm61a0 – Passed: push multiple registers – special mode
rbiq2130 – Passed: reset bit instruction, memory indirect
rbix2130 – Passed: reset bit instruction, memory indirect mode with indexing
rbqq2120 – Passed: reset bit instruction, memory direct mode
rbrq2110 – Passed: reset bit instruction, register mode
rbxq2120 – Passed: reset bit instruction, memory direct mode with indexing
rcfr1140 – F–CNC : Read and Clear Fault Register XIO
rmfs1140 – F–CNC : read memory fault status register XIO
rseth1a0 – F–CNC : initial conditions after reset is done
rsmk1140 – F–CNC : test the "Set Interrupt Mask" and Read Interrupt Mask" XIO instructions
rspi1140 – F–CNC : Read and Set Pending Interrupt and Reset Pending Interrupt register XIO commands
rvbr2110 – Passed: reset variable bit in register
rvbr2111 – Passed: reset variable bit in register
rwsw1140 – F–CNC : Read and Write Status word XIO
sarq3110 – Passed: shift arithmetic – count in register
sarq3111 – Passed: shift arithmetic – count in register
sbbq8180 – Passed: single precision integer subtract base relative mode
sbbx8190 – Passed: single precision integer subtract base relative indexed mode
sbiq2130 – Passed: set bit instruction, memory indirect
sbix2130 – Passed: set bit instruction, memory indirect mode with indexing
sbqq2120 – Passed: set bit instruction, memory direct mode
sbrq2110 – Passed: set bit instruction, register mode
sbxq2120 – Passed: set bit instruction, memory direct mode with indexing
scrq3110 – Passed: shift cyclic – count in register instruction
scrq3111 – Passed: shift cyclic – count in register instruction
simq8140 – Passed: single precision subtract instruction, immediate long mode
sisp8150 – Passed: single precision integer "subtract immediate positive

sjsq4120 – F–CNC : stack and jump to subroutine instruction memory direct mode without indexing
sjsx4120 – F–CNC : stack and jump to subroutine instruction memory direct mode with indexing
sjsx4121 – F–CNC : stack and jump to subroutine instruction memory direct mode with indexing
slbi6130 – Passed: store into lower byte – memory indirect
slbi6131 – Passed: store into lower byte – memory indirect indexed
slcq3110 – Passed: shift left cyclic instruction, register mode
sllq3110 – Passed: shift left logical instruction, register mode
slrq3110 – Passed: shift logical – count in register instruction
slrq3111 – Passed: shift logical – count in register instruction
sojx4120 – F–CNC : subtract one and jump instruction, memory direct mode with indexing
sojx4121 – F–CNC : subtract one and jump instruction, memory direct mode with indexing
sqqq8120 – Passed: single precision integer subtract direct non–indexed" instruction
sraq3110 – Passed: shift right arithmetic instruction, register mode
srlq3110 – Passed: shift right logical instruction, register mode
srmq6120 – Passed: store register through mask
srmx6120 – Passed: store register through mask indexed
srqq8110 – Passed: single precision integer subtract register mode
srqq8111 – Passed: single precision integer subtract register mode" instruction
stbq6180 – Passed: single precision store – based instruction
stbx6190 – Passed: single precision store – based indexed
stci6130 – Passed: single precision store a non–negative constant, indirect" instruction
stci6131 – Passed: store a non–negative constant, indirect indexed
stcq6120 – Passed: single precision store a non–negative constant
stcx6120 – Passed: store a non–negative constant, indexed
stiq6130 – Passed: single precision store – memory indirect
stix6130 – Passed: single precision store – memory indirect indexed
stlb6120 – Passed: store into lower byte – memory direct
stlb6121 – Passed: store into lower byte – memory direct indexed
stmq6120 – Passed: store multiple registers – memory direct
stmx6120 – Passed: store multiple registers – memory direct indexed
stqq6120 – Passed: single precision store" instruction
stub6120 – Passed: store into upper byte – memory direct
stub6121 – Passed: store into lower byte – memory direct indexed
stxq6120 – Passed: single precision store, indexed" instruction
stzi6130 – Passed: single precision store a zero constant, indirect
stzi6131 – Passed: store a ZERO constant, indirect indexed
stzq6120 – Passed: single precision store a zero
stzx6120 – Passed: store a zero constant, indexed
subi6130 – Passed: store into upper byte – memory indirect
subi6131 – Passed: store into upper byte – memory indirect indexed
svbr2110 – Passed: set variable bit instruction, register mode
svbr2111 – Passed: set variable bit instruction, register mode
sxqq8120 – Passed: single precision integer subtract – indexed
tbiq2130 – Passed: test bit instruction, memory indirect mode

tbix2130 – Passed: test bit instruction, memory indirect mode with indexing
tbqq2120 – Passed: test bit instruction, memory direct mode
tbrq2110 – Passed: test bit instruction, register mode
tbxq2120 – Passed: test bit instruction, memory direct mode with indexing
tima1140 – F–CNC : test the Timer A XIO commands
timb1140 – F–CNC : Timer B XIO commands
time1140 – F–CNC : test the Timer A XIO commands
tsbx2120 – Passed: test–and–set bit instruction, memory direct mode with indexing
tvbr2110 – Passed: test variable bit – register mode" instruction
tvbr2111 – Passed: test variable bit – register mode" instruction
ursq41a0 – F–CNC : unstack IC and return from subroutine instruction, special mode
vioq1121 – F–CNC : checks to make sure the Reserved and unused Spare XIOs return a Machine Error when executed in a VIO string
viox1121 – F–CNC : checks to make sure the Reserved and unused Spare XIOs return a Machine Error when executed in a VIO string
xbrqc1a0 – Passed: exchange bytes in register
xmemh1a0 – F–CNC : Expanded memory feature
xmemh1a1 – F–CNC : Expanded memory feature
xmemh1a2 – F–CNC : Expanded memory feature
xmemh1a3 – F–CNC : Expanded memory feature
xmemh1a4 – F–CNC : Expanded memory feature
xormb140 – Passed: exclusive–or instruction, immediate
xorqb120 – Passed: exclusive–or direct
xorrb110 – Passed: exclusive–or instruction, register–to–register
xorrb111 – Passed: exclusive–or instruction, register–to–register using the same register
xorxb120 – Passed: exclusive–or instruction, memory direct mode with indexing
xwrqc1a – Passed: exchange words in registers
xwrqc1a1 – Passed: exchange words in registers