Experiment #4: Introduction to Secure Memory Management

Steve Comer

Implementation Due: 29 Oct 2012
Report Due: 7 Nov 2012

Steve Comer

Table of Contents

1.0 Introduction

This lab consisted of two related subsections. Both involved using VHDL programming. VHDL is a nested acronym that stands for Very High Speed Integrated Circuit (VHISIC) Hardware Description Language (VHDL). The first task was to create a Random Access Memory (RAM) component in VHDL with sixteen 2-bit memory elements. The second task was to add security in the form of a reference monitor to the RAM. Both the RAM and the secure RAM components output to seven segment displays.

2.0 Method

2.1 Background Information

2.1.1 RAM

Random access memory may be read from or written to. The read operation is synchronous and requires a valid address and a high readEnable signal. The value stored at the indicated memory address is then output. The write operation is also synchronous and requires a valid address and a high writeEnable signal. The 2-bit value from input will be stored in the indicated address of memory. When a write operation occurs, the output is set to "00". Simultaneous reading and writing is not allowed. If the user attempts to read and write simultaneously, no operation is performed. Shown in Figure 2.1, the RAM component has five inputs and one output. The inputs include readEnable, writeEnable, clock, 4-bit address, and 2-bit dataIn. The output is 2-bit dataOut.

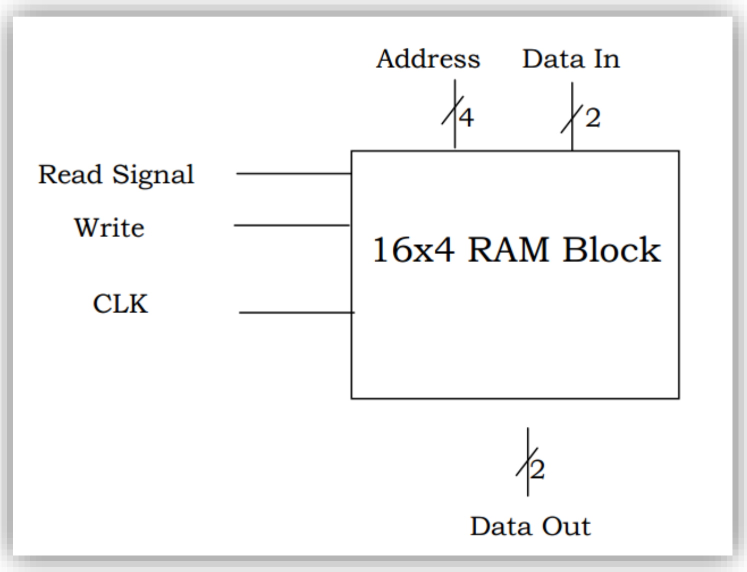


Figure 2.1 - RAM Diagram

2.1.2 Reference Monitor

The reference monitor adds security to the RAM component. This part of the project adds four processes, each with its own read and write permissions. The reference monitor sits between the user and the RAM. It checks user input against access control permissions for the given process. It also monitors output coming from the RAM. If user input violates access control policies, the reference monitor discards the input and clears output coming from the RAM. If user input is valid and allowed, this portion operates in the same manner as the previously described RAM. The only additional input for the reference monitor is a 2-bit Process input. Two additional LEDs are used to display type and address violations. An address error occurs when the given process has no access at the input address. A type error occurs when the process has some type of access at the input address, but the wrong type of access is attempted.

2.2 Procedure

2.2.1 Design

The method underlying the VHDL code in this project was to check for errors, instead of checking for legal access. If no error conditions exist, input and output pass between the user and RAM as if the reference monitor does not exist.

2.2.2 VHDL Coding

Variables were used as the primary storage element within the different processes. The values contained in these variables were used to update corresponding signals at the end of a given process.

2.2.2.1 RAM

The RAM component contains 16 separate 2-bit memory elements. The array data structure fits well for implementing the RAM. An array of 2-bit logic vectors with 16 elements was used in this case. The individual memory elements are described by an index in the array. The first element is array[0] and the last is array[15]. A previously unused function "conv_integer(x)" was useful for converting the 4-bit address input to an array index. Read and write operations are only performed on a rising edge of the clock signal. The RAM is also coded to ensure that the read and write operations are mutually exclusive, meaning that only one operation can be performed during one clock cycle.

2.2.2.2 Reference Monitor

The reference monitor ensures that only legal input is passed from the user to the RAM. If access control policies are violated, a clear signal is set high and user input is discarded. The clear signal is passed to the RAM. Upon receiving a clear signal, the RAM clears the current output value sets the output to "00". The access control policies are defined in Figure 2.2. The strategy for coding the reference monitor uses flags to identify error conditions. Instead of checking for valid access, the reference monitor checks for invalid access. The access control policies are separated by process. If the user attempts access which violates a particular process's permissions then one of the error flags will be set, based on the error. If the access is allowed, the reference monitor passes the input through to the RAM.

Process	Read Addresses	Write Addresses
0 (00)	0-7	4-7
1 (01)	4-7	0-3
2 (10)	0-15	0-14
3 (11)	None	0-15

Figure 2.2 - Process Permissions

2.2.3 Pin Assignments

The full list of pin assignments for the ALU is given in Appendix B. Pin assignments are explained in the order in which they appear in the appendix.

2.2.3.1 RAM Pin Assignments

address_d[3 DOWNTO 0]: These are four input switches used to select the 4-bit input address. These switches can be toggled in different combinations to represent all possible values of the input address. Note that address_d[3 DOWNTO 0] corresponds to SW[17 DOWNTO 14] on the DE2 board.

clk_d: This is a push-button input to the circuit. Pushing this button down and then releasing it corresponds to one clock cycle for the circuit. The button gives a high signal when it is pressed. This corresponds to KEY[3] on the DE2 board.

clkOut_d: This is a green LED which shows the current value of the clock signal. A high clock signal turns on the LED. This LED was chosen because it is physically close to the push-button input used for clock. This corresponds to LEDG[7] on the DE2 board.

dataIn_d[1 DOWNTO 0]: These are two input switches used to select the 2-bit input data value. These switches can be toggled in different combinations to represent all possible values of the input. Note that dataIn_d[1 DOWNTO 0] corresponds to SW[13 DOWNTO 12] on the DE2 board.

readEnable_d: This input switch determines whether or not a read operation is performed. If the readEnable signal is high on a rising clock edge, then dataOut will contain the value at the input address specified by the user. This corresponds to SW[1] on the DE2 board.

sevenSegmentOut_d[6 DOWNTO 0]: These seven outputs are segments 'a' through 'g', forming one hexadecimal character on the DE2 board. These segments are active low because they turn on when set to 0. This displays the value of the dataOut. Note that sevenSegmentOut[x] corresponds to HEX7 [x] on the DE2 board.

writeEnable_d: This input switch determines whether or not a write operation is performed. If the writeEnable signal is high on a rising clock edge, then dataIn will be written to the input address specified by the user. This corresponds to SW[0] on the DE2 board.

2.2.3.2 Reference Monitor Pin Assignments

addr_issue_LED: This is a green LED which shows the current value of the address issue flag. A high addr_issue signal turns on the LED. This corresponds to LEDG[4] on the DE2 board.

address[3 DOWNTO 0]: These are four input switches used to select the 4-bit input address. These switches can be toggled in different combinations to represent all possible values of the input address. Note that address[3 DOWNTO 0] corresponds to SW[17 DOWNTO 14] on the DE2 board.

clk: This is a push-button input to the circuit. Pushing this button down and then releasing it corresponds to one clock cycle for the circuit. The button gives a high signal when it is pressed. This corresponds to KEY[3] on the DE2 board.

clkOut: This is a green LED which shows the current value of the clock signal. A high clock signal turns on the LED. This LED was chosen because it is physically close to the push-button input used for clock. This corresponds to LEDG[7] on the DE2 board.

dataIn[1 DOWNTO 0]: These are two input switches used to select the 2-bit input data value. These switches can be toggled in different combinations to represent all possible values of the input. Note that dataIn[1 DOWNTO 0] corresponds to SW[13 DOWNTO 12] on the DE2 board.

proc[1 DOWNTO 0]: These are two input switchse used to select the 2-bit user process. These switches can be toggled in different combinations to represent any of the four processes. Note that proc[1 DOWNTO 0] corresponds to SW[11 DOWNTO 10] on the DE2 board.

sevenSegmentOut_a[6 DOWNTO 0]: These seven outputs are segments 'a' through 'g', forming one hexadecimal character on the DE2 board. These segments are active low because they turn on when set to 0. This displays the value of the address. Note that sevenSegmentOut[x] corresponds to HEX4 [x] on the DE2 board.

sevenSegmentOut_data[6 DOWNTO 0]: These seven outputs are segments 'a' through 'g', forming one hexadecimal character on the DE2 board. These segments are active low because they turn on when set to 0. This displays the value of dataOut. Note that sevenSegmentOut[x] corresponds to HEX7 [x] on the DE2 board.

sevenSegmentOut_p[6 DOWNTO 0]: These seven outputs are segments 'a' through 'g', forming one hexadecimal character on the DE2 board. These segments are active low because they turn on when set to 0. This displays the value of the process. Note that sevenSegmentOut[x] corresponds to HEX5 [x] on the DE2 board.

type_issue_LED: This is a green LED which shows the current value of the type issue flag. A high type_issue signal turns on the LED. This corresponds to LEDG[5] on the DE2 board.

writeEnable: This input switch determines whether or not a write operation is performed. If the writeEnable signal is high on a rising clock edge, then dataIn will be written to the input address specified by the user. This corresponds to SW[0] on the DE2 board.

2.3 Problems and Solutions

Problem: Conversion from logic vector to integer causes errors.
First solution: Use case statement to assign all possible input logic vectors to their integer equivalents.
Better solution: Change conversion function from "to_integer(x)" to "conv_integer(x)".

Problem: Multiple constant drivers for signals in the reference monitor.
Solution: Move all modifications to a given signal into the same process.

Problem: Reference monitor checks for legal access and allowing based on legal access.
Solution: Think about the problem in terms of checking for illegal cases and allow access otherwise.

Problem: Two clock cycles required to change output back to legal state after entering error state.
Solution: No solution found.

2.4 Materials and Tools Used

The materials and tools used in this experiment are as follows:

1. Quartus II Software
2. Altera DE 2 Board with Cyclone II – EP2C35F672C6 FPGA

3.0 Results

3.1 Observations

Because this project was composed almost exclusively of programming, observations relate to the output of the FPGA in the form of the results. The project functioned correctly with the exception of the double clock issue for error conditions, described in section 2.3: Problems and Solutions.

3.2 Testing Procedure

The demonstration of the project involved implementation of two separate test plans, one for the RAM and one for the Reference Monitor added to the RAM. Each plan is detailed in the following subsections. The test plans do not include every possible permutation of input values. The test plans use selected inputs ,which could be problematic, to demonstrate the effectiveness of the solution.

3.2.1 RAM Test Plan

Test plan:

1. Read from every register to show that all values are initially "00".
2. Write the value "01" to memory.
3. Read the value stored in memory and show that it is different from "00" and equal to "01".
4. Repeat steps 2 and 3 for the following addresses: {0000, 0001, 0010, 0100, 1000}.
5. Set readEnable and writeEnable high simultaneously and show that no operation is performed.
6. Set readEnable and writeEnable low simultaneously and show that no operation is performed.
7. Show changing input without a clock cycle to show that the RAM is synchronous.

3.2.2 Reference Monitor Test Plan

Preface demonstration with explanation of error in transitioning from error state to legal access. Error LEDs update on first clock cycle, as expected. Update of output does not occur until one clock cycle later.

Access Control Policy (from Figure 2.2):

Process	Read Addresses	Write Addresses
0 (00)	0-7	4-7
1 (01)	4-7	0-3
2 (10)	0-15	0-14
3 (11)	None	0-15

Test plan:

1. Demonstrate process 0 (process = "00"):
    a. Show that attempts to access memory locations 8-15 result in an address error.
    b. Show that attempting to write to memory locations 0-3 causes a type error.
    c. Show valid operation of writing "01" to memory address 4.
    d. Show valid operation of reading "01" from memory address 4.
2. Demonstrate process 1 (process = "01"):
    a. Show that attempts to access memory locations 8-15 result in an address error.
    b. Show that attempting to write to memory locations 4-7 causes a type error.
    c. Show that attempting to read from memory locations 0-3 causes a type error.
    d. Show valid operation of writing "01" to memory address 0.
    e. Prove that write operation worked by reading "01" from address 0 with process 0.
    f. Show valid operation of reading "01" from address 4, written by process 0 in step 1c.
3. Demonstrate process 2 (process = "10"):
    a. Show that process 2 is not able to generate an address error.
    b. Show that attempting to write to memory location causes a type error.
    c. Show that read operation is not able to generate a type error.
    d. Show valid operation of writing "01" to address 14.
    e. Show valid operation of reading "01" from address 14.

4. Demonstrate process 3 (process = "11"):
    a. Show that process 3 is not able to generate an address error.
    b. Show that any attempt to perform read operation causes a type error.
    c. Show that write operation is not able to generate a type error.
    d. Show valid operation of writing "01" to address 12.
    e. Prove that write operation worked by reading "01" from address 12 with process 2.
5. Demonstrate switching processes:
    a. From valid state, transition to illegal state by switching processes.
        i. With read high and write low and address 0, change processes from 0 to 1.
    b. From illegal state, transition to illegal state by switching processes.
        i. With read high and write low and address 8, change processes from 0 to 1.
    c. From valid state, transition to valid state by switching processes.
        i. With read high and write low and address 4, change processes from 0 to 1.

3.3 Experiment Evaluation

The experiment was successful with one exception. One extra clock cycle was required to fully transition from an error state to the next valid state in the Reference Monitor. The standalone RAM worked exactly as specified. Aside from the double clock problem, the Reference Monitor worked as intended. The double clock problem, while not accurate to the specifications of the experiment, does not cause secure information to be released by the reference monitor. All access control and security policies are implemented effectively. Lessons learned from the double clock problem are addressed in the Conclusion.

3.4 Warning Explanations

Quartus gives four main types of messages when compiling code: info, warnings, critical warnings, and errors. The same types of warnings occurred in both the RAM and in the Reference Monitor. For this reason, the warnings will be addressed together.

3.4.1 Errors

Errors are the most important type of message. They explicitly describe to the user where in the code or design Quartus found a problem. It is essential that all errors are removed from a project, since the code will not compile correctly until the errors are fixed. It is also important to realize that just because all the errors are removed from a project does not mean the code works as intended.

3.4.2 Critical Warnings

The second most important message type is a critical warning. While these do not explicitly mean that something is wrong with the code, the user should examine these warnings when attempting to debug a functional issue.

Critical Warning 1:
Critical Warning (332012): Synopsys Design Constraints File file not found: 'Lab1_new.sdc'. A Synopsys Design Constraints File is required by the TimeQuest Timing Analyzer to get proper timing constraints. Without it, the Compiler will not properly optimize the design.

Reason:
The project did not supply a timing constraint file, so Quartus was unable to test the timing of the circuit to see if it will work in its environment.

Critical Warning 2:
Critical Warning (332148): Timing requirements not met

Reason:
This critical warning is directly related to critical warning 1. Because a timing constraints file was not supplied, Quartus was unable to do timing analysis, so the timing requirements were not met.

3.4.3 Warnings
Warnings are common in Quartus projects. Due to the fact that Quartus compiles to an industry standard set of specifications, it is concerned with seemingly insignificant details. Quartus displayed five different types of warnings. They are described below.

Warning 1:
Warning (13024): Output pins are stuck at VCC or GND
Warning (13410): Pin "sevenSegmentOut_d[1]" is stuck at GND
Warning (13410): Pin "sevenSegmentOut_data[1]" is stuck at GND
Warning (13410): Pin "sevenSegmentOut_p[1]" is stuck at GND

Reason:
Quartus notices that these segments are never turned off. This is correct based on the implementation of the hexadecimal display driver.

Warning 2:
Warning (306006): Found output pins without output pin load capacitance assignment

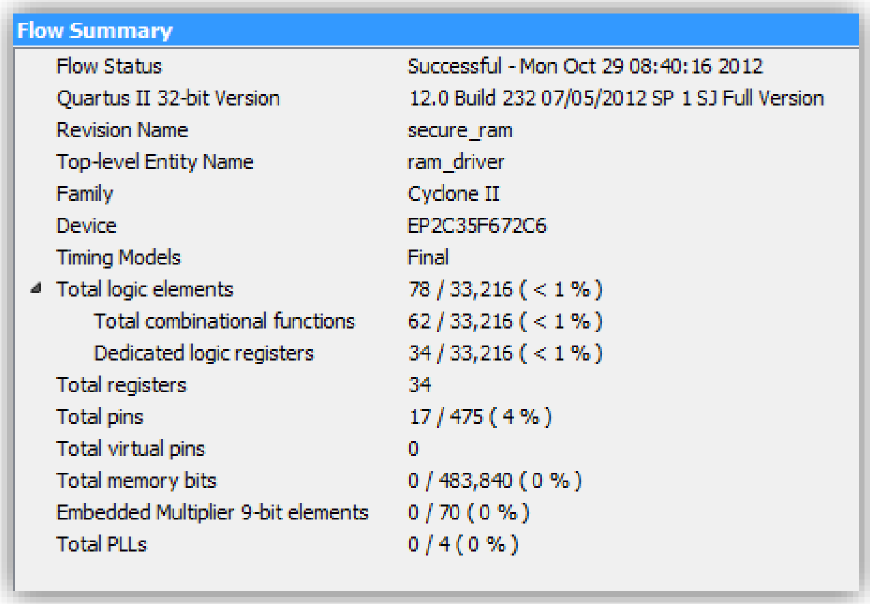
Reason:
This warning states that the reported timing of these pins will be faster than in reality because load capacitance is unknown.

Warning 3:
Warning (169174): The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'.

Reason:
The pin voltages are not specified, therefore Quartus has a fail-safe default of setting them to ground.

3.5 Quartus Flow Summary
Compared with an experiment that has been implemented manually in hardware, the Flow Summaries here are not as useful. This is because there is no human implementation that the Quartus implementation can be compared against.

3.5.1 RAM
The Flow Summary for RAM is shown in Figure 3.1. The final design includes 78 combinational logic elements and 34 dedicated logic registers. There are 17 total pins used for input and output. The 34 registers come from the 16 instances of 2-bit memory elements plus 2 more registers for the dataIn input for a total of 34. For the 78 combinational logic elements, the estimate included the following: 4 elements for the NOT functions in the driver file, 7 elements for Hexadecimal display driver, 13 elements for each IF statement (5 total) within the RAM module. This comes to a total of 76 elements which is close to the 78 delivered by Quartus.



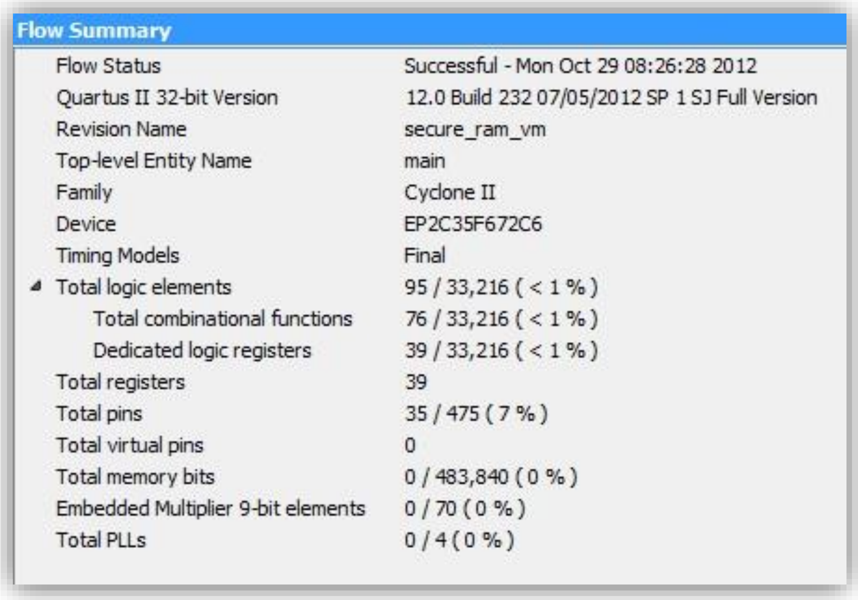
Flow Summary

Flow Status	Successful - Mon Oct 29 08:40:16 2012
Quartus II 32-bit Version	12.0 Build 232 07/05/2012 SP 1 SJ Full Version
Revision Name	secure_ram
Top-level Entity Name	ram_driver
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	78 / 33,216 (< 1 %)
Total combinational functions	62 / 33,216 (< 1 %)
Dedicated logic registers	34 / 33,216 (< 1 %)
Total registers	34
Total pins	17 / 475 (4 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 3.1 RAM Flow Summary

3.5.2 Reference Monitor

The Flow Summary for RAM is shown in Figure 3.2. The final design includes 95 combinational logic elements and 39 dedicated logic registers. There are 35 total pins used for input and output. The 39 registers come from the 16 instances of 2-bit memory elements, 2 registers for the dataIn input, 2 registers for the process input, and finally 3 registers for error checking flags. For the 95 combinational logic elements, the estimate included the following: 4 elements for the NOT functions in the driver file, 21 elements for Hexadecimal display drivers, 5 elements for each branch of the CASE statement (5 total) within the Reference Monitor module, 40 elements for the combinational logic within the CASE statement for error checking, 20 elements for the IF statements within the RAM file. This comes to a total of 90 elements which is close to the 95 delivered by Quartus.



Flow Summary

Flow Status	Successful - Mon Oct 29 08:26:28 2012
Quartus II 32-bit Version	12.0 Build 232 07/05/2012 SP 1 SJ Full Version
Revision Name	secure_ram_vm
Top-level Entity Name	main
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	95 / 33,216 (< 1 %)
Total combinational functions	76 / 33,216 (< 1 %)
Dedicated logic registers	39 / 33,216 (< 1 %)
Total registers	39
Total pins	35 / 475 (7 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 3.2 Reference Monitor Flow Summary

4.0 Conclusion

The first objective of this experiment was to implement a RAM component. This project effectively completed this task. The second objective was to implement a Reference Monitor and add it to the RAM. This objective proved to be a challenge. The second objective could be considered a failure because the final product does not implement a completely correct Reference Monitor. Failure to meet the second objective led to many hours being spent trying to correct a double clocking error. This time was not wasted however. The struggle allowed me to understand the difficulties of programming hardware. One fact that bothered me was that, in a higher level language, my code should have been functionally equivalent to that of others whose projects worked correctly. This frustrated me, but helped to impress upon me the fact that, while useful, the circuit-building algorithms used by Quartus are not perfect. I may not have accomplished the outlined objective, but I now understand the struggles felt by many in industry.

A. Appendix A – Code

A.1 ram_driver.vhd

```
----------------------------------------------------------------------------------------------
-- Lab 4                                                                                    --
-- ram_driver                                                                               --
-- Steve Comer                                                                              --
-- Updated 27 Oct 2012                                                                      --
--          Driver for ram                                                                  --
----------------------------------------------------------------------------------------------

-- Import the necessary libraries.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

----------------------------------------------------------------------------------------------

ENTITY ram_driver IS

-- Input and output for the RAM_driver
-- Note: *_d shows that a value is an instance in the driver, not the component itself
PORT(
        clk_d               : IN STD_LOGIC;
        readEnable_d        : IN STD_LOGIC;
        writeEnable_d       : IN STD_LOGIC;
        address_d           : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dataIn_d            : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        sevenSegmentOut_d   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        clkOut_d            : OUT STD_LOGIC
);

END ram_driver;

----------------------------------------------------------------------------------------------

ARCHITECTURE Behavioral OF ram_driver IS

        -- See ram.vhd
        COMPONENT ram
                PORT(
                        clk          : IN STD_LOGIC;
                        readEnable   : IN STD_LOGIC;
                        writeEnable  : IN STD_LOGIC;
                        address      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                        dataIn       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        dataOut      : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
                );
        END COMPONENT;

        -- See display_driver.vhd
        COMPONENT HexDriver
                PORT(
                        numberToDisplay  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
                        sevenSegmentOut  : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
                );
        END COMPONENT;

        -- Ensures that values updated in ram are displayed correctly in HexDriver
```

```
   SIGNAL dataOut_signal : STD_LOGIC_VECTOR(1 DOWNTO 0);
            SIGNAL clkNeg_signal  : STD_LOGIC;

BEGIN

            Negate_Clock_Process : PROCESS(clk_d)
            BEGIN
                        clkNeg_signal <= NOT clk_d;                        -- store the reversed value in the clkNeg wire
                        clkOut_d <= NOT clk_d;                       -- use this value for the clock output check
            END PROCESS Negate_Clock_Process;

            ram_d : ram PORT MAP(clkNeg_signal,readEnable_d,writeEnable_d,address_d,dataIn_d,dataOut_signal);
            hexDriver_d : HexDriver PORT MAP(dataOut_signal,sevenSegmentOut_d);

END Behavioral;
```

A.2 ram.vhd

```
----------------------------------------------------------------------------------------------
-- Lab 4                                                                                    --
-- ram                                                                                      --
-- Steve Comer                                                                              --
-- Updated 27 Oct 2012                                                                      --
--              16x4 RAM (16 addresses, 2-bit data)                                         --
----------------------------------------------------------------------------------------------

-- Import the necessary libraries.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

----------------------------------------------------------------------------------------------

ENTITY ram IS

-- Input and output for the RAM
PORT(
        clk          : IN STD_LOGIC;
        readEnable   : IN STD_LOGIC;
        writeEnable  : IN STD_LOGIC;
        address      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dataIn                  : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        dataOut      : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
);

END ram;

----------------------------------------------------------------------------------------------

ARCHITECTURE Behavioral OF ram IS

        TYPE tRam IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL sRam : tRam;

BEGIN

        -- Performs write operation first
        Read_Write_Process : PROCESS(clk)
        BEGIN
                IF rising_edge(clk) THEN
                        IF writeEnable = '1' THEN
                                IF readEnable = '0' THEN
                                        sRam(conv_integer(address)) <= dataIn;
                                        dataOut <= "00";
                                END IF;
                        ELSIF readEnable = '1' THEN
                                IF writeEnable = '0' THEN
                                        dataOut <= sRam(conv_integer(address));
                                END IF;
                        END IF;
                END IF;
        END PROCESS Read_Write_Process;


END Behavioral;
```

A.3 display_driver.vhd

```
-- Lab 4
-- Hex Display Driver
-- Steve Comer
-- Updated 27 Oct 2012
--          HexDriver takes a 2-bit number and displays it

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY HexDriver IS

PORT(numberToDisplay  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
          sevenSegmentOut  : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END HexDriver;

ARCHITECTURE Behavioral OF HexDriver IS
        BEGIN

                HexDriver : PROCESS(numberToDisplay)
                        BEGIN
                                CASE numberToDisplay IS                  -- Disp
                                        WHEN "00" => sevenSegmentOut <= "1000000";              -- 0
                                        WHEN "01" => sevenSegmentOut <= "1111001";              -- 1
                                        WHEN "10" => sevenSegmentOut <= "0100100";              -- 2
                                        WHEN "11" => sevenSegmentOut <= "0110000";              -- 3
                                        WHEN OTHERS => sevenSegmentOut <= "1111111";            -- null
                                END CASE;

                        END PROCESS HexDriver;

        END Behavioral;
```

A.4 main.vhd

```
-----------------------------------------------------------------------------------------------
-- Lab_4_RM                                                                                 --
-- main                                                                                     --
-- Steve Comer                                                                              --
-- Updated 28 Oct 2012                                                                      --
--          main program for ram with reference monitor                                     --
--                    passes input to reference monitor                                     --
--          receives output from reference monitor                                          --
--          passes output from reference monitor to displays/LEDs                           --
--          problems:                                                                       --
--                    two clock cycles for transition from error state                      --
-----------------------------------------------------------------------------------------------

-- Import the necessary libraries.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

-----------------------------------------------------------------------------------------------

ENTITY main IS

-- Input and output for main
PORT(
          clk               : IN STD_LOGIC;
          readEnable        : IN STD_LOGIC;
          writeEnable       : IN STD_LOGIC;
          address           : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          proc              : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          dataIn            : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          sevenSegmentOut_data : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          sevenSegmentOut_p    : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          sevenSegmentOut_a    : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          type_issue_LED    : OUT STD_LOGIC;
          addr_issue_LED    : OUT STD_LOGIC;
          clkOut            : OUT STD_LOGIC
);

END main;

-----------------------------------------------------------------------------------------------

ARCHITECTURE Behavioral OF main IS

          -- See ref_mon.vhd
          COMPONENT ref_mon
                    PORT(
                              clk_rm            : IN STD_LOGIC;
                              readEnable_rm     : IN STD_LOGIC;
                              writeEnable_rm    : IN STD_LOGIC;
                    address_rm        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                    dataIn_rm                          : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                    dataOut_rm        : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                    proc_rm                            : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                    type_issue_LED                : OUT STD_LOGIC;
                    addr_issue_LED        : OUT STD_LOGIC
               );
          END COMPONENT;

          -- See display_driver_2.vhd
          COMPONENT HexDriver_2
               PORT(
                    numberToDisplay  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
                    sevenSegmentOut  : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
               );
          END COMPONENT;

          -- See display_driver_4.vhd
          COMPONENT HexDriver_4
               PORT(
                    numberToDisplay  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                    sevenSegmentOut  : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
               );
          END COMPONENT;

          -- Reference Monitor <==> Hex Driver
  SIGNAL dataOut_signal     : STD_LOGIC_VECTOR(1 DOWNTO 0);
          SIGNAL clkNeg_signal      : STD_LOGIC;

BEGIN

          Negate_Clock_Process : PROCESS(clk)
          BEGIN
               clkNeg_signal <= NOT clk;
               clkOut <= NOT clk;
          END PROCESS Negate_Clock_Process;

          ref_mon_instance : ref_mon PORT MAP(clkNeg_signal,readEnable,writeEnable,address,
             dataIn,dataOut_signal,proc,type_issue_LED,addr_issue_LED);
          hexDriver_data   : HexDriver_2 PORT MAP(dataOut_signal,sevenSegmentOut_data);
          hexDriver_proc   : HexDriver_2 PORT MAP(proc,sevenSegmentOut_p);
          hexDriver_addr   : HexDriver_4 PORT MAP(address,sevenSegmentOut_a);

END Behavioral;
```

A.5 ref_mon.vhd

```
------------------------------------------------------------------------------------------------
-- Lab_4_RM                                                                                    --
-- ref_mon                                                                                     --
-- Steve Comer                                                                                 --
-- Updated 28 Oct 2012                                                                         --
--                     Process                                                                 --
--                      0                   Read: 0-7  Write: 4-7                              --
--                      1                   Read: 4-7  Write: 0-3                              --
--                      2                   Read: 0-15 Write: 0-14                             --
--                      3                   Read: none Write: 0-15                             --
------------------------------------------------------------------------------------------------

-- Import the necessary libraries.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

------------------------------------------------------------------------------------------------

ENTITY ref_mon IS

-- Input and output for ref_mon
PORT(
        clk_rm          : IN STD_LOGIC;
        readEnable_rm   : IN STD_LOGIC;
        writeEnable_rm  : IN STD_LOGIC;
        address_rm      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dataIn_rm       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        dataOut_rm      : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        proc_rm         : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        type_issue_LED  : OUT STD_LOGIC;
        addr_issue_LED  : OUT STD_LOGIC
);

END ref_mon;

------------------------------------------------------------------------------------------------

ARCHITECTURE Behavioral OF ref_mon IS

        -- See ram.vhd
        COMPONENT ram
                PORT(
                        clk         : IN STD_LOGIC;
                        readEnable  : IN STD_LOGIC;
                        writeEnable : IN STD_LOGIC;
                        address     : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                        dataIn      : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        dataOut     : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                        clear       : IN STD_LOGIC
```

```
		);
	END COMPONENT;

	-- Reference Monitor <==> RAM
	SIGNAL clear_flag  : STD_LOGIC;
	SIGNAL type_issue  : STD_LOGIC;
	SIGNAL addr_issue  : STD_LOGIC;

BEGIN

	Reference_Monitor_Process : PROCESS(clk_rm)

		-- local variables
		VARIABLE addr_int   : INTEGER;
		VARIABLE clear_var  : STD_LOGIC;
		VARIABLE type_var   : STD_LOGIC; -- access type error
		VARIABLE addr_var   : STD_LOGIC; -- access address error

	BEGIN
		IF rising_edge(clk_rm) THEN

			-- if readEnable NAND writeEnable
			IF (NOT(readEnable_rm = '1' AND writeEnable_rm = '1')) THEN

				-- reset flags
				clear_var  := '0';
				addr_var   := '0';
				type_var   := '0';

				addr_int := conv_integer(address_rm);

				CASE proc_rm IS
					WHEN "00" =>
						-- valid address
						IF (addr_int >= 0 AND addr_int <= 7) THEN
							-- illegal read cases
								-- none
							-- illegal write cases
							IF (writeEnable_rm = '1' AND addr_int <= 3) THEN
								type_var := '1';
							END IF;
						-- invalid address
						ELSE
							addr_var := '1';
						END IF;

					WHEN "01" =>
						-- valid address
						IF (addr_int >= 0 AND addr_int <= 7) THEN
							-- illegal read cases
							IF (readEnable_rm = '1' AND addr_int <= 3) THEN
								type_var := '1';
							-- illegal write cases
							ELSIF (writeEnable_rm = '1' AND addr_int >= 4) THEN
```

```
                                                type_var := '1';
                                        END IF;
                                -- invalid address
                                ELSE
                                        addr_var := '1';
                                END IF;

                        WHEN "10" =>
                                -- valid address
                                IF (addr_int >= 0 AND addr_int <= 15) THEN
                                        -- illegal read cases
                                                -- none
                                        -- illegal write cases
                                        IF (writeEnable_rm = '1' AND addr_int = 15) THEN
                                                type_var := '1';
                                        END IF;
                                -- invalid address
                                ELSE
                                        addr_var := '1';
                                END IF;
                        WHEN "11" =>
                                -- valid address
                                IF (addr_int >= 0 AND addr_int <= 15) THEN
                                        -- illegal read cases
                                        IF (readEnable_rm = '1') THEN
                                                type_var := '1';
                                        -- illegal write cases
                                                -- none
                                        END IF;
                                -- invalid address
                                ELSE
                                        addr_var := '1';
                                END IF;
                        WHEN OTHERS =>
                                -- Security Condition
                                type_var := '1';
                                addr_var := '1';
                END CASE;
        END IF;

        -- clear conditions
        clear_var := type_var OR addr_var;

        -- update signals
        clear_flag <= clear_var;
        type_issue_LED <= type_var;
        addr_issue_LED <= addr_var;
    END IF;
END PROCESS Reference_Monitor_Process;

ram_instance : ram PORT
MAP(clk_rm,readEnable_rm,writeEnable_rm,address_rm,dataIn_rm,dataOut_rm,clear_flag);

END Behavioral;
```

A.6 ram.vhd

```
--------------------------------------------------------------------------------------------
-- Lab_4_RM                                                                               --
-- ram                                                                                    --
-- Steve Comer                                                                            --
-- Updated 28 Oct 2012                                                                    --
--          16x4 RAM (16 addresses, 2-bit data)                                           --
--------------------------------------------------------------------------------------------

-- Import the necessary libraries.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

--------------------------------------------------------------------------------------------

ENTITY ram IS

-- Input and output for the RAM
PORT(
        clk         : IN STD_LOGIC;
        readEnable  : IN STD_LOGIC;
        writeEnable : IN STD_LOGIC;
        address     : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dataIn              : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        dataOut     : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        clear       : IN STD_LOGIC
);

END ram;

--------------------------------------------------------------------------------------------

ARCHITECTURE Behavioral OF ram IS

        TYPE tRam IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL sRam : tRam;

BEGIN

        -- Writes to RAM or Reads from RAM
        Read_Write_Process : PROCESS(clk,clear)

        VARIABLE dataOut_var : STD_LOGIC_VECTOR(1 DOWNTO 0);

        BEGIN
                -- asynchronous clear
                IF clear = '1' THEN
                        dataOut_var := "00";
                -- synchronous remainder
                ELSIF rising_edge(clk) THEN
                        IF (readEnable = '1' AND writeEnable = '1') OR
                            (readEnable = '0' AND writeEnable = '0') THEN
                            dataOut_var := "00";
                        ELSIF writeEnable = '1' THEN
                            sRam(conv_integer(address)) <= dataIn;
                            dataOut_var := "00";
                        ELSIF readEnable = '1' THEN
                            dataOut_var := sRam(conv_integer(address));
                        ELSE
                            dataOut_var := "00";
                        END IF;
                END IF;

                dataOut <= dataOut_var;

        END PROCESS Read_Write_Process;

END Behavioral;
```

A.7 display_driver_2.vhd

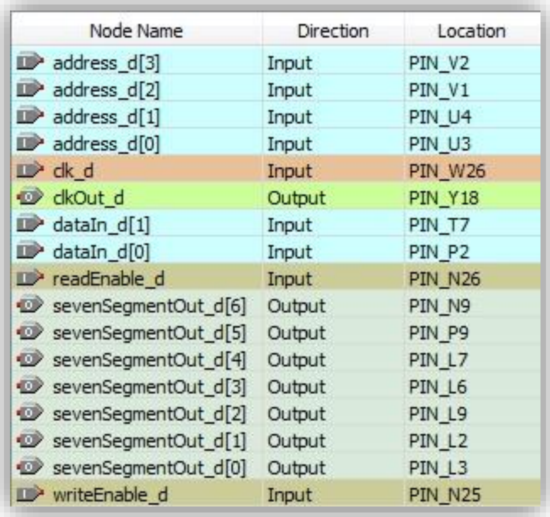
```
----------------------------------------------------------------------------------
-- Lab_4_RM
-- display_driver_2
-- Steve Comer
-- Updated 17 Oct 2012
--		HexDriver takes a 2-bit number and displays it
----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY HexDriver_2 IS

PORT(
		numberToDisplay  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
		sevenSegmentOut  : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END HexDriver_2;

ARCHITECTURE Behavioral OF HexDriver_2 IS
		BEGIN

				HexDriver_2 : PROCESS(numberToDisplay)
						BEGIN
								CASE numberToDisplay IS				-- Disp
										WHEN "00" => sevenSegmentOut <= "1000000";			-- 0
										WHEN "01" => sevenSegmentOut <= "1111001";			-- 1
										WHEN "10" => sevenSegmentOut <= "0100100";			-- 2
										WHEN "11" => sevenSegmentOut <= "0110000";			-- 3
										WHEN OTHERS => sevenSegmentOut <= "1111111";		-- null
								END CASE;

						END PROCESS HexDriver_2;

		END Behavioral;
```

A.8 display_driver_4.vhd

```
----------------------------------------------------------------------------------------------
-- Lab_4_RM
-- display_driver_4
-- Steve Comer
-- Updated 17 Oct 2012
--          HexDriver takes a 4-bit number and displays it
----------------------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY HexDriver_4 IS
        PORT(
                numberToDisplay    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
          sevenSegmentOut          : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END HexDriver_4;

ARCHITECTURE Behavioral OF HexDriver_4 IS
        BEGIN

                HexDriver_4 : PROCESS(numberToDisplay)
                        BEGIN
                                CASE numberToDisplay IS                 -- Disp
                                        WHEN "0000" => sevenSegmentOut <= "1000000";    -- 0
                                        WHEN "0001" => sevenSegmentOut <= "1111001";    -- 1
                                        WHEN "0010" => sevenSegmentOut <= "0100100";    -- 2
                                        WHEN "0011" => sevenSegmentOut <= "0110000";    -- 3
                                        WHEN "0100" => sevenSegmentOut <= "0011001";    -- 4
                                        WHEN "0101" => sevenSegmentOut <= "0010010";    -- 5
                                        WHEN "0110" => sevenSegmentOut <= "0000010";    -- 6
                                        WHEN "0111" => sevenSegmentOut <= "1111000";    -- 7
                                        WHEN "1000" => sevenSegmentOut <= "0000000";    -- 8
                                        WHEN "1001" => sevenSegmentOut <= "0010000";    -- 9
                                        WHEN "1010" => sevenSegmentOut <= "0001000";    -- A
                                        WHEN "1011" => sevenSegmentOut <= "0000011";    -- b
                                        WHEN "1100" => sevenSegmentOut <= "1000110";    -- C
                                        WHEN "1101" => sevenSegmentOut <= "0100001";    -- d
                                        WHEN "1110" => sevenSegmentOut <= "0000110";    -- E
                                        WHEN "1111" => sevenSegmentOut <= "0001110";    -- F
                                        WHEN OTHERS => sevenSegmentOut <= "1111111";  --  null
                                END CASE;

                        END PROCESS HexDriver_4;

        END Behavioral;
```

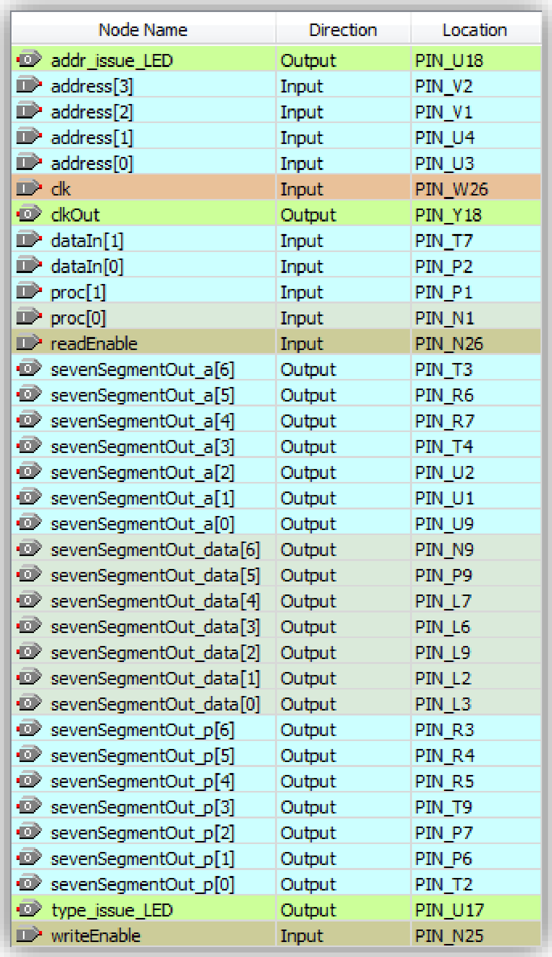
B. Appendix B – Quartus II Pinout

B.1 RAM Pinout



Node Name	Direction	Location
address_d[3]	Input	PIN_V2
address_d[2]	Input	PIN_V1
address_d[1]	Input	PIN_U4
address_d[0]	Input	PIN_U3
clk_d	Input	PIN_W26
clkOut_d	Output	PIN_Y18
dataIn_d[1]	Input	PIN_T7
dataIn_d[0]	Input	PIN_P2
readEnable_d	Input	PIN_N26
sevenSegmentOut_d[6]	Output	PIN_N9
sevenSegmentOut_d[5]	Output	PIN_P9
sevenSegmentOut_d[4]	Output	PIN_L7
sevenSegmentOut_d[3]	Output	PIN_L6
sevenSegmentOut_d[2]	Output	PIN_L9
sevenSegmentOut_d[1]	Output	PIN_L2
sevenSegmentOut_d[0]	Output	PIN_L3
writeEnable_d	Input	PIN_N25

B.2 Reference Monitor Pinout



Node Name	Direction	Location
addr_issue_LED	Output	PIN_U18
address[3]	Input	PIN_V2
address[2]	Input	PIN_V1
address[1]	Input	PIN_U4
address[0]	Input	PIN_U3
clk	Input	PIN_W26
clkOut	Output	PIN_Y18
dataIn[1]	Input	PIN_T7
dataIn[0]	Input	PIN_P2
proc[1]	Input	PIN_P1
proc[0]	Input	PIN_N1
readEnable	Input	PIN_N26
sevenSegmentOut_a[6]	Output	PIN_T3
sevenSegmentOut_a[5]	Output	PIN_R6
sevenSegmentOut_a[4]	Output	PIN_R7
sevenSegmentOut_a[3]	Output	PIN_T4
sevenSegmentOut_a[2]	Output	PIN_U2
sevenSegmentOut_a[1]	Output	PIN_U1
sevenSegmentOut_a[0]	Output	PIN_U9
sevenSegmentOut_data[6]	Output	PIN_N9
sevenSegmentOut_data[5]	Output	PIN_P9
sevenSegmentOut_data[4]	Output	PIN_L7
sevenSegmentOut_data[3]	Output	PIN_L6
sevenSegmentOut_data[2]	Output	PIN_L9
sevenSegmentOut_data[1]	Output	PIN_L2
sevenSegmentOut_data[0]	Output	PIN_L3
sevenSegmentOut_p[6]	Output	PIN_R3
sevenSegmentOut_p[5]	Output	PIN_R4
sevenSegmentOut_p[4]	Output	PIN_R5
sevenSegmentOut_p[3]	Output	PIN_T9
sevenSegmentOut_p[2]	Output	PIN_P7
sevenSegmentOut_p[1]	Output	PIN_P6
sevenSegmentOut_p[0]	Output	PIN_T2
type_issue_LED	Output	PIN_U17
writeEnable	Input	PIN_N25

Appendix C

C.1 Access Control Logic Policy for Reference Monitor

Trap Op @ A

1. $Reference\ Monitor\ controls\ (User\ says\ \langle OP@A \rangle \supset RR\ says\ \langle (base, bound) \rangle \supset A + base \geq bound \supset \langle trap \rangle)$	*Jurisdiction*
2. $Reference\ Monitor \rightarrow ACL$	*Authorization*
3. $Reference\ Monitor\ says\ (User\ says\ \langle OP@A \rangle \supset RR\ says\ \langle (base, bound) \rangle \supset A + base \geq bound \supset \langle trap \rangle)$	*Request*
4. $User\ says\ \langle OP@A \rangle$	*Input statement*
5. $RR\ says\ \langle (base, bound) \rangle$	*Input statement*
6. $A + base \geq bound$	*Input statement*
7. $ACL\ says\ (User\ says\ \langle OP@A \rangle) \supset RR\ says\ \langle (base, bound) \rangle \supset A + base \geq bound \supset \langle trap \rangle)$	*Derived Speaks For* 2, 3
8. $User\ says\ \langle OP@A \rangle \supset RR\ says\ \langle (base, bound) \rangle \supset A + base \geq bound \supset \langle trap \rangle$	*Controls* 1, 7
9. $RR\ says\ \langle (base, bound) \rangle \supset A + base \geq bound \supset \langle trap \rangle$	*Modus Ponens* 4, 8
10. $A + base \geq bound \supset \langle trap \rangle$	*Modus Ponens* 5, 9
11. $\langle trap \rangle$	*Modus Ponens* 6, 10

Trap Op @ A

1. $Reference\ Monitor\ controls\ (User\ says\ \langle OP@A \rangle \supset RR\ says\ \langle (base, bound) \rangle \supset A \geq bound \supset \langle trap \rangle)$	*Jurisdiction*
2. $Reference\ Monitor \rightarrow ACL$	*Authorization*
3. $Reference\ Monitor\ says\ (User\ says\ \langle OP@A \rangle \supset RR\ says\ \langle (base, bound) \rangle \supset A \geq bound \supset \langle trap \rangle)$	*Request*
4. $User\ says\ \langle OP@A \rangle$	*Input statement*
5. $RR\ says\ \langle (base, bound) \rangle$	*Input statement*
6. $A \geq bound$	*Input statement*
7. $ACL\ says\ (User\ says\ \langle OP@A \rangle) \supset RR\ says\ \langle (base, bound) \rangle \supset A \geq bound \supset \langle trap \rangle)$	*Derived Speaks For* 2, 3
8. $User\ says\ \langle OP@A \rangle \supset RR\ says\ \langle (base, bound) \rangle \supset A \geq bound \supset \langle trap \rangle$	*Controls* 1, 7
9. $RR\ says\ \langle (base, bound) \rangle \supset A \geq bound \supset \langle trap \rangle$	*Modus Ponens* 4, 8
10. $A \geq bound \supset \langle trap \rangle$	*Modus Ponens* 5, 9
11. $\langle trap \rangle$	*Modus Ponens* 6, 10

Permit Op @ A

1. $ACL\ controls\ (RR\ says\ \langle(base, bound)\rangle$	$Jurisdiction$
$\supset A + base < q \supset A < bound \supset User\ controls\ \langle OP@A\rangle)$	
2. $Reference\ Monitor\ \rightarrow ACL$	$Authorization$
3. $Reference\ Monitor\ says\ (RR\ says\ \langle(base, bound)\rangle$	$Request$
$\supset A + base < q \supset A < bound \supset Usercontrols\langle OP@A\rangle)$	
4. $Reference\ Monitor\ says\ \langle OP@A\rangle$	$Input\ statement$
5. $RR\ says\ \langle(base, bound)\rangle$	$Input\ statement$
6. $A + base < q$	$Input\ statement$
7. $A < bound$	$Input\ statement$
8. $ACL\ says\ (RR\ says\ \langle(base, bound)\rangle$	$Derived\ Speaks\ For\ 2, 3$
$\supset A + base < q \supset A < bound \supset User\ controls\ \langle OP@A\rangle)$	
9. $RR\ says\ \langle(base, bound)\rangle$	$Controls\ 1, 8$
$\supset A + base < q \supset A < bound \supset User\ controls\ \langle OP@A\rangle$	
10. $A + base < q \supset A < bound \supset User\ controls\ \langle OP@A\rangle$	$Modus\ Ponens\ 5, 9$
11. $A < bound \supset User\ controls\ \langle OP@A\rangle$	$Modus\ Ponens\ 6, 10$
12. $User\ controls\ \langle OP@A\rangle$	$Modus\ Ponens\ 7, 11$
13. $\langle OP@A\rangle$	$Controls\ 12, 4$