This document contains the write-up for all four components of the Fall 2010 Final for ECE 57D.

**Part 1 – Embedded Design**

The goal of this portion of the Final Exam is to design a wait state insertion logic that performs a specific number of wait states depending on the corresponding and respective slave. The wait state insertion logic has a *request* input from the Master and an *init* input from each Slave. For the initialization, the *init* input is issued which leads the count input for the particular slave in the wait state insertion logic counter. During operation, the Master makes its request by asserting its *request* output, which is an input to the wait state insertion logic. While the wait states occur, the wait state insertion logic asserts its *wait\_request* output. This output remains asserted for the duration of the wait states and is de-asserted upon completion. At this point, the *request* signal is also de-asserted from the Master.

A high system level implementation of a **1 Master (M), 4 Slave (S) Embedded Design** is shown below. A few assumptions that will make this analysis more realistic with this design are as follows:

* Slave 0 = 0 CLK delay, Slave 1 = 1 CLK delay, Slave 2 = 2 CLK delay, Slave 3 = 3 CLK delay
* Only 1 *init(0-3)* signal will be asserted at a given time, and will stay high until logic is done



With the overall high level system now designed, it is time to implement the Wait State Insertion Logic (WSIL) that will drive all system inputs and outputs. The diagram below showcases the logic contained in the WSIL block located in the figure preceding this paragraph:



The following text below contains a description of this Wait State Insertion Logic, and how it operates:

**State:** Do-Nothing, Steady Operation

* + Since *request* = 0:
    - On each ***clk*** cycle (*posedge*): 2 Bit Up Counter Resets to 00
    - ***S\_MSB*** and ***S\_LSB*** (into 2 Bit Comparator) = 11
    - ***EQ*** output = 0, since ***C\_MSB***=***C\_LSB*** = 0, and ***S\_MSB*** *=* ***S\_LSB*** = 1
  + All ***initX*** values (X = 1 – 4) = 0

**State**: Wait State Transitions

* Since one of the ***initX*** values (X = 1 – 4) = 1:
  + ***M\_MSB*** and ***M\_LSB*** will = assigned wait state # of that specific slave into COMP
* Since ***request*** = 1:
  + AND Gate = 1, since one of the ***initX*** values (X = 1 – 4) = 1
  + 2-to-1 MUXs now pass through actual ***M\_MSB*** and ***M\_LSB*** to ***S\_MSB*** and ***S\_LSB***
  + 2 Bit Counter is enabled, stops clearing, and begins incrementing each ***clk***
  + ***wait\_request*** signal is asserted, and sent back to the Master

**State**: Wait State End, Pass Control Request, then Reset

* On final ***clk*** when {***C\_MSB***,***C\_LSB***} == {***S\_MSB***,***S\_LSB***}:
  + ***EQ*** = 1, and ***controlX*** output mapping to ***initX*** AND gate = 1
* Control is granted to Slave X after X ***clk*** cycles, in this specific system
* Once ***EQ*** = 1, ***wait\_request*** is de-asserted, and sent back to Master, which will then proceed to de-assert ***request*** signal (this logic not shown)

The hardware implementation (Datapath) above, coupled with the description of the state-machine layout (Controller), will yield a successful implementation of Wait State Insertion Logic for a Master – Slave circuit design.

This concludes the analysis for Final Exam – Part 1 (Embedded Design).

**Part 2 – C Coding**

The goal of this portion of the Final Exam is to design a 4\*4 array multiplier using C++ gate models with timing. A test bench will then be generated to simulate this circuit – different inputs will be used. Inputs will change at zero time in the test bench.

A diagram of the 4\*4 array multiplier, with each rectangle representing a multiplier cell in the ascending order logic is computed, is shown below:



Here, **x0-x3** and **y0-y3** are the input bits, and **p0-p7** are the output bits of the result, from LSB to MSB.

Assumptions and Details for the design of this circuit are as follows:

* AND gate delay = 2 (since 2 inputs)
* OR gate delay = 3 (since 3 inputs)
* XOR gate delay = 2 (since 2 inputs)
* X and Y input bit wires start with a delay of 1
* Worst case delay in AND and XOR are gates are computed by worst case wire.
* Worst case delay in OR, if w = 1, is computed by whichever input is seen as 1 first.

From a C++ code design perspective, *wires* will be used so that timing can be propagated through the circuit. In order to conserve space and optimize code, the multiplier cell logic will be implemented as its own class, with wires feeding to and from each multiplier cell. Code implementation follows below.

### Logic Definitions

The first two files contained in the project will define the logic interfaces necessary to implement the 4x4 Array Multiplier. All RT level components (AND, OR, XOR gates with timing) are defined here, and are then used to construct the multiplier cell logic. In addition, small test benches were created to validate all logic components as development was done. These are commented out but were left in the code for inspection purposes:

*logicPrimitives.h*

*logicPrimitives.cpp*

*logicPrimitives.h* defines the templates for each of the logic gate classes, along with the multiplier cell logic. This information is seen in the file below:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 2

#include <fstream>

#include <iostream>

#include <string>

using namespace std;

#define MAX2(a,b)((a < b) ? b : a);

// Define the wire class, which will hold:

// logic value and timing

class wire {

char value;

int delay;

public:

wire () {value = 'X'; delay = 0;}

wire (char v, int d) { value = v; delay = d;}

void put (char v, int d) {value = v; delay = d;}

void get (char& v, int& d) {v = value; d = delay;}

int delayValue() {return this->delay;}

char getVal() {return this->value;}

};

// Define a 2-input AND gate

class and {

wire i1, i2, o1;

int gateDelay;

public:

and() {gateDelay = 2;} // Gate Delay = 2, since there are 2 inputs.

int delayValue () {return this->gateDelay;}

void AND (wire, wire, wire&);

};

// Define a 3-input OR gate

class or {

wire i1, i2, i3, o1;

int gateDelay;

public:

or() {gateDelay = 3;} // Gate Delay = 6, since there are 3 inputs.

int delayValue() {return this->gateDelay;}

void OR (wire, wire, wire, wire&);

};

// Define a 2-input XOR gate

class xor {

wire i1, i2, o1;

int gateDelay;

public:

xor() {gateDelay = 2;} // Gate Delay = 2, since there are 2 inputs.

int delayValue () {return this->gateDelay;}

void XOR (wire, wire, wire&);

};

// Define an Array Multiplier cell

class multcell {

public:

void perform\_mult(wire, wire, wire, wire, wire&, wire&);

};

The corresponding file is *logicPrimitives.cpp*, which implements all functions described above in the header file. Detailed comments are written within each function to explain the operation, and are all based on the *wire* implementation denoted above.

// Carlos Lazo

// ECE579D

// Final Exam - Problem 2

#include "logicPrimitives.h"

// Define the AND function

void and::AND(wire i1, wire i2, wire &o1)

{

char av, bv, wv;

int ad, bd, wd;

i1.get(av, ad);

i2.get(bv, bd);

// Perform gate level operations.

if ((av == '0') || (bv == '0'))

{

wv = '0';

// Output delay = Max(i1 delay, i2 delay) + gate delay

wd = this->gateDelay + MAX2(ad,bd);

}

else

{

wv = '1';

// Output delay = Max(i1 delay, i2 delay) + gate delay

wd = this->gateDelay + MAX2(ad,bd);

}

o1.put(wv,wd);

}

/\*

Test for AND Gate in Main Program:

wire w\_t1('1',3); wire w\_t2('1',2); wire w\_o;

and g\_t;

g\_t.AND(w\_t1,w\_t2,w\_o);

cout << "Output value = " << w\_o.getVal() << ", and Delay = " << w\_o.delayValue() << endl << endl;

\*/

// Define the OR function

void or::OR(wire i1, wire i2, wire i3, wire& o1)

{

char av, bv, cv, wv;

int ad, bd, cd, wd;

i1.get(av, ad);

i2.get(bv, bd);

i3.get(cv, cd);

if ((av == '1')||(bv == '1')||(cv == '1'))

{

wv = '1'; // Value = 1

// Output delay = Max(i1 delay, i2 delay, i3 delay) + gate delay

wd = MAX2(ad,bd);

wd = this->gateDelay + MAX2(wd,cd);

}

else

{

wv = '0'; // Value = 0

// Output delay = Max(i1 delay, i2 delay, i3 delay) + gate delay

wd = MAX2(ad,bd);

wd = this->gateDelay + MAX2(wd,cd);

}

o1.put(wv, wd); // Place calculated value on output wire

}

/\*

Test for OR Gate in Main Program:

wire w\_t1('0',1); wire w\_t2('0',2); wire w\_t3('0',8); wire w\_o;

or g\_t;

g\_t.OR(w\_t1,w\_t2,w\_t3,w\_o);

cout << "Output value = " << w\_o.getVal() << ", and Delay = " << w\_o.delayValue() << endl << endl;

\*/

// Define the XOR function

void xor::XOR(wire i1, wire i2, wire &o1)

{

char av, bv, wv;

int ad, bd, wd;

i1.get(av, ad);

i2.get(bv, bd);

// Perform gate level operations.

if (((av == '0') && (bv == '0')) || ((av == '1') && (bv == '1')))

{

wv = '0';

// Output gate delay should be worst case from either gate.

wd = this->gateDelay + MAX2(ad,bd);

}

else

{

wv = '1';

// Output delay = Max(i1 delay, i2 delay) + gate delay

wd = this->gateDelay + MAX2(ad,bd);

}

o1.put(wv,wd);

}

/\*

Test for XOR Gate in Main Program:

wire w\_t1('0',5); wire w\_t2('1',3); wire w\_o;

xor g\_t;

g\_t.XOR(w\_t1,w\_t2,w\_o);

cout << "Output value = " << w\_o.getVal() << ", and Delay = " << w\_o.delayValue() << endl << endl;

\*/

// Define the gate level logic for the Multiplier Cell

void multcell::perform\_mult(wire xi, wire yi, wire ci, wire pi, wire &co, wire &po)

{

// Define all necessary wires for this operation:

wire w\_A1, w\_A2, w\_A3, w\_A4; // Wires for AND gate outputs.

wire w\_X1; // Wires for XOR gate outputs.

// Define all necessary gates for this operation:

and g\_A1, g\_A2, g\_A3, g\_A4; // AND gates.

or g\_O1; // OR gate.

xor g\_X1, g\_X2; // XOR gate.

// Perform all associated logic and connections with multiplier cell:

g\_A1.AND(xi,yi,w\_A1);

g\_A2.AND(pi,w\_A1,w\_A2);

g\_A3.AND(w\_A1,ci,w\_A3);

g\_A4.AND(ci,pi,w\_A4);

g\_O1.OR(w\_A2,w\_A3,w\_A4,co);

g\_X1.XOR(pi,w\_A1,w\_X1);

g\_X2.XOR(w\_X1,ci,po);

}

/\*

Test for MultCell logic in Main Program:

wire xi ('1',3);

wire yi ('1',1);

wire ci ('0',1);

wire pi ('0',1);

wire co; wire po;

multcell MC1;

MC1.perform\_mult(xi,yi,ci,pi,co,po);

cout << "po = " << po.getVal() << ", with delay = " << po.delayValue() << endl;

cout << "co = " << co.getVal() << ", with delay = " << co.delayValue() << endl;

\*/

### Array Multiplier Definition

In order to easily encapsulate the data necessary for the Array Multiplier, two separate files were created to contain all necessary data and logic:

*arrayMult.h*

*arrayMult.cpp*

*arrayMult.h* is the file that defines the *arrayMult* class, and declares all templates for implementation in the respective .cpp file to follow. This class loads in input from two strings via the *load\_input* function, performs the multiplication via *MULT*, and outputs to the terminal with *output\_result*:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 2

#include "logicPrimitives.h"

// Define the arrayMult Class

class arrayMult {

private:

// Declare all private member variables of the 4x4 Array Multiplier:

wire x\_in [4]; // Input X wires.

wire y\_in [4]; // Input Y wires.

wire po [8]; // Declare output wires.

string x\_str, y\_str, result;

public:

void load\_input (string x, string y); // Load input wires.

void MULT(); // Perform array multiplication.

void output\_result (); // Output multiplication result.

};

*arrayMult.cpp* generates the logic that is needed to implement the Array Multiplier. The *multcell* class is used here, in conjunction with the System Level Diagram presented at the beginning of this particular analysis. Multiple *wires* with a simple naming convention were drawn between all Multiplier Cells (MCs), and can be traced back to the System Level Diagram:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 2

#include "arrayMult.h"

// Load in X and Y input strings, and assume delay = 1 for each.

void arrayMult::load\_input(string x, string y)

{

// Since string[0] = MSB, ensure wire array is setup in ascending order.

x\_in[3].put(x[0],1);

x\_in[2].put(x[1],1);

x\_in[1].put(x[2],1);

x\_in[0].put(x[3],1);

y\_in[3].put(y[0],1);

y\_in[2].put(y[1],1);

y\_in[1].put(y[2],1);

y\_in[0].put(y[3],1);

x\_str = x;

y\_str = y;

/\*

Test for input values:

for (int i = 0; i < 4; i++)

cout << x\_in[3-i].getVal();

cout << endl;

for (int i = 0; i < 4; i++)

cout << y\_in[3-i].getVal();

\*/

}

// Generate logic for 4x4 Multiplier array, using multcell class.

void arrayMult::MULT()

{

// Create an ZERO wire for initial values -

wire ZERO('0',1);

// Multiplier Cell #01

multcell M01;

wire w\_M01\_co;

M01.perform\_mult(x\_in[0],y\_in[0],ZERO,ZERO,w\_M01\_co,po[0]);

// Multiplier Cell #02

multcell M02;

wire w\_M02\_co; wire w\_M02\_po;

M02.perform\_mult(x\_in[0],y\_in[1],w\_M01\_co,ZERO,w\_M02\_co,w\_M02\_po);

// Multiplier Cell #03

multcell M03;

wire w\_M03\_co; wire w\_M03\_po;

M03.perform\_mult(x\_in[0],y\_in[2],w\_M02\_co,ZERO,w\_M03\_co,w\_M03\_po);

// Multiplier Cell #04

multcell M04;

wire w\_M04\_co; wire w\_M04\_po;

M04.perform\_mult(x\_in[0],y\_in[3],w\_M03\_co,ZERO,w\_M04\_co,w\_M04\_po);

// Multiplier Cell #05

multcell M05;

wire w\_M05\_co;

M05.perform\_mult(x\_in[1],y\_in[0],ZERO,w\_M02\_po,w\_M05\_co,po[1]);

// Multiplier Cell #06

multcell M06;

wire w\_M06\_co; wire w\_M06\_po;

M06.perform\_mult(x\_in[1],y\_in[1],w\_M05\_co,w\_M03\_po,w\_M06\_co,w\_M06\_po);

// Multiplier Cell #07

multcell M07;

wire w\_M07\_co; wire w\_M07\_po;

M07.perform\_mult(x\_in[1],y\_in[2],w\_M06\_co,w\_M04\_po,w\_M07\_co,w\_M07\_po);

// Multiplier Cell #08

multcell M08;

wire w\_M08\_co; wire w\_M08\_po;

M08.perform\_mult(x\_in[1],y\_in[3],w\_M07\_co,ZERO,w\_M08\_co,w\_M08\_po);

// Multiplier Cell #09

multcell M09;

wire w\_M09\_co;

M09.perform\_mult(x\_in[2],y\_in[0],ZERO,w\_M06\_po,w\_M09\_co,po[2]);

// Multiplier Cell #10

multcell M10;

wire w\_M10\_co; wire w\_M10\_po;

M10.perform\_mult(x\_in[2],y\_in[1],w\_M09\_co,w\_M07\_po,w\_M10\_co,w\_M10\_po);

// Multiplier Cell #11

multcell M11;

wire w\_M11\_co; wire w\_M11\_po;

M11.perform\_mult(x\_in[2],y\_in[2],w\_M10\_co,w\_M08\_po,w\_M11\_co,w\_M11\_po);

// Multiplier Cell #12

multcell M12;

wire w\_M12\_co; wire w\_M12\_po;

M12.perform\_mult(x\_in[2],y\_in[3],w\_M11\_co,ZERO,w\_M12\_co,w\_M12\_po);

// Multiplier Cell #13

multcell M13;

wire w\_M13\_co;

M13.perform\_mult(x\_in[3],y\_in[0],ZERO,w\_M10\_po,w\_M13\_co,po[3]);

// Multiplier Cell #14

multcell M14;

wire w\_M14\_co;

M14.perform\_mult(x\_in[3],y\_in[1],w\_M13\_co,w\_M11\_po,w\_M14\_co,po[4]);

// Multiplier Cell #15

multcell M15;

wire w\_M15\_co;

M15.perform\_mult(x\_in[3],y\_in[2],w\_M14\_co,w\_M12\_po,w\_M15\_co,po[5]);

// Multiplier Cell #16

multcell M16;

M16.perform\_mult(x\_in[3],y\_in[3],w\_M15\_co,ZERO,po[7],po[6]);

/\*

// See worst case delay path:

cout << "Worst Case Delay Path:\n\n";

cout << "\tMC01 - cout = " << w\_M01\_co.delayValue() << endl;

cout << "\tMC02 - cout = " << w\_M02\_co.delayValue() << endl;

cout << "\tMC03 - cout = " << w\_M03\_co.delayValue() << endl;

cout << "\tMC04 - pout = " << w\_M04\_po.delayValue() << endl;

cout << "\tMC07 - cout = " << w\_M07\_co.delayValue() << endl;

cout << "\tMC08 - pout = " << w\_M08\_po.delayValue() << endl;

cout << "\t\tMC10 - cout = " << w\_M10\_co.delayValue() << endl;

cout << "\tMC11 - cout = " << w\_M11\_co.delayValue() << endl;

cout << "\tMC12 - pout = " << w\_M12\_po.delayValue() << endl;

cout << "\t\tMC14 - cout = " << w\_M14\_co.delayValue() << endl;

cout << "\tMC15 - cout = " << w\_M15\_co.delayValue() << endl;

cout << "\tMC16 - cout = " << po[7].delayValue() << endl;

\*/

}

void arrayMult::output\_result()

{

// First, look at output wires to determine maximum delay:

int maxDelay = 0;

for (int i = 0; i < 8; i++) {

if (po[i].delayValue() > maxDelay){

maxDelay = po[i].delayValue();

// cout << "Delay Value @ p" << i << "is: " << po[i].delayValue() << endl;

}}

// Second, create output string:

result = "";

for (int i = 0; i < 8; i++)

result = result + po[7-i].getVal();

// Print out results to terminal window:

cout << "Multiplication of X = " << x\_str << " and Y = " << y\_str << " is: " << result << endl;

cout << "\tDelay Value for this operation = " << maxDelay << endl << endl;

}

### Simple Testbench

A simple testbench is generated in the *arrayMultiplierTB.cpp* file, which houses the *main* function the project. An Array Multiplier instance is created and a total of 5 operations are performed, all of which call the *output\_result* function of the *arrayMult* defined in this testbench:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 2

#include "arrayMult.h"

int main ()

{

arrayMult AM;

AM.load\_input("0001", "0001");

AM.MULT();

AM.output\_result();

AM.load\_input("0010", "0010");

AM.MULT();

AM.output\_result();

AM.load\_input("0111", "0000");

AM.MULT();

AM.output\_result();

AM.load\_input("0111", "0111");

AM.MULT();

AM.output\_result();

AM.load\_input("1111", "0111");

AM.MULT();

AM.output\_result();

// Freeze screen output.

int freeze;

cin >> freeze;

return 0;

}

Command line output from the above test bench yields the following correct results, which in decimal map to:

1 x 1 = 1, 2 x 2 = 4, 7 x 0 = 0, 7 \* 7 = 49, and 15 x 7 = 105

***Testbench – Command Line Output***

Multiplication of X = 0001 and Y = 0001 is: 00000001

Delay Value for this operation = 48

Multiplication of X = 0010 and Y = 0010 is: 00000100

Delay Value for this operation = 48

Multiplication of X = 0111 and Y = 0000 is: 00000000

Delay Value for this operation = 48

Multiplication of X = 0111 and Y = 0111 is: 00110001

Delay Value for this operation = 48

Multiplication of X = 1111 and Y = 0111 is: 01101001

Delay Value for this operation = 48

### Worst Case Delay

The worst case delay is measured at all of the p[0-7] {output} wires at the end of the circuit. These lines of code can be seen commented out in the *MULT()* function contained in *arrayMult.cpp*. Output which maps exactly to the worst case delay path can be seen below, as computed by the individual wires. It is important to note that in 2 cases in the circuit, delays coming from another cell are greater than those in the original longest path.

***Worst Case Delay – Command Line Output***

Worst Case Delay Path:

MC01 - cout = 8

MC02 - cout = 13

MC03 - cout = 18

MC04 - pout = 20

MC07 - cout = 25

MC08 - pout = 27

MC10 - cout = 29

MC11 - cout = 34

MC12 - pout = 36

MC14 - cout = 38

MC15 - cout = 43

MC16 - cout = 48

To visually see the worst case delay, the original System Level Diagram was utilized. The **purple lines** map to the physical worst case delay of the circuit, and the **purple numbers** represent the summation of the worst case delay across the Array Multiplier. In the end, it can be easily seen that with my circuit implementation and gate delay assignments (shown **in purple** on the multiplier cell at the top left of the diagram, **48** is the correct worst case circuit delay:



This concludes the analysis for Final Exam – Part 2 (C Coding).

**Part 3 – SystemC Channels**

The goal of this portion of the Final Exam is to model the arbitration of a shared device between several Master modules using a *sc\_semaphore*, a primitive channel defined in SystemC. In this specific system, a total of 5 Master modules will be attempting to access the device at random times for random amounts of time. The *sc\_semaphore* will be defined as a size of 4. All 5 Master modules will be connected with the *sc\_semaphore*, and will all be trying to access the device at the same time. Each master module will print the time it requests the use of the device, report that it was granted access, and report the time it releases the device. At given times, there will be instances where all Master’s try to access the device within the same time frame, at which point the last device will be denied access. All printing will be written to an external activity file.

The code for this portion of the Final Exam is all contained to one file, called *main.*cpp. This file defines an SC\_MODULE, called *Master*, which takes in two inputs – the “name” of the instance and a unique identifier, which will be used for printing. An *sc\_semaphore bus* is defined, which will be utilized by all *Master* modules. A timing SC\_MODULE, called *semaphore\_access*, is also used to terminate the simulation after t = 100. The code for *main.cpp* is contained below, and is filled with detailed comments that will showcase the functionality of this system design:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 3

#include "systemc"

#include "time.h"

#include "sys/timeb.h"

#include "common\_header.h"

using namespace sc\_core;

// Declare a common semaphore, called bus, with 4 tokens

sc\_semaphore bus(4);

// Define the Master module:

SC\_MODULE (Master) {

// Define private member variables of module:

sc\_in <bool> clock; // External clock provided to module

int id; // Unique identifier of Master module

void MASTER\_REQ()

{

while(true)

{

int t1\_rand = rand()% 5 + 1; // Random # from 1 - 5.

// Wait time prior to issuing request.

int t2\_rand = rand()% 10 + 1; // Random # from 1 - 10.

// If request granted, wait time before releasing semaphore.

wait(t1\_rand);

// Print time when Master requests access to device after t1\_rand seconds.

fout << "\n@" << sc\_time\_stamp() << "\tMASTER " << id << " is requesting access to device";

fout << " after waiting << " << t1\_rand << " seconds." << endl;

// Attempt to obtain semaphore, if available (non-blocking).

if (bus.trywait() != -1)

{

// Print time that Master will need device, and then wait t2\_rand seconds.

fout << "\t\* MASTER " << id << " granted access to device and needs it for ";

fout << t2\_rand << " seconds." << endl;

wait (t2\_rand);

// Return semaphore to device.

bus.post();

// Print time that Master has released semaphore back to device.

fout << "\n@" << sc\_time\_stamp() << "\tMASTER " << id << " releases semaphore back to device.";

fout << " Count = " << bus.get\_value();

}

// If obtaining the semaphore was unsuccessful, then print so.

else

fout << "\t!!! MASTER " << id << " denied access to device. !!!";

fout << endl;

}

}

// Define Master constructor:

Master(sc\_module\_name \_n, int \_id) : sc\_module(\_n), id(\_id)

{

SC\_CTHREAD(MASTER\_REQ,clock.pos());

}

SC\_HAS\_PROCESS(Master); // Required if processes are used but SC\_CTOR is not called

};

// Define the semaphore\_access module, which will be used as a simple timer.

SC\_MODULE (semaphore\_access) {

// Define inputs of the module:

sc\_in<bool> clock;

// Define local variables of the module:

int count;

// Stop the Clock after t = 100 seconds

void timer()

{

while(true)

{

wait();

count++;

if (count > 100)

sc\_stop();

}

}

SC\_CTOR(semaphore\_access)

{

count = 0;

SC\_CTHREAD(timer,clock.pos());

}

};

// MAIN PROGRAM

int sc\_main (int argc, char\* argv[]) {

// Initialize a random seed for timing:

srand ( time(NULL) );

// Initialize semaphore\_access module & common clock:

sc\_clock clock ("my\_clock",1,0.5);

semaphore\_access SA ("SA");

SA.clock(clock.signal());

// Initialize all 5 Master modules:

Master M1("M1",1);

M1.clock(clock);

Master M2("M2",2);

M2.clock(clock);

Master M3("M3",3);

M3.clock(clock);

Master M4("M4",4);

M4.clock(clock);

Master M5("M5",5);

M5.clock(clock);

cout << endl;

sc\_start(0); // First time called will init schedular

sc\_start(); // Run the simulation till sc\_stop is encountered

int freeze;

cin >> freeze;

return 0;// Terminate simulation

}

As stated in the problem description, the print statements in the *Master* SC\_MODULE are all reported to an external file, called activity\_report.txt – only beginning portion of this file will be showcased to highlight activity between the *Master* modules and the *bus* semaphore. Particular colored statements will be explained after this information, which specifically highlight the correct behavior of this program:

***Activity File Print Statements***

**@2 ns MASTER 5 is requesting access to device after waiting << 2 seconds.**

**\* MASTER 5 granted access to device and needs it for 10 seconds.**

**@3 ns MASTER 3 is requesting access to device after waiting << 3 seconds.**

**\* MASTER 3 granted access to device and needs it for 3 seconds.**

**@4 ns MASTER 1 is requesting access to device after waiting << 4 seconds.**

**\* MASTER 1 granted access to device and needs it for 8 seconds.**

**@5 ns MASTER 2 is requesting access to device after waiting << 5 seconds.**

**\* MASTER 2 granted access to device and needs it for 9 seconds.**

**@5 ns MASTER 4 is requesting access to device after waiting << 5 seconds.**

**!!! MASTER 4 denied access to device. !!!**

@6 ns MASTER 3 releases semaphore back to device. Count = 1

@8 ns MASTER 4 is requesting access to device after waiting << 3 seconds.

\* MASTER 4 granted access to device and needs it for 8 seconds.

@9 ns MASTER 3 is requesting access to device after waiting << 3 seconds.

!!! MASTER 3 denied access to device. !!!

**@12 ns MASTER 1 releases semaphore back to device. Count = 1**

@12 ns MASTER 5 releases semaphore back to device. Count = 2

@13 ns MASTER 5 is requesting access to device after waiting << 1 seconds.

\* MASTER 5 granted access to device and needs it for 4 seconds.

@14 ns MASTER 2 releases semaphore back to device. Count = 2

@14 ns MASTER 3 is requesting access to device after waiting << 5 seconds.

\* MASTER 3 granted access to device and needs it for 8 seconds.

@16 ns MASTER 1 is requesting access to device after waiting << 4 seconds.

\* MASTER 1 granted access to device and needs it for 9 seconds.

@16 ns MASTER 4 releases semaphore back to device. Count = 1

@17 ns MASTER 2 is requesting access to device after waiting << 3 seconds.

\* MASTER 2 granted access to device and needs it for 3 seconds.

@17 ns MASTER 5 releases semaphore back to device. Count = 1

@19 ns MASTER 4 is requesting access to device after waiting << 3 seconds.

\* MASTER 4 granted access to device and needs it for 2 seconds.

@20 ns MASTER 2 releases semaphore back to device. Count = 1

@20 ns MASTER 5 is requesting access to device after waiting << 3 seconds.

\* MASTER 5 granted access to device and needs it for 7 seconds.

@21 ns MASTER 2 is requesting access to device after waiting << 1 seconds.

!!! MASTER 2 denied access to device. !!!

…

**Green**: After 2 sec, Master 5 requests semaphore for 10 sec. Granted, and released back @ 12 sec.

**Blue**: Three other devices (Masters 3, 1, and 2) request semaphore and granted. *Count = 0*.

**Red**: Master 4 requests semaphore, but since *Count = 0*, request is DENIED.

As can be seen in this portion of the *activity\_file.txt* output, all information is being reported correctly, timing between unique semaphore request and release is correct, and non-blocking semaphore logic is being denied when not available.

This concludes the analysis for Final Exam – Part 3 (SystemC Channels).

**Part 4 – TLM-2.0**

The goal of this portion of the Final Exam is design a system that reads address and data information from an input file, routes it based on the parity of the address to specific memories, and reads that information to external memory when requested. TLM 2.0 channels are to be used for this design.

The overall layout of the system follows the structure below, with A = *Reader*, Interconnect = *Switch*, and the two *Memory* segments:



### Code Analysis

The code will be presented sequentially from left to write. A different instantiation of blocking *b\_transport* is defined in all 3 primary components of the design. Once all definitions are established, *main.cpp* establishes a top level SC\_MODULE which instantiates all components, and connects are Initiator and Target sockets appropriately.

The first file is *common\_header.h*, and serves to define the primary STL libraries and variables that will be used throughout the design:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 4

#ifndef \_\_COMMON\_HEADER\_H\_\_

#define \_\_COMMON\_HEADER\_H\_\_

#define SC\_INCLUDE\_DYNAMIC\_PROCESSES

#include "systemc"

using namespace sc\_core;

using namespace sc\_dt;

using namespace std;

#include "tlm.h"

#include "tlm\_utils/simple\_initiator\_socket.h"

#include "tlm\_utils/simple\_target\_socket.h"

#include "tlm\_utils/passthrough\_target\_socket.h"

#include "tlm\_utils/multi\_passthrough\_initiator\_socket.h"

#include "tlm\_utils/multi\_passthrough\_target\_socket.h"

#include "tlm\_utils/peq\_with\_cb\_and\_phase.h"

#include "tlm\_utils/instance\_specific\_extensions.h"

#include <iomanip>

#include <deque>

// Added in iostream for C++ terminal debug output.

#include <iostream>

// Added in fstream to allow for external file read/write.

#include <fstream>

// Added in to make file-reading easier.

#include <bitset>

#include <string>

#include <sstream>

// Added in to facilitate string rotations

#include <algorithm>

#include <vector>

// Allows for viewing of commands that access memory (R/W)

static fstream in\_data("in.txt");

#endif

The *Reader* of the program, located in *reader.cpp*, is what is responsible for taking in information from the *in.txt* file and send it to the *Switch*, which will route information to a respective *Memory* based on whether the address is even or odd. Detailed comments in the code provide exactly how the SC\_THREAD is instantiated, which continues until input is fully exhausted. String parsing is done here, and the *TLM\_READ\_COMMAND* and *TLM\_WRITE\_COMMAND* is set here, which will be used by the respective *Memory* to determine what operation is to be performed:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 4

#define SC\_INCLUDE\_DYNAMIC\_PROCESSES

#include "systemc"

#include "common\_header.h"

using namespace sc\_core;

using namespace std;

struct Reader : sc\_module

{

tlm\_utils::simple\_initiator\_socket<Reader> socket;

SC\_CTOR(Reader) : socket("readerSocket")

{

SC\_THREAD(thread\_process);

}

void thread\_process()

{

cout << " - Initiating Reader thread process -\n\n";

tlm::tlm\_generic\_payload\* trans = new tlm::tlm\_generic\_payload;

tlm::tlm\_command cmd;

sc\_time delay = sc\_time(8, SC\_PS);

string in\_buf; // To be used for initial data input.

string a\_str; // Address string.

string d\_str; // Data string.

int a\_int; // Integer form of address.

int d\_int; // Integer form of data;

// Continue while loop for as long as information can be read in.

while (getline(in\_data,in\_buf))

{

cout << "\* Instruction read in = " << in\_buf << "\n\n";

// If in\_buf[0] == R, then this is a READ command.

// If not, then it is a WRITE command.

if (in\_buf[0] == 'R')

{

cout << "\tIn Reader thread: This is a READ command.\n";

cmd = tlm::TLM\_READ\_COMMAND;

}

else

{

cout << "\tIn Reader thread: This is a WRITE command.\n";

cmd = tlm::TLM\_WRITE\_COMMAND;

}

// Define address and data strings based on command.

a\_str = in\_buf.substr(2,2);

if (in\_buf[0] == 'R')

d\_str = "";

else

d\_str = in\_buf.substr(5,in\_buf.size()-5);

cout << "\t\ta\_str = " << a\_str << ", and d\_str = " << d\_str << endl;

// Now convert strings into their integer representations.

sscanf(a\_str.c\_str(), "%x", &a\_int);

sscanf(d\_str.c\_str(), "%x", &d\_int);

cout << "\t\ta\_int = " << a\_int << ", and d\_int = " << d\_int << endl;

// Setup b\_transport package and send to Switch.

trans->set\_command(cmd);

trans->set\_address(a\_int \* 16);

trans->set\_data\_ptr( reinterpret\_cast<unsigned char\*>(&d\_int) );

// Modify data\_length to 16 bits

trans->set\_data\_length( 16 );

// Modify data\_streaming\_width to 16 bits

trans->set\_streaming\_width( 16 ); // = data\_length to indicate no streaming

trans->set\_byte\_enable\_ptr( 0 ); // 0 indicates unused

trans->set\_dmi\_allowed( false ); // Mandatory initial value

trans->set\_response\_status( tlm::TLM\_INCOMPLETE\_RESPONSE ); // Mandatory initial value

// Send information to the Switch via b\_transport:

socket->b\_transport(\*trans, delay ); // Blocking transport call.

}

}

};

The *Switch* of the program is what is responsible for taking in information from the *Reader* and parses it out to the correct *Memory*. It is important to note that there is one target port here (from the *Reader*), and two Initiator ports (two both *Memory* modules). The respective *b\_transport* of each socket is mapped by the *Switch* constructor, which will help dictate where the information is routed. Parity is established here in *b\_transport*, and the respective initiator socket’s *b\_transport* is called accordingly:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 4

#define SC\_INCLUDE\_DYNAMIC\_PROCESSES

#include "common\_header.h"

template<unsigned int N\_TARGETS, unsigned int N\_INITIATORS>

struct Switch:sc\_module

{

tlm\_utils::simple\_target\_socket\_tagged<Switch>\* targ\_socket[N\_INITIATORS];

tlm\_utils::simple\_initiator\_socket\_tagged<Switch>\* init\_socket[N\_TARGETS];

SC\_CTOR(Switch)

{

// Set the number of incoming socket connections (= 1)

for (unsigned int i = 0; i < N\_INITIATORS; i++)

{

char txt[20];

sprintf(txt, "targ\_socket\_%d", i);

targ\_socket[i] = new tlm\_utils::simple\_target\_socket\_tagged<Switch>(txt);

targ\_socket[i]->register\_b\_transport( this, &Switch::b\_transport, i);

}

// Set the number of outgoing socket connections (= 2)

for (unsigned int i = 0; i < N\_TARGETS; i++)

{

char txt[20];

sprintf(txt, "init\_socket\_%d", i);

init\_socket[i] = new tlm\_utils::simple\_initiator\_socket\_tagged<Switch>(txt);

}

}

// Define an overloaded b\_transport that takes ID into account for the Reader.

virtual void b\_transport( int id, tlm::tlm\_generic\_payload& trans, sc\_time& delay ){

int target\_nr = 0;

sc\_dt::uint64 address = trans.get\_address();

// If address is ODD, go to Memory1. Else, Memory2.

if ((address / 16) % 2 == 1)

target\_nr = 0;

else

target\_nr = 1;

// If ID is defined as Reader (ID = 0), select which memory socket chosen.

if (id == 0)

{

cout << "\tIn Switch b\_transport. Target memory chosen is Memory" << (target\_nr + 1) << endl;

(\*init\_socket[target\_nr])->b\_transport(trans, delay);

}

}

};

The *Memory* of the system, defined in *memory.cpp*, receives the correct information from the *Switch* and performs the correct operation. The *Switch* took care of parity already, so information is collected from the *b\_transport* package that was received, reformatted, and either placed into the memory (if a WRITE command) or written into the correct external file (if a READ command). The output streams are established in the constructor of the *Memory* module using template input:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 4

#define SC\_INCLUDE\_DYNAMIC\_PROCESSES

#include "common\_header.h"

template<unsigned int id>

struct Memory: sc\_module

{

tlm\_utils::simple\_target\_socket<Memory> socket;

enum { SIZE = 4096 }; // Increase memory size to 4096 elements

const sc\_time LATENCY;

SC\_CTOR(Memory)

: socket("socket"), LATENCY(0, SC\_PS)

{

socket.register\_b\_transport( this, &Memory::b\_transport);

for (int i = 0; i < SIZE; i++)

mem[i] = 0;

// Link memory instance to an output file.

out\_id = id;

if (out\_id == 1)

out.open("Out1.txt");

if (out\_id == 2)

out.open("Out2.txt");

}

virtual void b\_transport( tlm::tlm\_generic\_payload& trans, sc\_time& delay )

{

tlm::tlm\_command cmd = trans.get\_command();

unsigned int adr = trans.get\_address(); // Change since now 8-bit words.

unsigned char\* ptr = trans.get\_data\_ptr();

unsigned int len = trans.get\_data\_length();

unsigned char\* byt = trans.get\_byte\_enable\_ptr();

unsigned int wid = trans.get\_streaming\_width();

delay = SC\_ZERO\_TIME;

// If TLM Command == READ, output information @ mem[adr] to external file.

// Choose the correct external file.

if ( cmd == tlm::TLM\_READ\_COMMAND )

{

cout << "\tIn Memory b\_transport. A READ operation is being performed.\n";

unsigned int a\_int = adr / 16;

unsigned int d\_int = mem[adr];

cout << "\t\ta\_int = " << a\_int << ", and d\_int = " << d\_int << endl;

char a\_buf [32];

char d\_buf [32];

itoa(a\_int,a\_buf,16);

itoa(d\_int,d\_buf,16);

string a\_str(a\_buf);

string d\_str(d\_buf);

// If address/data is a single memory element, append 0 to the front.

if (a\_str.size() == 1)

a\_str = "0" + a\_str;

if (d\_str.size() == 1)

d\_str = "0" + d\_str;

cout << "\t\ta\_str = " << a\_str << ", and d\_str = " << d\_str << endl;

// Output to respective memory element based on address parity:

if (out\_id == 1)

{

cout << "\tOutput address and data info to Out1.txt\n\n";

out << a\_str << " " << d\_str << endl;

}

if (out\_id == 2)

{

cout << "\tOutput address and data info to Out2.txt\n\n";

out << a\_str << " " << d\_str << endl;

}

}

else if ( cmd == tlm::TLM\_WRITE\_COMMAND )

{

cout << "\tIn Memory b\_transport. A WRITE operation is being performed.\n\n";

memcpy(&mem[adr], ptr, len);

}

}

int mem[SIZE];

int out\_id;

ofstream out;

};

Last is the *main.cpp* file, which instantiates all of the classes above in their respective configurations. It is important to note that two *Memory* modules are created, each with unique ID’s for output file differentiation. Also, all socket mappings are established in the constructor of the *Top* SC\_MODULE:

// Carlos Lazo

// ECE579D

// Final Exam - Problem 4

#include "common\_header.h"

#include "systemc"

#include "time.h"

#include "sys/timeb.h"

#include "reader.cpp"

#include "switch.cpp"

#include "memory.cpp"

using namespace sc\_core;

SC\_MODULE(Top)

{

// Initialize all instances for system:

Reader\* reader;

Switch<2,1>\* router;

Memory<1>\* memory1;

Memory<2>\* memory2;

SC\_CTOR(Top)

{

reader = new Reader("reader");

router = new Switch<2,1>("router");

memory1 = new Memory<1>("memory1");

memory2 = new Memory<2>("memory2");

// Bind Writer initiator socket with Switch target socket

router->targ\_socket[0]->bind(reader->socket);

// Bind Switch initiator sockets with Memory target sockets

(\*(router->init\_socket[0]) ).bind( memory1->socket ); //bind memory1 socket

(\*(router->init\_socket[1]) ).bind( memory2->socket ); //bind memory2 socket

}

};

int sc\_main(int argc, char\* argv[])

{

Top top("top");

sc\_start();

cout << "DONE!\n\n";

int freeze;

cin >> freeze;

return 0;

}

### Output Validation and Verification

Upon compiling and running the program, given the following in.txt file, the contents in the resulting Out1.txt and Out2.txt files was fully correct. A coloring scheme will be used to identify **odd** file placement in **Out1.txt** and **even** file placement in **Out2.txt** – it’s also important to note that the parity of the 1st bit will determine the overall parity of the number. Arrows in the in.txt file indicate the things that should be written into the different output files in chronological order:

***“in.txt” Input File***

**W** 00 10

**W** 01 20

**W** 02 30

**W** 10 40

**W** 12 50

**R** 01 🡪 **Write 01 20 to Out1.txt (Line 1)**

**R** 10 🡪 **Write 10 40 to Out2.txt (Line 1)**

**W** 01 120

**W** 11 03

**R** 01 🡪 **Write 01 120 to Out1.txt (Line 2)**

**W** 31 08

**R** 00 🡪 **Write 00 10 to Out2.txt (Line 2)**

**R** 01 🡪 **Write 01 120 to Out1.txt (Line 3)**

**R** 02 🡪 **Write 02 30 to Out2.txt (Line 3)**

**R** 10 🡪 **Write 10 40 to Out2.txt (Line 4)**

**R** 11 🡪 **Write 11 03 to Out1.txt (Line 4)**

**R** 12 🡪 **Write 12 50 to Out2.txt (Line 5)**

**R** 31 🡪 **Write 31 08 to Out1.txt (Line 5)**

***“Out1.txt” Contents (post-run)***

0**1** 20

0**1** 120

0**1** 120

1**1** 03

3**1** 08

***“Out2.txt” Contents (post-run)***

1**0** 40

0**0** 10

0**2** 30

1**0** 40

1**2** 50

As seen above, all output is written **in the correct order** to the Out1.txt and Out2.txt files.

In addition, in order to verify and validate progress of this project, *cout* statements were used to allow for easy analysis and to confer with the resulting output file information above. This is seen below:

***Command Line Print Statements***

- Initiating Reader thread process -

\* Instruction read in = W 00 10

In Reader thread: This is a WRITE command.

a\_str = 00, and d\_str = 10

a\_int = 0, and d\_int = 16

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = W 01 20

In Reader thread: This is a WRITE command.

a\_str = 01, and d\_str = 20

a\_int = 1, and d\_int = 32

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = W 02 30

In Reader thread: This is a WRITE command.

a\_str = 02, and d\_str = 30

a\_int = 2, and d\_int = 48

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = W 10 40

In Reader thread: This is a WRITE command.

a\_str = 10, and d\_str = 40

a\_int = 16, and d\_int = 64

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = W 12 50

In Reader thread: This is a WRITE command.

a\_str = 12, and d\_str = 50

a\_int = 18, and d\_int = 80

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = R 01

In Reader thread: This is a READ command.

a\_str = 01, and d\_str =

a\_int = 1, and d\_int = 80

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A READ operation is being performed.

a\_int = 1, and d\_int = 32

a\_str = 01, and d\_str = 20

Output address and data info to Out1.txt

\* Instruction read in = R 10

In Reader thread: This is a READ command.

a\_str = 10, and d\_str =

a\_int = 16, and d\_int = 80

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A READ operation is being performed.

a\_int = 16, and d\_int = 64

a\_str = 10, and d\_str = 40

Output address and data info to Out2.txt

\* Instruction read in = W 01 120

In Reader thread: This is a WRITE command.

a\_str = 01, and d\_str = 120

a\_int = 1, and d\_int = 288

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = W 11 03

In Reader thread: This is a WRITE command.

a\_str = 11, and d\_str = 03

a\_int = 17, and d\_int = 3

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = R 01

In Reader thread: This is a READ command.

a\_str = 01, and d\_str =

a\_int = 1, and d\_int = 3

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A READ operation is being performed.

a\_int = 1, and d\_int = 288

a\_str = 01, and d\_str = 120

Output address and data info to Out1.txt

\* Instruction read in = W 31 08

In Reader thread: This is a WRITE command.

a\_str = 31, and d\_str = 08

a\_int = 49, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A WRITE operation is being performed

\* Instruction read in = R 00

In Reader thread: This is a READ command.

a\_str = 00, and d\_str =

a\_int = 0, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A READ operation is being performed.

a\_int = 0, and d\_int = 16

a\_str = 00, and d\_str = 10

Output address and data info to Out2.txt

\* Instruction read in = R 01

In Reader thread: This is a READ command.

a\_str = 01, and d\_str =

a\_int = 1, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A READ operation is being performed.

a\_int = 1, and d\_int = 288

a\_str = 01, and d\_str = 120

Output address and data info to Out1.txt

\* Instruction read in = R 02

In Reader thread: This is a READ command.

a\_str = 02, and d\_str =

a\_int = 2, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A READ operation is being performed.

a\_int = 2, and d\_int = 48

a\_str = 02, and d\_str = 30

Output address and data info to Out2.txt

\* Instruction read in = R 10

In Reader thread: This is a READ command.

a\_str = 10, and d\_str =

a\_int = 16, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A READ operation is being performed.

a\_int = 16, and d\_int = 64

a\_str = 10, and d\_str = 40

Output address and data info to Out2.txt

\* Instruction read in = R 11

In Reader thread: This is a READ command.

a\_str = 11, and d\_str =

a\_int = 17, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A READ operation is being performed.

a\_int = 17, and d\_int = 3

a\_str = 11, and d\_str = 03

Output address and data info to Out1.txt

\* Instruction read in = R 12

In Reader thread: This is a READ command.

a\_str = 12, and d\_str =

a\_int = 18, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory2

In Memory b\_transport. A READ operation is being performed.

a\_int = 18, and d\_int = 80

a\_str = 12, and d\_str = 50

Output address and data info to Out2.txt

\* Instruction read in = R 31

In Reader thread: This is a READ command.

a\_str = 31, and d\_str =

a\_int = 49, and d\_int = 8

In Switch b\_transport. Target memory chosen is Memory1

In Memory b\_transport. A READ operation is being performed.

a\_int = 49, and d\_int = 8

a\_str = 31, and d\_str = 08

Output address and data info to Out1.txt

DONE!

This concludes the analysis for Final Exam – Part 4 (TLM-2.0).