The goal of the assignment is to develop an 8-bit Shift Register with 4 distinct modes of operation –

Mode 0 = Do Nothing

Mode 1 = Shift Right (with MSB = serial right input)

Mode 2 = Shift Left (with LSB = serial left input)

Mode 3 = Parallel Load

This 8-bit Shift Register is to be modeled using SystemC, along with utilizing a VCD Wave Viewer in order to generate waveform output. A test bench which simulates data passed to the Shift Register is created to verify full functionality of the element in SystemC.

**Shift Register Declaration** (Part A)

In developing this SystemC description of the Shift Register, a total of 2 files were created:

*shiftReg.h*

*shiftReg.cpp*

The first file, *shiftReg.h*, defines the *ShiftRegister* class, which is declared as a SC\_MODULE() according to the SystemC language. It’s one function, called update\_reg(), is declared here, and is registered as an SC\_METHOD(), with a sensitivity list based on the positive edge of rst or clk:

// Carlos Lazo

// ECE579

// Homework 05

#include <iostream>

#include <string>

#include "systemc.h"

using namespace std;

SC\_MODULE(ShiftRegister)

{

// Define all fixed module variables:

sc\_in\_clk clk; // Define internal clock

sc\_in<bool> rst; // Define internal reset

sc\_in<bool> sri; // Define serial right

sc\_in<bool> sli; // Define serial left

sc\_in <sc\_uint<2> > mode; // Define mode array

sc\_in <sc\_uint<8> > pin; // Define parallel input

sc\_out<sc\_uint<8> > pout; // Define parallel output

// Declare temporary module variables for computation:

sc\_uint<8> p\_temp;

sc\_uint<2> m\_temp;

// Declare function for ShiftRegister:

void update\_reg();

// Declare inline constructor for ShiftRegister:

SC\_CTOR(ShiftRegister)

{

// Set sensitivies based on rst & clk values.

SC\_METHOD(update\_reg);

sensitive << rst.pos() << clk.pos();

}

};

The corresponding file is the *shiftReg.cpp*, which implements the update\_reg function as described in the header file. This file also instantiates the testbench, which is a hard-coded sequence of events located in the sc\_main function of the file. It is here that the *ShiftRegister* is established, the VCD file is setup for outputting all port values, and that functionality is verified:

// Carlos Lazo

// ECE579

// Homework 05

#include "shiftReg.h"

#include "systemc.h"

// Define the functionality of the Shift Register:

void ShiftRegister::update\_reg()

{

// Current value to be processed were last contents of ShiftRegister

p\_temp = pout.read();

// Set internal mode to current mode

m\_temp = mode.read();

// Use a switch statement to perform ShiftRegister functionality

switch (m\_temp)

{

// Mode = 00: Do Nothing

case 0: break;

// Mode = 01: Shift right

case 1:

{

p\_temp = (sri.read(), p\_temp.range(7,1));

pout.write(p\_temp);

break;

}

// Mode = 10: Shift left

case 2:

{

p\_temp = (p\_temp.range(6,0),sli.read());

pout.write(p\_temp);

break;

}

// Mode = 11: Parallel load

case 3:

{

pout.write(pin);

break;

}

}

// Reset takes precedence and sets pout = 0

if (rst.read() == 1)

pout.write(0);

}

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

{

cout << endl << endl;

// Declare all sc\_signals in the main (testbench):

sc\_signal<bool> clk; // Define internal clock

sc\_signal<bool> rst; // Define internal reset

sc\_signal<bool> sri; // Define serial right

sc\_signal<bool> sli; // Define serial left

sc\_signal <sc\_uint<2> > mode; // Define mode array

sc\_signal <sc\_uint<8> > pin; // Define parallel input

sc\_signal<sc\_uint<8> > pout; // Define parallel output

// Connect all ports to ShiftRegister:

ShiftRegister sysC\_Reg ("sysC\_Reg");

sysC\_Reg.clk(clk);

sysC\_Reg.rst(rst);

sysC\_Reg.sri(sri);

sysC\_Reg.sli(sli);

sysC\_Reg.mode(mode);

sysC\_Reg.pin(pin);

sysC\_Reg.pout(pout);

// Open the VCD file to be used for data recording and place signals:

sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("reg\_Out");

sc\_trace(wf,clk,"clk");

sc\_trace(wf,rst,"rst");

sc\_trace(wf,sri,"sri");

sc\_trace(wf,sli,"sli");

sc\_trace(wf,mode,"mode");

sc\_trace(wf,pin,"pin");

sc\_trace(wf,pout,"pout");

// Set all initial variable values:

rst = 0;

mode = 0;

sri = 0;

sli = 0;

pin = 15; // 0x0F in Hex

// Apply a reset for 1 clock cycle:

rst = 1;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Deassert reset signal for 2 clock cycle:

rst = 0;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Perform a load operation:

mode = 3;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Shift right a total of 2 times, with sri = 1

sri = 1;

mode = 1;

for (int i = 0; i < 2; i++)

{

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

}

// Shift left a total of 2 times, with sli = 0

mode = 2;

for (int i = 0; i < 2; i++)

{

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

}

// Apply a reset for 1 clock cycle:

rst = 1;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Deassert reset signal for 1 clock cycles:

rst = 0;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Perform a load operation:

pin = 153; //0x99 in Hex

mode = 3;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Shift right 3x and fill with 1's

mode = 1;

sri = 1;

for (int i = 0; i < 3; i++)

{

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

}

// Change to do nothing and finish simulation

mode = 0;

clk = 0; sc\_start(1);

clk = 1; sc\_start(1);

// Close trace file and end simulation

sc\_close\_vcd\_trace\_file(wf);

return 0;

}

**Shift Register Testbench** (Part B)

All of the code necessary to implement the testbench is described above in *shiftReg.cpp*. Based on the inline comments and the variable declarations before the clock changes, the following steps are the expected outcome of the test bench, as it relates to *pout*. Note that all values in the VCD output will be displayed in Hexadecimal. For the sake of the shift operations, I will list the binary equivalents as well:

1. **RESET**

**Pout** = **h00**

1. **LOAD** :: din = h0F (b00001111)

**Pout** = **h0F** (b00001111)

1. **R\_SHIFT x 2 ::** sri *=* 1

**Pout** = **h87** (b10000111)

**Pout** = **hC3** (b11000011)

1. **L\_SHIFT x 2 ::** sli *=* 0

**Pout** = **h86** (b10000110)

**Pout** = **h0C** (b00001100)

1. **RESET**

**Pout** = **h00**

1. **LOAD** :: din = h99 (b10011001)

**Pout** = **h99** (b10011001)

1. **R\_SHIFT x 3 ::** sri *=* 1

**Pout** = **hCC** (b11001100)

**Pout** = **hE6** (b11100110)

**Pout** = **hF3** (b11110011)

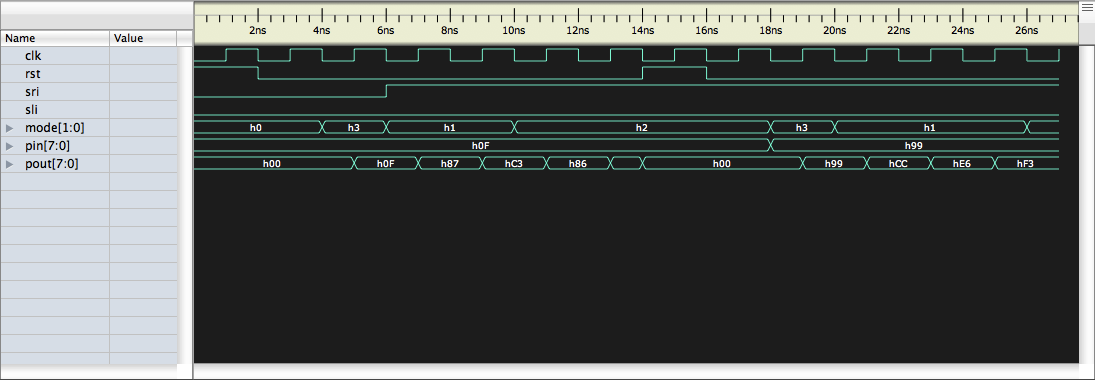
1. **DO NOTHING**

**Pout** = **hF3**

**<END TESTBENCH>**

The following VCD output confirms the exact steps listed above, based on the clock timings contained within the sc\_main portion testbench in the *shiftReg.cpp* file. For some reason, I could not install VCD Wave Viewer on my windows computer, so this output was taken from my MacBook Pro (MAC OS X), using the Scansion software program. The original VCD is located in the project directory of my code provided with this write-up.

**Waveform Output for the ShiftRegister Testbench**



The waveform outputs generated in this VCD file verify the operation of the testbench as described above.

This concludes the analysis for Homework 05.