The goal of the assignment is to design and code a simple serial-to-parallel converter (S2PC). The S2PC collects eight bits of received synchronous serial data, *serin*, (LSB first) and puts the parallel data on its parallel output. When in the ready state, its *ready* output is asserted. While in this state, serial transmission of data begins after a one-clock duration pulse on its start input. After collecting all eight bits of data, S2PC adds an odd parity bit to the received data and makes it available on its 9-bit *parout* output. Availability of parallel data is indicated with the assertion of *done* output. Valid data remains on the output until a next round of serial to parallel conversion is started (start is issued). A synchronous reset, *rst*, is provided to reset the controller and necessary data components.

The S2PC module, along with a testbench, will be developed in SystemC, and output in VCD format will serve to prove the full and correct operation of the circuit that has been developed.

**RT Level Design, Datapath, and Controller** (Parts A, B, & C)

The first part of the assignment asks us to model the S2PC at the RT Level, and include diagrams of the Datapath and Controller. As far as the RT Level design goes, the only items that are needed would be a 9-bit register (to hold the value of the 8-bit *parout* and the parity bit) and a counter, which will be used to address the different locations of the register. The simple Datapath, containing all RT Level components, is shown in the diagram below:



Similarly, the Controller of this signal will work with the Datapath in order to set appropriate control signals and keep track of where the S2PC is in operation. Both external and internal signals analyzed by the system are showcased here:



**S2PC SystemC Code & Testbench** (Parts D, E, & F)

In developing the SystemC description of the S2PC, a total of 3 files were created:

*S2PC.h*

*S2PC.cpp*

*S2PC\_Tester.cpp*

The first file, *S2PC.h*, defines the *S2PC* class, which is declared as a SC\_MODULE() according to the SystemC language. It’s contains two functions, controller() and datapath(), which are registered as SC\_THREADS(). All internal signals are declared here, with the controller() being looked at in the pos() edge of the clk and the datapath on the neg() edge of the clock:

// Carlos Lazo

// ECE579

// Homework 06

#include "S2PC.h"

#include "systemc.h"

// Define the functionality of the S2PC Controller:

void S2PC::controller()

{

// Declare all Controller state machine logic:

while (1)

{

// First, check the reset condition -

if (rst.read() == 1){

p\_state = rdy; }

// If not, perform next state computation -

else switch (p\_state)

{

// If ready and start = 1, begin reading in data.

case (rdy) :

{

if (start.read() == 1)

p\_state = rd\_in;

else

p\_state = rdy;

break;

}

// If in data collection stage, check counter state for transition.

// Else, continue reading in serial data.

case (rd\_in) :

{

if (counter == 7)

p\_state = out;

else

p\_state = rd\_in;

break;

}

// If in the output stage,

case (out) :

p\_state = rdy;

default : p\_state = rdy;

}

// Now define and set all Controller signals to Datapath:

if (p\_state == rdy)

{

ready = 1; done = 1;

count\_en.write(0); par\_en.write(1); ld\_en.write(0);

}

if (p\_state == rd\_in)

{

ready = 0; done = 0;

count\_en.write(1); par\_en.write(0); ld\_en.write(1);

}

if (p\_state == out)

{

ready = 0; done = 1;

count\_en.write(0); par\_en.write(1); ld\_en.write(0);

}

// Now that Controller signals are done, end thread.

wait();

}

}

// Define the functionality of the S2PC Datapath:

void S2PC::datapath()

{

while (1)

{

// First, check the reset condition:

if (rst.read() == 1)

t\_reg = 0;

// If we have enabled loading of the register, perform a shift:

else if (ld\_en.read() == 1)

{

t\_reg = (t\_reg[8], serin.read(), t\_reg.range(7,1));

// Increment the counter if currently enabled:

if (count\_en.read() == 1)

counter++;

// If counter is at maximum value, compute parity:

if (counter == 7)

{

t\_reg = (xor\_reduce(t\_reg), t\_reg.range(7,0));

counter = 0;

}

}

// If parout is ready, write the internal register to it:

if (par\_en.read() == 1)

parout.write(t\_reg);

wait();

}

}

The corresponding file is the *S2PC.cpp*, which implements both the controller() and datapath() functions as described in the header file. As can be seen, handshaking is done between the Controller function and the Datapath. This handoff of signals represents the RT Level behavior we expect if we were to design this circuit:

// Carlos Lazo

// ECE579

// Homework 06

#include "S2PC.h"

#include "systemc.h"

// Define the functionality of the S2PC Controller:

void S2PC::controller()

{

// Declare all Controller state machine logic:

while (1)

{

// First, check the reset condition -

if (rst.read() == 1){

p\_state = rdy; }

// If not, perform next state computation -

else switch (p\_state)

{

// If ready and start = 1, begin reading in data.

case (rdy) :

{

if (start.read() == 1)

p\_state = rd\_in;

else

p\_state = rdy;

break;

}

// If in data collection stage, check counter state for transition.

// Else, continue reading in serial data.

case (rd\_in) :

{

if (counter == 7)

p\_state = out;

else

p\_state = rd\_in;

break;

}

// If in the output stage,

case (out) :

p\_state = rdy;

default : p\_state = rdy;

}

// Now define and set all Controller signals to Datapath:

if (p\_state == rdy)

{

ready = 1; done = 1;

count\_en.write(0); par\_en.write(1); ld\_en.write(0);

}

if (p\_state == rd\_in)

{

ready = 0; done = 0;

count\_en.write(1); par\_en.write(0); ld\_en.write(1);

}

if (p\_state == out)

{

ready = 0; done = 1;

count\_en.write(0); par\_en.write(1); ld\_en.write(0);

}

// Now that Controller signals are done, end thread.

wait();

}

}

// Define the functionality of the S2PC Datapath:

void S2PC::datapath()

{

while (1)

{

// First, check the reset condition:

if (rst.read() == 1)

t\_reg = 0;

// If we have enabled loading of the register, perform a shift:

else if (ld\_en.read() == 1)

{

t\_reg = (t\_reg[8], serin.read(), t\_reg.range(7,1));

// Increment the counter if currently enabled:

if (count\_en.read() == 1)

counter++;

// If counter is at maximum value, compute parity:

if (counter == 7)

{

t\_reg = (xor\_reduce(t\_reg), t\_reg.range(7,0));

counter = 0;

}

}

// If parout is ready, write the internal register to it:

if (par\_en.read() == 1)

parout.write(t\_reg);

wait();

}

}

Finally, the *S2PC\_Testier.cpp* file contains the testbench for this project. Two simple scenarios have been tested and verified – resetting the circuit and loading in all 1s, and resetting the circuit and loading in alternating Boolean values of 0 and 1. The following is the testbench code:

// Carlos Lazo

// ECE579

// Homework 06

#include "S2PC.h"

#include "systemc.h"

using namespace std;

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

{

cout << endl << endl;

// Define all fixed module variables:

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

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

sc\_signal<bool> serin; // Define serial input of S2PC

sc\_signal<bool> start; // Define start signal

sc\_signal<bool> ready; // Indicating system is ready to start

sc\_signal<bool> done; // Indicating that serial input is done

sc\_signal<sc\_lv<9> > parout; // Define parallel output

// Connect all ports to ShiftRegister:

S2PC sysC\_S2PC ("sysC\_S2PC");

sysC\_S2PC.clk(clk);

sysC\_S2PC.rst(rst);

sysC\_S2PC.serin(serin);

sysC\_S2PC.start(start);

sysC\_S2PC.ready(ready);

sysC\_S2PC.done(done);

sysC\_S2PC.parout(parout);

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

sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("S2PC\_Out");

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

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

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

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

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

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

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

// Set all initial values:

clk = 0; rst = 1; serin = 1; start = 0;

// Reset the circuit to start off with:

clk = 1; sc\_start(1,SC\_NS);

clk = 0; sc\_start(1,SC\_NS);

rst = 0;

// Hit a pulse on start, which should fill in 1's to parout:

start = 1; sc\_start(1,SC\_NS);

start = 0; sc\_start(1,SC\_NS);

// Run a total of 10 pulses to see parout contain valid values:

for (int i = 0; i < 10; i++)

{

clk = 0; sc\_start(1,SC\_NS);

clk = 1; sc\_start(1,SC\_NS);

}

// Reset circuit once again:

clk = 0; rst = 1; serin = 1; start = 0;

clk = 1; sc\_start(1,SC\_NS);

clk = 0; sc\_start(1,SC\_NS);

rst = 0;

// Hit a pulse on start, then alternate serial input data:

start = 1; sc\_start(1,SC\_NS);

start = 0; sc\_start(1,SC\_NS);

for (int i = 0; i < 10; i++)

{

if (i%2 == 0)

serin = 0;

else

serin = 1;

clk = 0; sc\_start(1,SC\_NS);

clk = 1; sc\_start(1,SC\_NS);

}

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

return 0;

}

All output can be seen in the S2PC\_Out.vcd file. This concludes the analysis for Homework 06.