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 C++ classes, taking full advantage of language compartmentalization of different logic units (e.g. D Flip Flop). A testbench is also created, which will apply test data to the Shift Register and output the contents to an external file, which will detail all of the information and showcase the output.

**C++ Design of 8-bit Shift Register** (Part A)

The design of the 8-bit Shift Register encompasses a total of 2 classes that have been written and included in this homework solution:

*ShiftRegister.h*

*ShiftRegister.cpp*

The first file, *ShiftRegister.h*, serves to define a D Flip-Flop along with the class that will instantiate a ShiftRegister that will be tested.

The DFF contains the following class variables and functions:

*Q* : the current value at the output of the DFF

DFF() : default constructor that sets *Q* to a default value of ‘X’

set(char,char,char) : function that will set the DFF value based on D, clk, and rst

get\_Q() : function will return the value of Q currently in the DFF

The ShiftRegisgter contains the following class variables and functions:

d*in, pout* : variables that store the parallel in and out data to the ShiftRegister

*clk, rst, m1, m0, sri, sli* : inputs to the ShiftRegister that help determine mode and output *in\_str, pr\_dff* : strings that contain initial input and contents of ShiftRegister

*reg* : array of DFFs that represents the memory in the ShiftRegsiter

*dTemp* : temp char array used to store DFF values

ShiftRegister() : default constructor that sets all member variables to default values

load\_input(string) : function to parse input from external file into ShiftRegister

update\_reg() : updates the ShiftRegister based on current mode of operation

set\_output() : assign values to DFFs; assignment is superseded by rst and clk values

output\_reg() : formatted string that will output information to an external file nicely

// Carlos Lazo

// ECE579D

// Homework 03

#include <fstream>

#include <iostream>

#include <string>

using namespace std;

// Define the D-Flip Flop Class

class DFF {

private:

char Q;

public:

DFF(); // Default Constructor

void set (char, char, char);// Used to set DFF

char get\_Q() { return Q; } // Function to return Q;

};

// Define the ShiftRegister Class

class ShiftRegister {

private:

// Declare all private member variables of the Shift Register:

char din [9]; // Parallel input

char pout [9]; // Parallel output

char clk, rst, m1, m0, sri, sli;

string in\_str; // Stores input string for printing

string pr\_dff; // Stores previous flip flop values

DFF reg [8]; // DFF's that make up Shift Register

char dTemp [9]; // Define temp DFF chars for computation

public:

// Public functions that will be executed in order:

ShiftRegister(); // Default Constructor

void load\_input (string);// Parse input string intomember variables

void update\_reg(); // Based on m1m0 mode, update register accordingly

void set\_output(); // Set all DFF's, based on clock and reset signals.

string output\_reg();// Formatted output generated for the Shift Register.

};

The corresponding file is the *ShiftRegister.cpp*, which implements all functions described above in the header file. Detailed comments are written within each function to explain the operation. It is important to note that these functions will be run in a sequential order, as certain variables need to be initialized before others. Both input and output handling will be handled in the TestBench, which will be created external to the ShiftRegister.cpp file:

// Carlos Lazo

// ECE579D

// Homework 03

#include "ShiftRegister.h"

// Define the D-Flip Flop constructor

DFF::DFF() {Q = 'X';}

// Define the D-FLIP Flop set function

void DFF::set(char D, char clk, char rst)

{

if (rst == '1')

Q = '0';

else if (clk == 'P')

Q = D;

}

// Define the ShiftRegister constructor

ShiftRegister::ShiftRegister()

{

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

{ din[i] = 'X'; pout[i] = 'X'; dTemp[i] = 'X'; }

// Note that DFF's have Q = 'X' by default, as per that specific constructor

din[8] = '\0'; pout[8] = '\0'; dTemp [8] = '\0';

clk = 'X'; rst = 'X'; m1 = 'X'; sri = 'X'; sli = 'X';

}

// Define the ShiftRegister load\_input function

void ShiftRegister::load\_input(string indata)

{

in\_str = indata;

// The input stream from the text file is as follows:

// m1 m0 | sri | srl | din | rst | clk ==> Information

// 0 1 2 3 4-11 12 13 ==> String Location

m1 = indata[0]; m0 = indata[1];

sri = indata[2]; sli = indata[3];

for (int i = 4; i < 12; i++)

din[i-4] = indata[i]; // din[0] = MSB, din[7] = LSB

rst = indata[12]; clk = indata[13];

// Store current flip-flop values:

pr\_dff = "";

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

pr\_dff += pout[i];

}

void ShiftRegister::update\_reg()

{

// Begin analyzing mode logic -

if ( (m1 == '0') && (m0 == '0') )

{

// Mode 0 - Do nothing. Set dTemp values to current DFF values

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

dTemp[i] = reg[i].get\_Q();

}

if ( (m1 == '0') && (m0 == '1') )

{

// Mode 1 - Right Shift

for (int i = 0; i < 7; i++)

dTemp[i+1] = reg[i].get\_Q(); // Shift all elements to the right

dTemp[0] = sri; // Set MSB to the sri input

}

if ( (m1 == '1') && (m0 == '0') )

{

// Mode 2 - Left Shift

for (int i = 0; i < 7; i++)

dTemp[i] = reg[i+1].get\_Q(); // Shift all elements to the left

dTemp[7] = sli; // Set LSB to the sli input

}

if ( (m1 == '1') && (m0 == '1') )

{

// Mode 3 - Parallel Load

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

dTemp[i] = din[i]; // Set elements to load input

}

}

void ShiftRegister::set\_output()

{

// Set all DFF - this is dependant on clock and reset values.

// Utilize values determined in the update\_reg() function.

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

reg[i].set(dTemp[i], clk, rst); // Update all DFFs

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

pout[i] = reg[i].get\_Q(); // Set parallel output to DFF Q values

}

string ShiftRegister::output\_reg()

{

string pretty\_print;

pretty\_print += "Current ShiftRegister contains the following values: ";

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

pretty\_print += pr\_dff[i];

pretty\_print += "\n";

pretty\_print += "Input = " + in\_str.substr(4,8) + ", Mode = " + in\_str.substr(0,2);

pretty\_print += ", Clk = "; pretty\_print += in\_str[13];

pretty\_print += ", Rst = "; pretty\_print += in\_str[12];

pretty\_print += ", sri = "; pretty\_print += in\_str[2];

pretty\_print += ", sli = "; pretty\_print += in\_str[3];

pretty\_print += "\n\t";

pretty\_print += "After operation, ShiftRegister now contains: ";

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

pretty\_print += pout[i];

pretty\_print += "\n\n";

return pretty\_print;

}

**TestBench** (Part B)

The *shiftRegTestbench.cpp* file is created to instantiate a *ShiftRegister*. Input is taken from the *indata.tst* file, and output is written to the *outdata.tst* file. The input stream is continually read until the … sequence is found, at which point the program is terminated. The following is the Testbench .cpp file:

// Carlos Lazo

// ECE579D

// Homework 03

#include "ShiftRegister.h"

// Create main testbench for the ShiftRegister:

int main ()

{

string inVec;

ShiftRegister UUT;

ifstream finp ("indata.tst");

ofstream fout ("outdata.tst");

char stop\_in ('0');

finp >> inVec; // Read in first line of testbench data

while (stop\_in != '.')

{

UUT.load\_input(inVec); // Load inputs into ShiftRegister

UUT.update\_reg(); // Compute operation based on mode

UUT.set\_output(); // Set DFFs based on computed values

fout << UUT.output\_reg(); // Output data to external file

// Gather next data input:

finp >> inVec;

stop\_in = inVec[0];

}

fout << "END FILESTREAM";

return 0;

}

Each line of the *indata.tst* file contains 6 different pieces of information for each and every clock cycle of the testbench: mode select, sri, sli, din, rst, and clk. The D Flip-Flop sets the current D variables when the clock is set to a value of P. The following is a list of events that the Testbench sequentially verifies:

1. Shift Register is reset from values of ‘X’ to ‘0’
2. Parallel Load of 00001111
3. Right shift with sri = 1
4. Right shift with sri = 1
5. Left shift with sli = 0
6. *Attempted* Left shift with sli = 0, but clk = N so should not occur
7. *Attempted* Parallel Load of 00001111, but rst = 1 so rst takes precedence
8. Parallel Load of 10011001

The layout of the *indata.tst* file, as described in *ShiftRegister.cpp*, is as follows:

m1, m0, sri, srl ,din [7:0], rst, clk 🡺 14 data inputs per line

The following are the contents of *indata.tst*, and map directly to the Testbench events listed above:

**indata.tst** \* Reminder : line = m1, m0, sri, srl ,din [7:0], rst, clk

0000000000001P

1100000011110P

0110000011110P

0110000011110P

1010000011110P

1010000011110N

1110000011111P

1110100110010P

...

When running the *shiftRegTestbench.cpp* file, output is generated in the following format:

Current ShiftRegister contains the following values: <current contents>

Input = ????????, Mode = MM, Clk = C, Rst = R, sri = R, sli = L

After operation, ShiftRegister now contains: NNNNNNNN

<current contents> = list current values located in the Shift Register (pre-mode operation)

???????? = input based on input file

MM = mode, C = clock pulse, R = reset value

R = serial right shift value, L = serial left shift value

NNNNNNNN = new values after mode change has been applied for that clock

Comparisons will be made between current and new ShiftRegister contents to verify operation. Upon running the testbench with the input stream above, the following output is generated:

**outdata.tst**

Current ShiftRegister contains the following values: XXXXXXXX

Input = 00000000, Mode = 00, Clk = P, Rst = 1, sri = 0, sli = 0

After operation, ShiftRegister now contains: 00000000

Current ShiftRegister contains the following values: 00000000

Input = 00001111, Mode = 11, Clk = P, Rst = 0, sri = 0, sli = 0

After operation, ShiftRegister now contains: 00001111

Current ShiftRegister contains the following values: 00001111

Input = 00001111, Mode = 01, Clk = P, Rst = 0, sri = 1, sli = 0

After operation, ShiftRegister now contains: 10000111

Current ShiftRegister contains the following values: 10000111

Input = 00001111, Mode = 01, Clk = P, Rst = 0, sri = 1, sli = 0

After operation, ShiftRegister now contains: 11000011

Current ShiftRegister contains the following values: 11000011

Input = 00001111, Mode = 10, Clk = P, Rst = 0, sri = 1, sli = 0

After operation, ShiftRegister now contains: 10000110

Current ShiftRegister contains the following values: 10000110

Input = 00001111, Mode = 10, Clk = N, Rst = 0, sri = 1, sli = 0

After operation, ShiftRegister now contains: 10000110

Current ShiftRegister contains the following values: 10000110

Input = 00001111, Mode = 11, Clk = P, Rst = 1, sri = 1, sli = 0

After operation, ShiftRegister now contains: 00000000

Current ShiftRegister contains the following values: 00000000

Input = 10011001, Mode = 11, Clk = P, Rst = 0, sri = 1, sli = 0

After operation, ShiftRegister now contains: 10011001

END FILESTREAM

A quick analysis of the output verifies the testbench events listed previously:

@ **Block 1**: ShiftRegister is reset, so all values go from ‘X’ to ‘0’

@ **Block 2**: Parallel Load of input value into ShiftRegister

@ **Block 3**: Perform right shift, with MSB = sri = 1

@ **Block 4**: Perform right shift, with MSB = sri = 1

@ **Block 5**: Perform left shift, with LSB = sli = 0

@ **Block 6**: clk = N, so effectively do nothing – ShiftRegister contents stay the same

@ **Block 7**: Even though mode = Parallel Load, clr = 1 forces ShiftRegister to clear

@ **Block 8**: Parallel Load of input value into Shift Register

This concludes the analysis of Homework 03.