Eastern Mediterranean University
Computer Engineering Department

CMSE 222 Introduction to Computer Organization

Lab 4 & Lab 5

Clocked Sequential Circuits and Registers Using Verilog HDL

OBJECTIVES:

There are two objectives in this laboratory work, one aims to introduce a practical work on the design of synchronous sequential circuits from architectural and behavioral descriptions and one aims to introduce a practical work on the design of Registers from architectural and behavioral descriptions.

- The architectural description covers both the schematic and the software implementation of circuits designed through the conventional design procedure. The behavioral descriptions cover the implementation using Mealy and Moore type state transition diagrams.
- The architectural description covers both the schematic and the software implementation of circuits designed through the conventional design procedure. The behavioral descriptions cover the implementation using state transition diagrams.

Important Note: *For each of the following experimental tasks (in each phase), open a new project to avoid compilation errors due to multiple use of components within the same project's files.*

1. Clocked Sequential Circuits

Phase 1: **Question**

Assume that the circuit to be designed has one input X and one output Y such that Y=1 iff there are three or more consecutive ones over the input X; Y=0 otherwise.

Considering the Mealy type state transition diagram in class, design this clocked sequential circuit using JK-FFs. Then, input the schematic diagram of your design into VeriLog HDL and verify its correctness through waveform simulations (follow the steps explained in the first experimental work).

Phase 2: Implementing the design in Verilog HDL

Enter the VeriLog code of your design using Quartus Lite development suite. Compile and simulate your code to verify its correctness (follow the steps explained in the first experimental work).

Phase 3: Implementing the design using Mealy-type State Transition Diagram

Consider the Mealy-type state transition diagram described in lecture, the corresponding Verilog HDL code that implements the state-transition and output generation behavior of this digital system is given below:

```
module Seq_3Ones_Detect_Mealy(X,Clk,Y);
       input Clk,X;
       output Y;
       reg Y;

       reg [1:0] pstate,nstate;  // present and next state variables as
registers
       parameter S0=2'b00, S0=2'b01, S0=2'b10, S0=2'b11; // state assignment

       always @(posedge Clk)
              case(pstate)
                     S0: if (x) begin nstate=S1; Y=0; end
                          else    begin nstate=S0; Y=0; end
                     S1: if (x) begin nstate=S2; Y=0; end
                          else    begin nstate=S0; Y=0; end
                     S2: if (x) begin nstate=S3; Y=1; end
                          else    begin nstate=S0; Y=0; end
                     S3: if (x) begin nstate=S3; Y=1; end
                          else    begin nstate=S0; Y=0; end
              endcase
       // sequential logic for state transitions
              Pstate <= nstate;
endmodule
```

Write the above given code in VeriLog HDL environment and simulate it to verify its correctness.

Phase 4: Implementing the design using Moore-type State Transition Diagram

Consider the Moore-type state transition diagram described in lecture, the corresponding Verilog HDL code that implements the state-transition and output generation behavior of this digital system is given below:

```
module Seq_3Ones_Detect_Moore(X,Clk,Y);
       input Clk,X;
       output Y;

       reg [1:0] state;  // state variables as register
       parameter S0=2'b00, S0=2'b01, S0=2'b10, S0=2'b11; // state assignment

       always @(posedge Clk)
              case(state)
                     S0: if (x) state <= S1;
                          else   state <= S0;
                     S1: if (x) state <= S2;
                          else   state <= S0;
                     S2: if (x) state <= S3;
                          else   state <= S0;
                     S3: if (x) state <= S3;
                          else   state <= S0;
              endcase

       // define the output
              assign Y=(state == S3);
endmodule
```

Write the above given code in VeriLog HDL environment and simulate it to verify its correctness.

EXPERIMENT:

Design a clocked sequential circuit with one input X and one output Z for the detection of the 4-bit sequence **0110** on input line X. Output Z=1 when this sequence is detected, Z=0 otherwise. Overlapping of 4-bit codes are allowed. Assume that MSB arrives first. You can use either Mealy of Moore type for your solution.

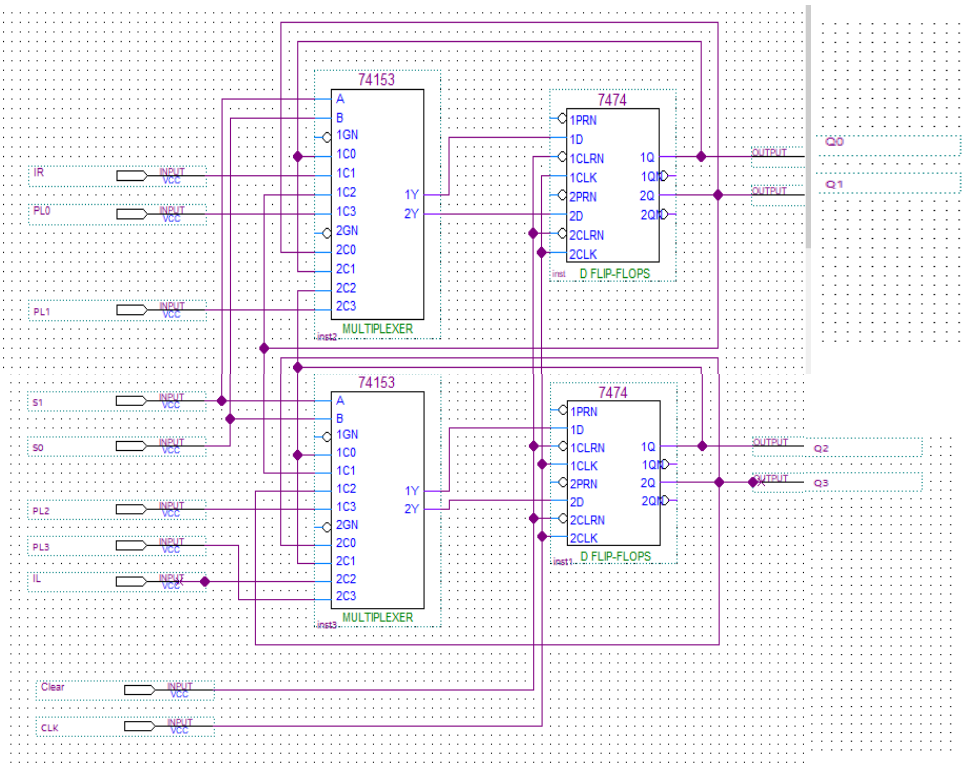
2. Registers

Phase 1: Schematic-Entry

Assume that we want to design the following multi-function register that is controlled by two control inputs S1 and S0 as follows:

Mode Control		**Register Operation**
S1	***S0***	
0	0	*No change*
0	1	*Shift left*
1	0	*Shift right*
1	1	*Parallel Load*

The schematic circuit corresponding to this multi-function register is given below:



Phase 2: Implementing the Architectural Design of Multi-Function Register in Verilog HDL

Enter the following architectural VeriLog code of multifunction register design into Quartus Lite development suite. Compile and simulate your code to verify its correctness.

```
/* 4-bit Multifucntion Register controlled by two control inputs S1 and S0 as
follws:
s1 s0=00 No change --- 01 Shift left --- 10 Shift right --- Parallel load
*/
module MultiFuncRegister_Arch(Clear,CLK,S,PL,IL,IR,Q);
	input CLK;
	input Clear;
	input IL, IR;		// Serial load from left and right
	input [1:0] S;		// Vector of control inputs S1 and S0
	input [3:0] PL;		// Parallel load
	output [3:0] Q;		// Register ouputs Q3, Q2, Q1, Q0
	wire [3:0] W;		// Internal signals among components

	MUX_4_1 m1(W[0],S[1],S[0],PL[0],IL,Q[1],Q[0]);
	MUX_4_1 m2(W[1],S[1],S[0],PL[1],Q[0],Q[2],Q[1]);
	MUX_4_1 m3(W[2],S[1],S[0],PL[2],Q[1],Q[3],Q[2]);
	MUX_4_1 m4(W[3],S[1],S[0],PL[3],Q[2],IR,Q[3]);

	D_FF d1(Q[0],W[0],CLK,Clear);
	D_FF d2(Q[1],W[1],CLK,Clear);
	D_FF d3(Q[2],W[2],CLK,Clear);
	D_FF d4(Q[3],W[3],CLK,Clear);
endmodule

module D_FF(Q,D,CLK,CLR);
	input D,CLK,CLR;
	output reg Q;

	always @(posedge CLK)
		if (CLR == 1'b1)
			Q<= 1'b0;
		else
			Q<= D;
endmodule

module MUX_4_1(Y,S1,S0,I3,I2,I1,I0);
	input S1,S0,I3,I2,I1,I0;
	output reg Y;

	always @(S1,S0,I3,I2,I1,I0)
	begin
		if (S1==0 & S0==0)
			Y=I0;
		else if (S1==0 & S0==1)
			Y=I1;
		else if (S1==1 & S0==0)
			Y=I2;
		else if (S1==1 & S0==1)
			Y=I3;
	end
endmodule
```

EXPERIMENT:

Behavioral VeriLog code of the above-described multifunction register is given below

```
/* Behavioral description of a multifunction register in veriLog HDL
s1 s0=00 No change --- 01 Shift left --- 10 Shift right --- Parallel load
*/
module MultiFunctRegister_Behav(Clear,CLK,S,PL,IL,IR,Q);
input Clear, CLK;
input [3:0] PL;
input [1:0] S;
input IL,IR;
output [3:0] Q;

reg [3:0] R;

always @(posedge CLK)
begin
	if (Clear == 1)
		R <= 4'b0000;
	else if (S[1]==0 & S[0]==0)   // No change
		R <= R;
	else if (S[1]==0 & S[0]==1)   // Shift left
	begin
		R[0] <= IR; R[1] <= R[0];
		R[2] <= R[1]; R[3] <= R[2];
	end
	else if (S[1]==1 & S[0]==0)     // Shift right
	begin
		R[3] <= IL; R[2] <= R[3];
		R[1] <= R[2]; R[0] <= R[1];
	end
	else if (S[1]==1 & S[0]==1)
	begin
		R=PL;
	end
end

assign Q = R;

endmodule
```

Modify the above-given behavioral code to design a 4-bit multifunction register that operates as follows:

Enable	S1	S0	Operation Mode
0	x	x	*No change*
1	0	0	*Rotate left*
1	0	1	*XOR contents with (0101)*
1	1	0	*Rotate right*
1	1	1	*Parallel load*