Logic Lab 6: Introduction to Sequential Logic using Latches and Flip Flops

Group # 2

Nathan Phipps, Wanzek

July 22, 2019

ABSTRACT:

In this lab we made Verilog code to create multiple latches and flip flops. In total, we created 6 separate modules each modeling a different type of latch or flip flop. Additionally, we implemented the use of “always @ statements and blocks” in several of these modules to fulfill our objectives in creating these latches/flip-flops. Through our work, within this lab, we are able to provide a full report on the processes, taken, as we created our latches and flip flops. The outline for this report will appear as follows:

1.) Group Member Contributions:

A brief discussion on group member contributions

2.) Lab Analysis:

A.) Analysis of part 1 (SR Latch) from our lab
B.) Analysis of part 2 (Gated SR Latch) from our lab
C.) Analysis of part 3 (D Latch) from our lab
D.) Analysis of part 4 (D Flip Flop) from our lab
E.) Analysis of part 5 (T Flip Flop) from our lab
F.) Analysis of part 6 (JK Flip Flop) from our lab

3.) Wave Diagram Pictures (Note: No Wave Simulation was required for the D latch):

A.) Wave diagram simulations from part 1 (SR Latch) from our lab
B.) Wave diagram simulations from part 2 (Gated SR Latch) from our lab
D.) Wave diagram simulations from part 4 (D Flip Flop) from our lab
E.) Wave diagram simulations from part 5 (T Flip Flop) from our lab
F.) Wave diagram simulations from part 6 (JK Flip Flop) from our lab

4.) Test Case Pictures

A.) Pictures of the test cases part 1 (SR Latch)
B.) Pictures of the test cases part 2 (Gated SR Latch)
C.) Pictures of the test cases part 3 (D Latch)

5.) Verilog Code:

A.) I.) Code from part 1 (SR Latch) from our lab
II.) Top level from (SR Latch)
B.) I.) Code from part 2 (Gated SR Latch) from our lab
II.) Top Level (Gated Sr Latch)
C.) I.)Code from part 3 (D Latch) from our lab
II.) Top Level (D Latch)
D.) Code from part 4 (D Flip Flop) from our lab
E.) Code from part 5 (T Flip Flop) from our lab
F.) Code from part 6 (JK Flip Flop) from our lab

GROUP MEMBER CONTRIBUTIONS

During the entirety of this lab, both group members, Nathan Phipps, and Erich Wanzek performed equal duties, including writing verilog code, performing wave diagram simulations, taking photos of the Altera FPGA board, and writing the lab report.

ANALYSIS:

Analysis Part 1(SR Latch) :

In part 1 of lab 6, we wrote verilog code that modeled an “SR (set, reset)” latch. This code helped us to understand several new concepts, with which we were first introduced to, during a brief preface to this lab. Specifically this code utilized registers, assign statements for those registers, blocking statements and always @ statements. Beyond the code for the SR latch module, we also implemented a top level to test our verilog code for this latch. After performing multiple tests, we were able to assess that our data agreed with the SR latch’s truth table, indicating that our code successfully modeled the SR latch. These pictures can be found in part “4.) A.),” of this lab report or within part 1 of our zip folder.

Analysis Part 2(Gated SR Latch):

During part 2 of lab 6, we created verilog code that modeled a “Gated SR (set, reset)” latch. To elaborate, this code instantiated our previous “SR” latch, but, when calling the old code we entered logical expressions into the sr_latches arguments (not to be confused with the arguments, or ports of the gated_sr_latch). For clarification please consult our verilog code, which can be found in part “5.) B.),” of this lab report, or within the “.v” file found in part 2 of our zip folder. Lastly, we incorporated a top level to test several cases, pertaining to the “Gated SR” latch’s truth table. Upon further inspection, from the aforementioned test cases, it appeared that our data did agree with the truth tables for the “Gated Sr” latch. Photos from these test cases can be found in part “4.) A.),” of this lab report, or within part 2 of our zip file folder.

Analysis Part 3(D Latch):

Next, in part 3, we made a code for a D Latch. This code used an “always @ block,” as well as, an “if statement” to help fully implement the D latch. Furthermore, within the if statement (which was nested in the always @ statement), we utilized blocking statements for our registers, rQ, and rQnot, allowing the registers to be set to the logical expressions for the D flip flop. In addition to creating the verilog code for the D latch, we also coded a top level to allow for full implementation of our D latch onto an Altera FPGA board. All this being done in order to test a number of cases, pertaining to the D latches truth table, as set forth by the objectives found within our lab rubric. Pictures from these test cases can be found within the lab report under part “4.) C.),” or part 3 from our attached zip file folder. Lastly, it is important to note, this part of the lab didn’t require a wave simulation, so no wave simulation file will be provided.

Analysis Part 4(D Flip Flop):

From part 4, we created a module of code for a D flip flop. In this part of the lab we utilized an always @ statement, with a predefined verilog function titled “posedge,” entered into the statement’s argument. The full expression in the always @ statement’s argument was “posedge clock,” which detected the positive edge of clock pulse. Next, nested within the always @ statement we utilized blocking statements on registers, in order to set the registers to logical expressions for the D flip flop. Since our lab rubrics did not request test case pictures, we did not have to create a top level to implement this code onto an Altera FPGA board. Finally, after performing a wave simulation of this trial, we were able to conclude that our code successfully simulated a D flip flop and its associated truth tables. The aforementioned wave simulation can be found in part “3.)D.)” of this lab report.

Analysis Part 5 (T Flip Flop):

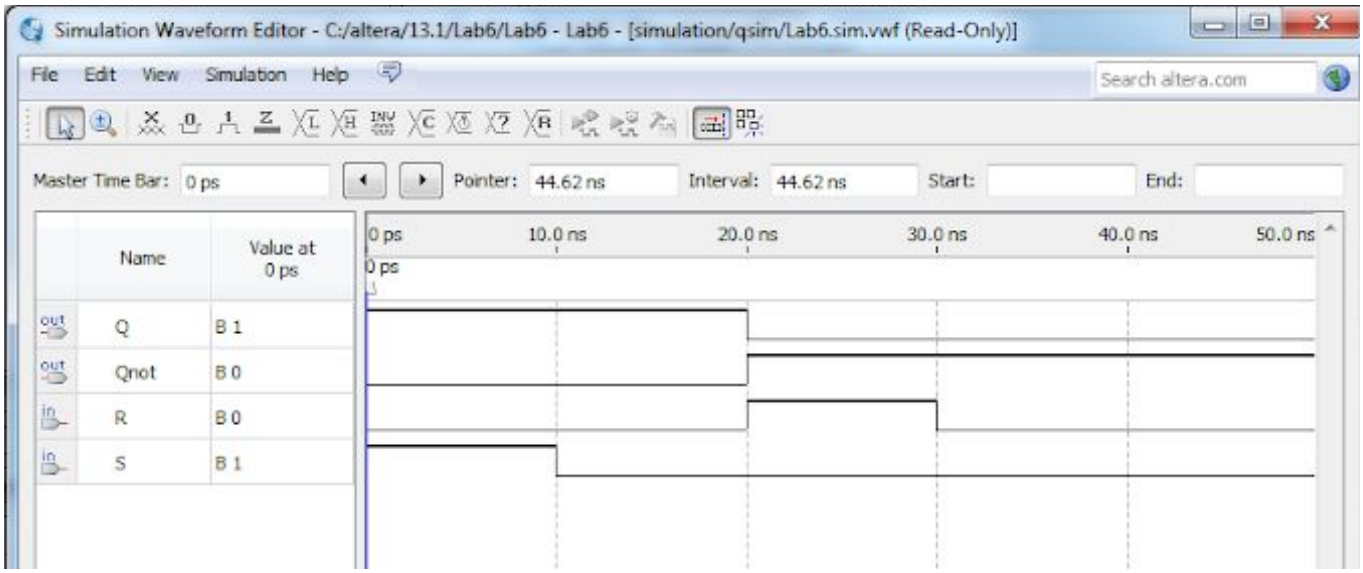
Within part 5, from the lab rubric, we generated verilog code for a T flip flop. It is during this part that we used an always @ statement with a positive edge clock trigger (inside of the always @ statements argument). Nested within the always @ statement, we set a register “Qnext” to a logical expression (“T ^ Q”). Finally, although it wasn’t required, we created an expression that represented the complemented expression for Qnext, which we called Qnextnot and then we ran a wave simulation. The wave simulation data from this part appeared to agree with the truth tables for the T flip flop as provided by the lab rubric, allowing us to conclude that our verilog code successfully modeled this T flip flop.

Analysis Part 6 (JK Flip Flop):

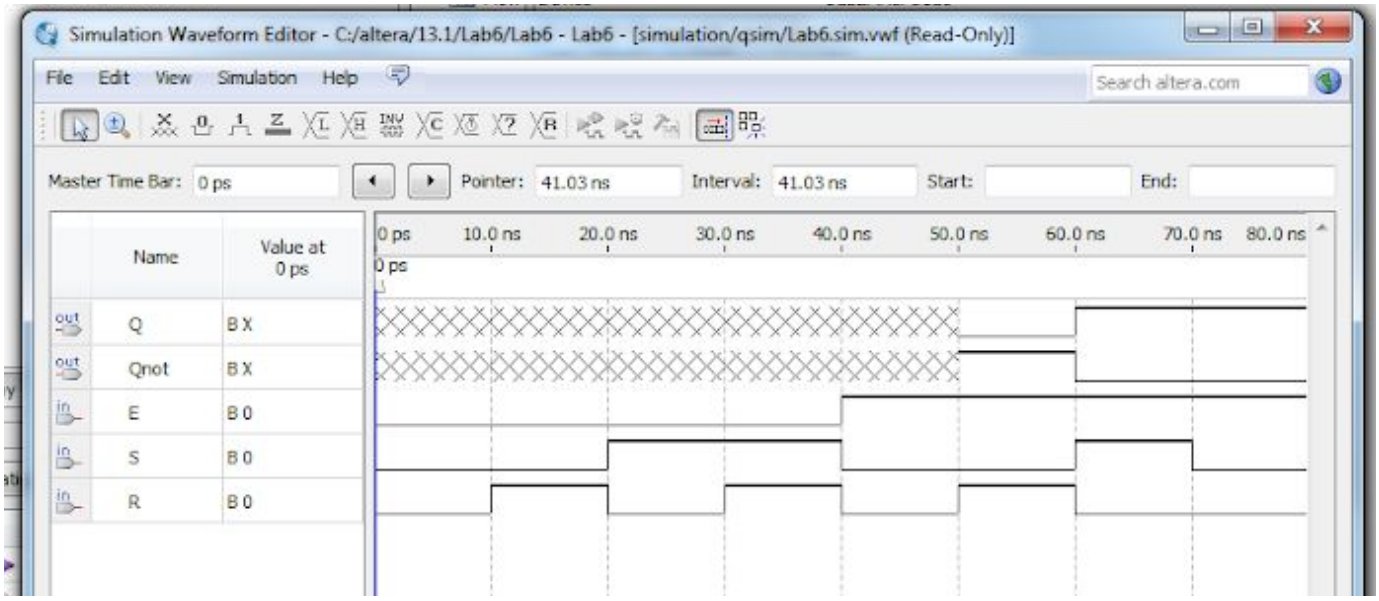
Finally, during part 6, we created a JK flip flop using verilog code. In this part we, again, utilized an always @ statement with a positive clock edge trigger. Additionally, nested within the always @ statement we assigned a logical expression (“J&(~Q))|((~K)&Q”) to our output register (“Q”). Finally we assigned an additional output Qnot to a logical expression (“~Q”).

Wave Diagrams

SR Latch Wave Diagram Simulation:

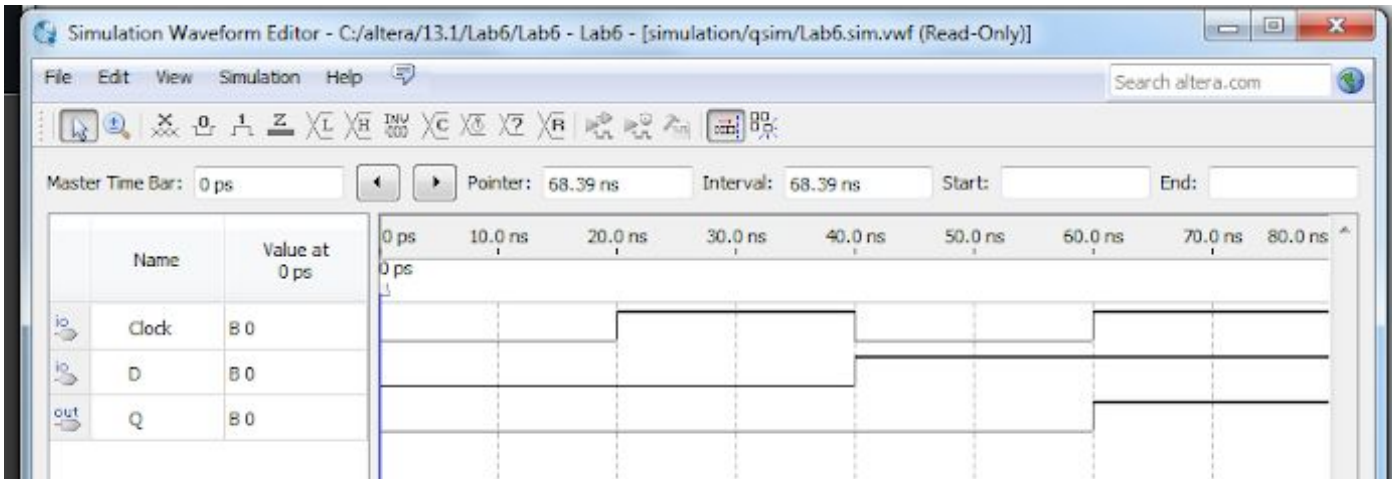


Gated SR Latch Wave Diagram Simulation:

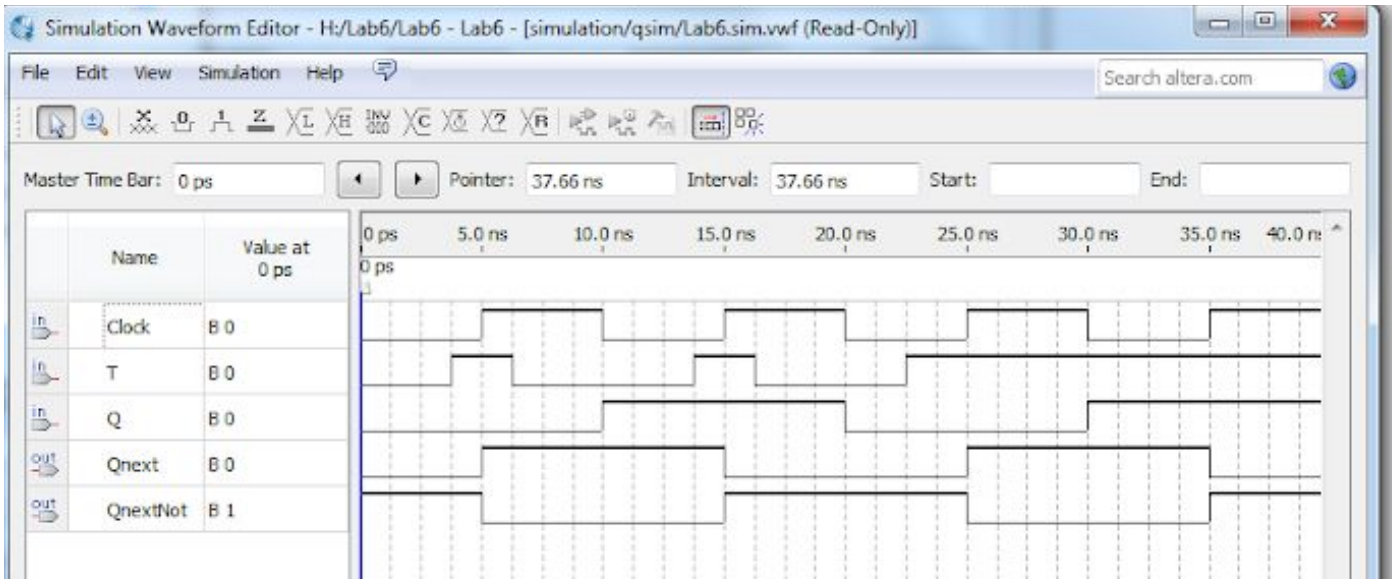


D Latch Wave Diagram Simulation: The lab didn't require a D latch Simulation

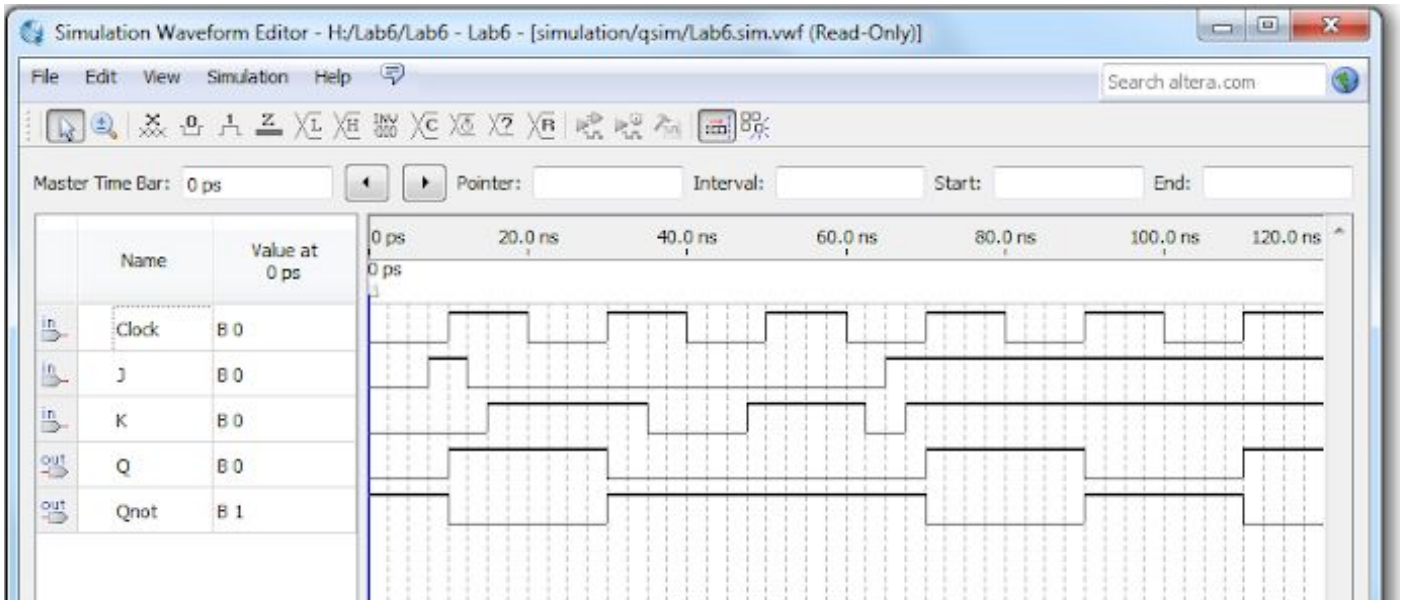
D Flip Flops Wave Diagram Simulation:



T Flip Flops Wave Diagram Simulation:



JK Flip Flops Wave Diagram Simulation:



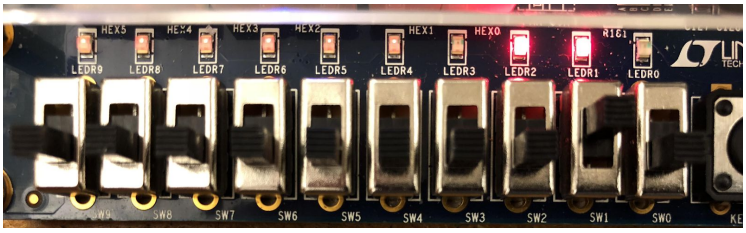
Test Case Pictures

Part 1 SR Latch

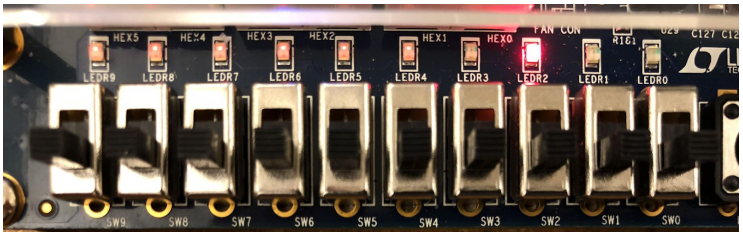
Assignments:

1. R to SW[0] and LEDR[0]
2. S to SW[1] and LEDR[1]
3. Q to LEDR[2]
4. Q' to LEDR[3]

S = 1, R=0, Q=1, Q'=0

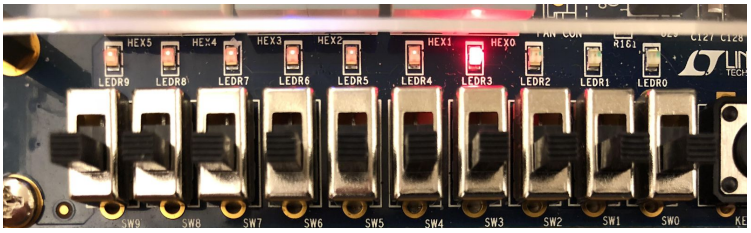


S = 0, R=0, Q=1, Q'=0



S = 0, R=1, Q=0, Q'=1

S = 0, R=0, Q=0, Q'=1

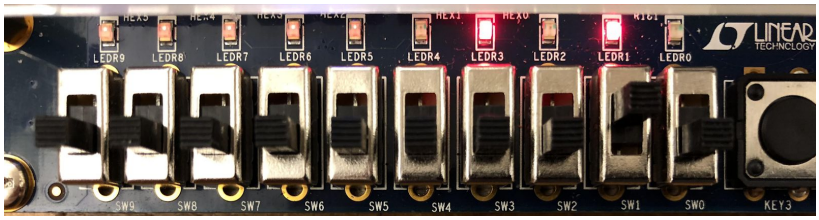


Part 2 Gated SR Latch

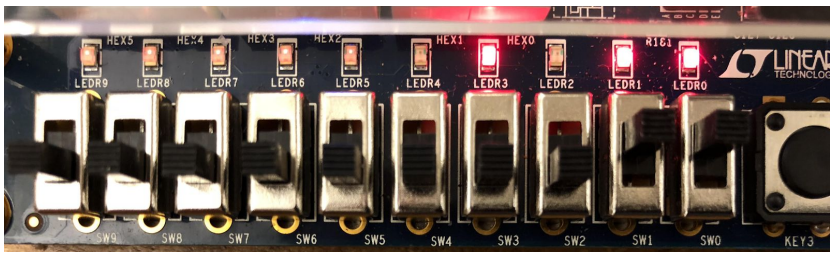
Assignments:

1. R to SW[0] and LEDR[0]
2. S to SW[1] and LEDR[1]
3. E to SW[2] and LEDR[2];
4. Q to LEDR[2]
5. Q' to LEDR[3]

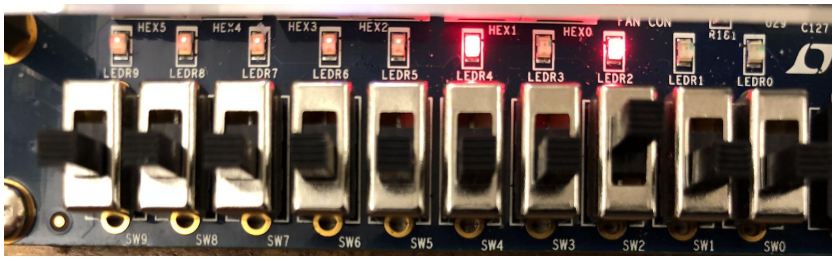
Case ESR: 010



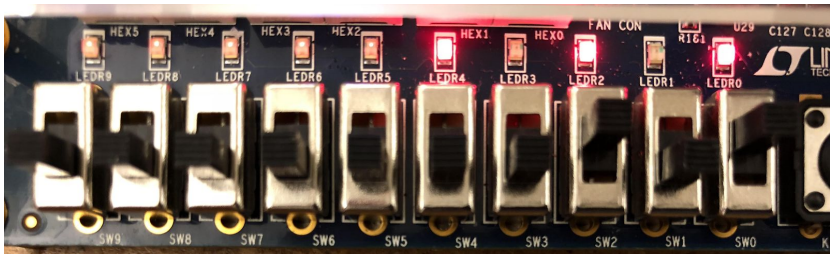
Case ESR: 011



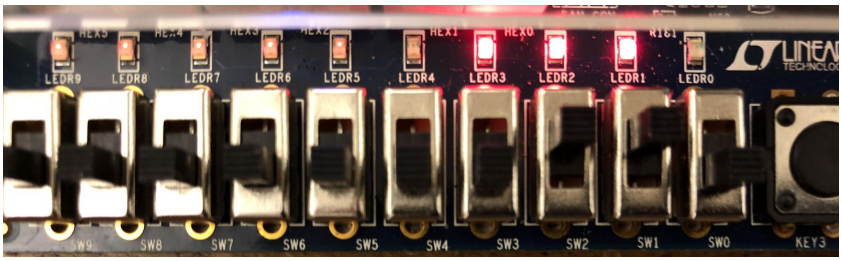
Case ESR: 100



Case ESR: 101



Case ESR: 110



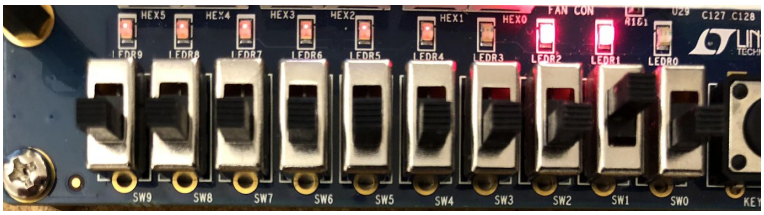
Part 3 D Latch

Assignments:

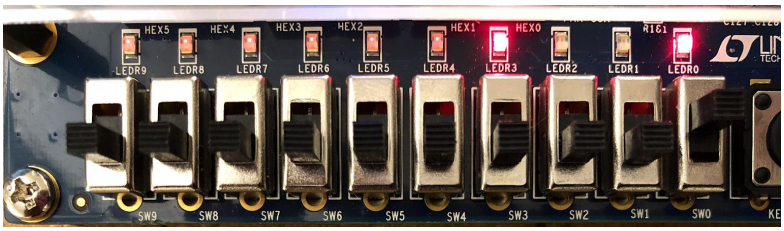
1. Assign E to SW[0] and LEDR[0]
2. Assign D to SW[1] and LEDR[1]
3. Assign Q to LEDR[2]
4. Assign Q’ to LEDR[3]

E=0, D=0, Q=Latch, Q’=Latch

E=0, D=1, Q=Latch, Q’=Latch



E=1, D=0, Q=0, Q’=1



E=1, D=1, Q=1, Q’=0

Verilog Codes

A.) I.) Code From Part 1 (SR Latch) From Our Lab

```
module sr_latch (S,R,Q, Qnot); //Declares module

	input wire S, R; //defines input wires, set and reset
	output Q, Qnot; //defines output wires Q, Qnot

	reg rQ, rQnot; //defines registers, rQ, rQnot

	always @ (*) begin //initiates always @

		rQ <=  ~(R |  rQnot); //defines expression for rQ
		rQnot <=  ~(S | rQ); //deines expression for rQnot

	end //ends always @

	assign Q = rQ; //assigns Q to an expression
	assign Qnot = rQnot; //assigns Qnot to an expression

endmodule //ends module
```

A.) II.) Top Level Code From Part 1 (SR Latch) From Our Lab

```
module top_level_sr_latch(SW, LEDR); //declares module

	input wire [1:0]SW; //defines input wires for Switches
	output wire [3:0]LEDR; //defines output wires for LEDRs

	sr_latch (SW[1],SW[0],LEDR[2], LEDR[3]); //instantiates sr_latch
	assign LEDR[1:0] = SW[1:0]; //assigns LEDRs to Switches

endmodule //ends module
```

B.) I.) Code From Part 2 (Gated SR Latch) From Our Lab

```
module gated_sr_latch(S,R,E, Q, Qnot); //declares module

	input wire S, R, E; //defines input wires S,R,E
	output Q, Qnot; //defines outputs Q, Qnot

	sr_latch ((S&E),(R & E),Q, Qnot); //instantiates SR_latch and uses additional logical expressions

endmodule //ends module
```

B.) II.) Top Level Code From Part 2 (Gated SR Latch) From Our Lab

```
module top_level_gated_sr_latch(SW, LEDR); //declares module

	input wire [2:0]SW; //defines input wires for Switches
	output wire [4:0]LEDR; //defines output wires for LEDRs

	gated_sr_latch(SW[1],SW[0],SW[2], LEDR[3], LEDR[4]); //instantiates gated_sr_latch
	assign LEDR[2:0] = SW[2:0]; //assigns LEDRs to Switches

endmodule //ends module
```

C.) I.) Code From Part 3 (D Latch) From Our Lab

```
module D_latch_try_2(D,E, Q, Qnot); //declares module

	input wire D, E; //defines input wires D, E
	output Q, Qnot; //defines output Q, Qnot
```

```
	reg rQ, rQnot; //defines rQ, rQnot

	always @(*) begin //initiates always @
		if(E) begin //if statement triggered by E

			rQ <= D; //defines logical expression for rQ for a D flip flop
			rQnot <=~D; //defines logical expression for rQnot for a D flip flop
		end //ends if statement
	end //ends always @

	assign Q = rQ; //assigns logical expression to rQ
	assign Qnot = rQnot; //assigns logical expression to rQnot

endmodule //ends module
```

C.) II.) Top Level (D Latch)

```
module top_level_D_latch_try_2(SW, LEDR); //Defines Module

	input wire [1:0]SW; //Defines input wire bus for 2 switches
	output wire [3:0]LEDR; //Defines output wire bus for 4 LEDRS

	D_latch_try_2(SW[1], SW[0], LEDR[2], LEDR[3]); //Instantiates the D Latch module

	assign LEDR[1:0] = SW[1:0]; //Assigns LEDRS[1:0] to SW[1:0]

endmodule //ends module
```

D.) Code From Part 4 (D Flip Flop) From Our Lab

```
module d_flipflop (D, Clock, Q, Qnot); //declares module

	inout wire D, Clock; //defines input wires D, Clock
	output reg Q, Qnot; //defines output registers Q, Qnot

	always @ (posedge Clock) begin //initiates always @

		Q <= D; //defines expression for D
		Qnot <= ~D; //defines expression for Qnot

	end //ends always @

endmodule //ends module
```

E.) Code From Part 5 (T Flip Flop) From Our Lab

```
module T_Flip_Flop (T, Clock, Q, Qnext, QnextNot); //Declares module for T Flip Flop

	input wire T,Q, Clock; //defines input wires T, Q, Clock
	output reg Qnext; //defines the output register Qnext
	output QnextNot; //defines output QnextNot

	always @(posedge Clock) begin //initiate always @
		Qnext = T ^ Q;	//Defines the logical expression for the T flip flop

	end //ends always @
	assign QnextNot = ~Qnext; //assigns an expression for QnextNot

endmodule //ends module
```

F.) Code From Part 6 (JK Flip Flop) From Our Lab

```
module JK_Flip_Flop (J,K,Clock, Q, Qnot); //Declares module
```

```
        input wire J, K, Clock; //Declares input wires J, K, Clock
        output reg Q; //Declares output register Q
        output Qnot; //Declares output Qnot

        always @(posedge Clock) begin // Initiates always @ loop trigger positive clock pulse

                Q = (J&(~Q))|((~K)&Q); //Defines the logical expression for the JK Flip Flop
        end //ends always @ loop

        assign Qnot = ~ Q; //Assigns logical expression for Qnot

endmodule //End module
```