Laboratory Exercise 3

Latches, Flip-flops, and Registers

The purpose of this exercise is to investigate latches, flip-flops, and registers.

Part I

Intel® FPGAs include flip-flops that are available for implementing a user's circuit. We will show how to make use of these flip-flops in Part IV of this exercise. But first we will show how storage elements can be created in an FPGA without using its dedicated flip-flops.
Figure 1 depicts a gated RS latch circuit. Two styles of Verilog code that can be used to describe this circuit are given in Figures 2 and 3. Figure 2 specifies the latch by instantiating logic gates, and Figure 3 uses logic expressions to create the same circuit. If this latch is implemented in an FPGA that has 4-input lookup tables (LUTs), then only one lookup table is needed, as shown in Figure 4a.

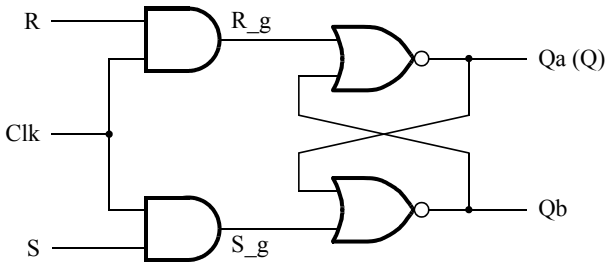


Figure 1: A gated RS latch circuit.

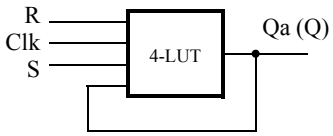
```
// A gated RS latch
module  part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    wire R_g, S_g, Qa, Qb /* synthesis keep */ ;

    and (R_g, R, Clk);
    and (S_g, S, Clk);
    nor (Qa, R_g, Qb);
    nor (Qb, S_g, Qa);

    assign Q = Qa;

endmodule
```

Figure 2: Specifying an RS latch by instantiating logic gates.

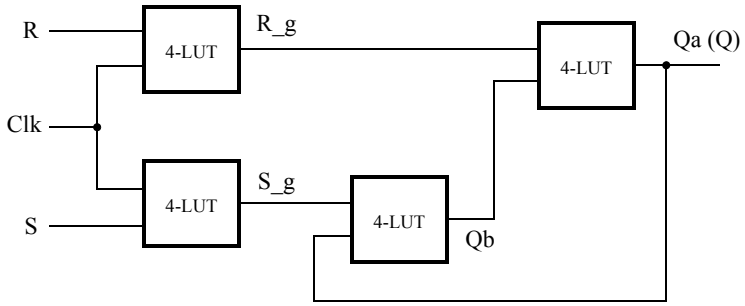
```
// A gated RS latch
module  part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    wire R_g, S_g, Qa, Qb /* synthesis keep */ ;

    assign R_g = R & Clk;
    assign S_g = S & Clk;
    assign Qa = ~(R_g | Qb);
    assign Qb = ~(S_g | Qa);

    assign Q = Qa;

endmodule
```

Figure 3: Specifying an RS latch by using Boolean expressions.

Although the latch can be correctly realized in one 4-input LUT, this implementation does not allow its internal signals, such as R_g and S_g, to be observed, because they are not provided as outputs from the LUT. To preserve these internal signals in the implemented circuit, it is necessary to include a *compiler directive* in the code. In Figures 2 and 3 the directive /* synthesis keep */ is included to instruct the Quartus® compiler to use separate logic elements for each of the signals R_g, S_g, Qa, and Qb. Compiling the code produces the circuit with four 4-LUTs depicted in Figure 4*b*.



(a) Using one 4-input lookup table for the RS latch.



(b) Using four 4-input lookup tables for the RS latch.

Figure 4: Implementation of the RS latch from Figure 1.

Create a Quartus project for the RS latch circuit as follows:

1. Create a new Quartus project for your DE-series board.
2. Generate a Verilog file for the RS latch. Use the code in either Figure 2 or Figure 3 (both versions of the

code should produce the same circuit) and include it in the project.

3. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code, and use the Technology Map Viewer tool to verify that the latch is implemented as shown in Figure 4*b*.

4. Simulate the behavior of your Verilog code by using the simulation feature provided in the Modelsim software. Use the testbench provided in the laboratory materials to drive the signals for your simulation. The procedure for using Modelsim for simulation is described in the tutorial *Introduction to Simulation of Verilog Designs Using ModelSim Graphical Waveform Editor*. An example of a vector waveform file is displayed in Figure 5. The waveforms in the figure begin by setting $Clk = 1$ and $R = 1$, which allows the simulation tool to initialize all of the signals inside of the latch to known values.

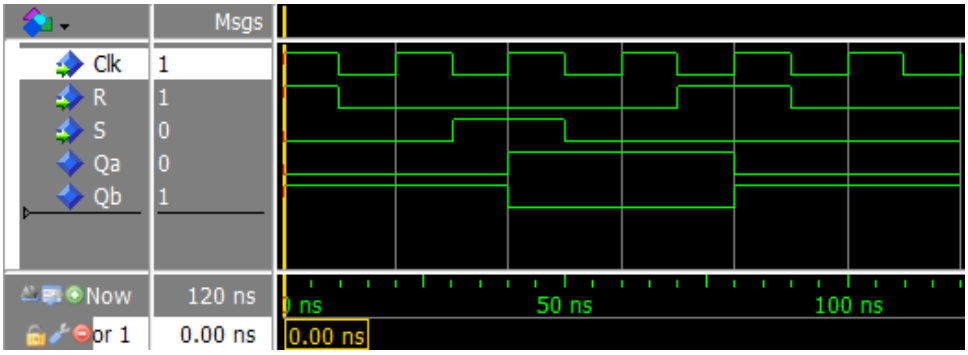


Figure 5: Simulation waveforms for the RS latch.

Part II

Figure 6 shows the circuit for a gated D latch.

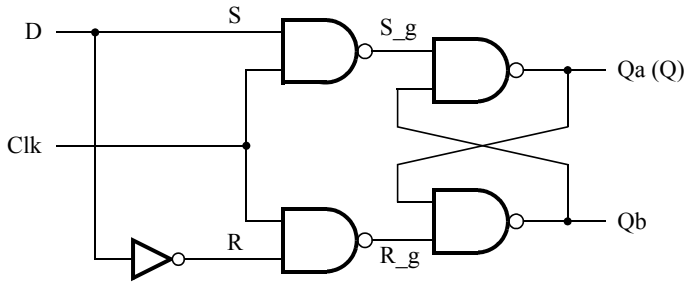


Figure 6: Circuit for a gated D latch.

Perform the following steps:

1. Create a new Quartus project. Generate a Verilog file using the style of code in Figure 3 for the gated D latch. Use the /* synthesis keep */ directive to ensure that separate logic elements are used to implement the signals R, S_g, R_g, Qa, and Qb.

2. Compile your project and then use the Technology Map Viewer tool to examine the implemented circuit.

3. Verify that the latch works properly for all input conditions by using functional simulation. Examine the timing characteristics of the circuit by using timing simulation.

4. Create a new Quartus project which will be used for implementation of the gated D latch on your DE-series board. This project should consist of a top-level module that contains the appropriate input and output ports (pins) for your board. Instantiate your latch in this top-level module. Use switch SW_0 to drive the D input of the latch, and use SW_1 as the *Clk* input. Connect the Q output to $LEDR_0$.

5. Include the required pin assignments and then compile your project and download the compiled circuit onto your DE-series board.

6. Test the functionality of your circuit by toggling the D and *Clk* switches and observing the Q output.

Part III

Figure 7 shows the circuit for a master-slave D flip-flop.

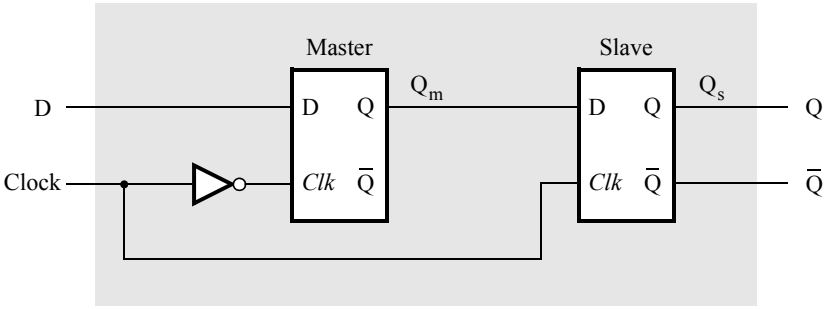


Figure 7: Circuit for a master-slave D flip-flop.

Perform the following:

1. Create a new Quartus project. Generate a Verilog file that instantiates two copies of your gated D latch module from Part II to implement the master-slave flip-flop.

2. Include in your project the appropriate input and output ports for your DE-series board. Use switch SW_0 to drive the D input of the flip-flop, and use SW_1 as the *Clock* input. Connect the Q output to $LEDR_0$.

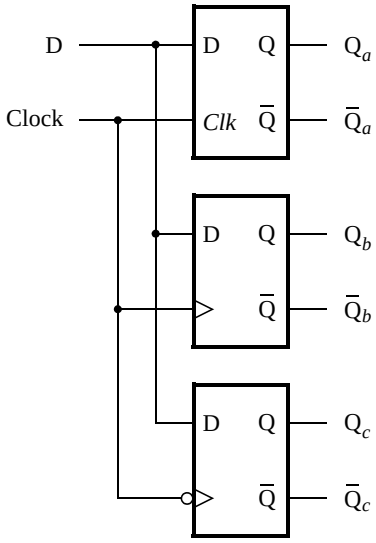
3. Include the required pin assignments and then compile your project.

4. Use the Technology Map Viewer to examine the D flip-flop circuit, and use simulation to verify its correct operation.

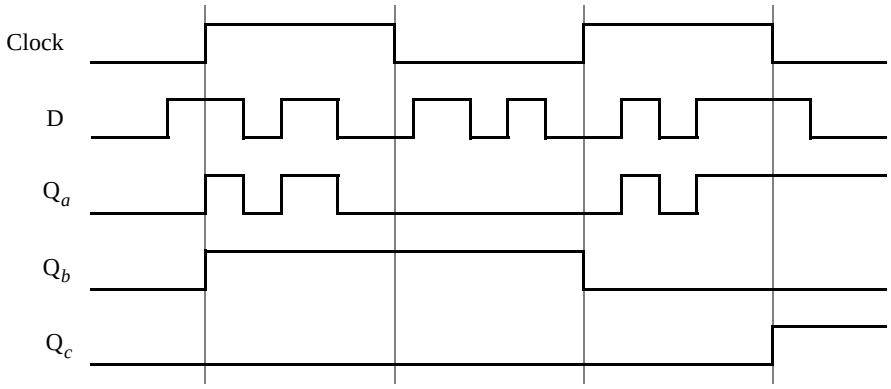
5. Download the circuit onto your DE-series board and test its functionality by toggling the D and *Clock* switches and observing the Q output.

Part IV

Figure 8 shows a circuit with three different storage elements: a gated D latch, a positive-edge triggered D flip-flop, and a negative-edge triggered D flip-flop.



(a) Circuit



(b) Timing diagram

Figure 8: Circuit and waveforms for Part IV.

Implement and simulate this circuit using the Quartus software as follows:

1. Create a new Quartus project.
2. Write a Verilog file that instantiates the three storage elements. For this part you should no longer use the /* synthesis keep */ directive from Parts I to III. Figure 9 gives a behavioral style of Verilog code that specifies the gated D latch in Figure 6. This latch can be implemented in one 4-input lookup table. Use a similar style of code to specify the flip-flops in Figure 8.
3. Compile your code and use the Technology Map Viewer to examine the implemented circuit. Verify that the latch uses one lookup table and that the flip-flops are implemented using the flip-flops provided in the target FPGA.

4. Use Modelsim to simulate the circuit you created. Use the included testbench file to specify the inputs D and *Clock* as indicated in Figure 8. Make sure that the testbench correctly instantiates the module you created and run the simulation to observe the different behavior of the three storage elements.

```
module D_latch  (D, Clk, Q);
   input D, Clk;
   output reg Q;

   always @ (D, Clk)
      if (Clk)
         Q = D;
endmodule
```

Figure 9: A behavioral style of Verilog code that specifies a gated D latch.

Part V

We wish to display the hexadecimal value of an 8-bit number A on the two 7-segment displays $HEX3-2$. We also wish to display the hex value of an 8-bit number B on the two 7-segment displays $HEX1-0$. The values of A and B are inputs to the circuit which are provided by means of switches SW_{7-0}. To input the values of A and B, first set the switches to the desired value of A, store these switch values in a register, and then change the switches to the desired value of B. Finally, use an adder to generate the arithmetic sum $S = A + B$, and display this sum on the 7-segment displays $HEX5-4$. Show the carry-out produced by the adder on LEDR[0].

1. Create a new Quartus project which will be used to implement the desired circuit on your DE-series board.
2. Write a Verilog file that provides the necessary functionality. Use KEY_0 as an active-low asynchronous reset, and use KEY_1 as a clock input.
3. Include the necessary pin assignments for the pushbutton switches and 7-segment displays, and then compile the circuit.
4. Download the circuit onto your DE-series board and test its functionality by toggling the switches and observing the output displays.