# Laboratory Exercise 6

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

**Part I**

Altera 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 Parts IV to VII 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 Figure 2. Part $a$ of the figure specifies the latch by instantiating logic gates, and part $b$ 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 3$a$.

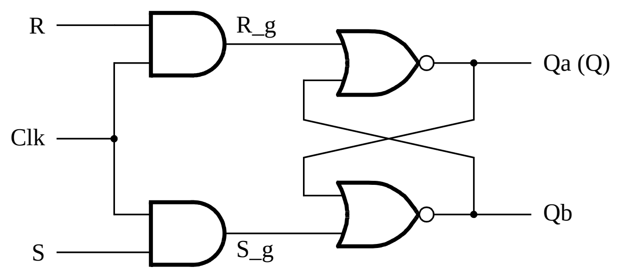


Figure 1. A gated RS latch circuit.

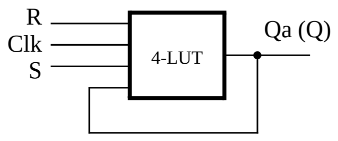
```
// A gated RS latch
module  part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    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$a$. Instantiating logic gates for the RS latch.

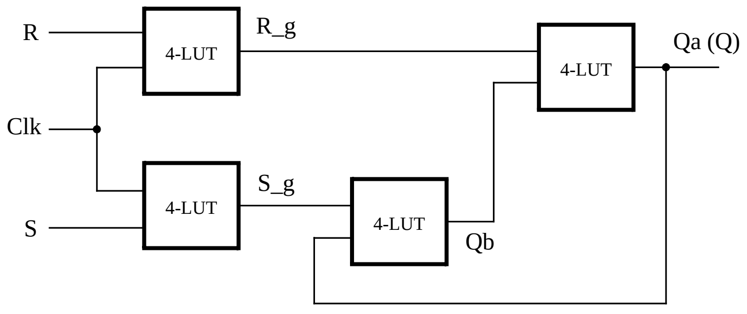
```
// A gated RS latch
module  part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    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 2$b$. Specifying the RS latch by using logic 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 Figure 2 the directive /* synthesis keep */ is included to instruct the Quartus II 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 3$b$.



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



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

Figure 3. Implementation of the RS latch from Figure 1.

Create a Quartus II project for the RS latch circuit as follows.

1. Create a new project for the RS latch. Select as the target chip the Cyclone II EP2C35F672C6, which is the FPGA chip on the Altera DE2 board.

2. Generate a Verilog file with the code in either part $a$ or $b$ of Figure 2 (both versions of the code should produce the same circuit) and include it in the project.

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

4. Create a Vector Waveform File (.vwf) which specifies the inputs and outputs of the circuit. Draw waveforms for the $R$ and $S$ inputs and use the Simulator to produce the corresponding waveforms for *R_g*, *S_g*, *Qa*, and *Qb*. Verify that the latch works as expected using both functional and timing simulation.

## Part II

Figure 4 shows the circuit for a gated D latch.

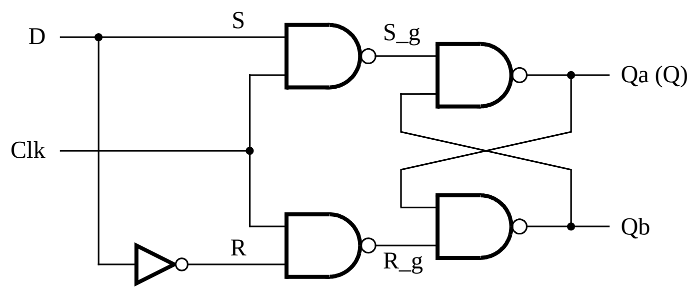


Figure 4. Circuit for a gated D latch.

Perform the following steps:

1. Create a new Quartus II project. Generate a Verilog file using the style of code in Figure 2$b$ 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. Select as the target chip the Cyclone II EP2C35F672C6 and compile the code. Use the Technology 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 II project which will be used for implementation of the gated D latch on the DE2 board. This project should consist of a top-level module that contains the appropriate input and output ports (pins) for the DE2 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. Test the functionality of your circuit by toggling the $D$ and *Clk* switches and observing the Q output.

## Part III

Figure 5 shows the circuit for a master-slave D flip-flop.

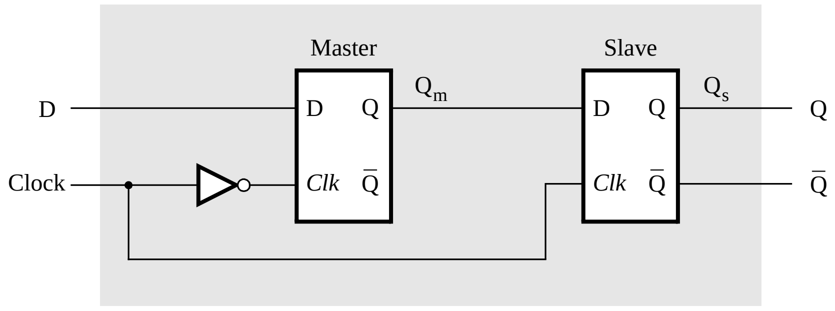


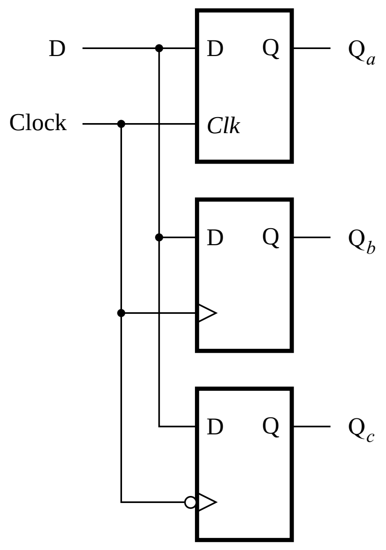
Figure 5. Circuit for a master-slave D flip-flop.

Perform the following.

1. Create a new Quartus II 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 the Altera DE2 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. Use the Technology Viewer to examine the D flip-flop circuit, and use simulation to verify its correct operation.
4. Download the circuit onto the Altera board and test its functionality by toggling the $D$ and *Clock* switches and observing the Q output.

## Part IV

Figure 6 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.



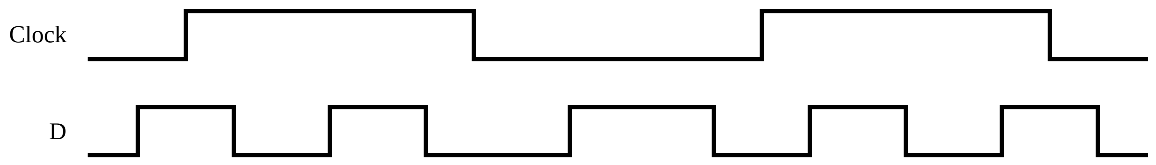


Figure 6. Circuit and waveforms for Part IV.

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

1. Create a new 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 7 gives a behavioral style of Verilog code that specifies the gated D latch in Figure 4. This latch can be implemented in one 4-input lookup table. Use a similar style of code to specify the flip-flops in Figure 6.
3. Compile your code and use the Technology 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. Create a Vector Waveform File (.vwf) which specifies the inputs and outputs of the circuit. Draw the inputs *D* and *Clock* as indicated in Figure 6. Use functional simulation to obtain the three output signals. Observe the different behavior of the three storage elements.

```
module D_latch  (input D, Clk, output Q);
   reg Q;
   always @ (D or Clk)
      if (Clk)
         Q = D;

endmodule
```

Figure 7. A behavioral style of Verilog code that specifies a gated D latch.

**Part V**

Consider the circuit in Figure 8. It is a 4-bit synchronous counter which uses four T-type flip-flops. The counter increments its count on each positive edge of the clock if the Enable signal is asserted. The counter is reset to 0 by using the Reset signal. You are to implement a 16-bit counter of this type.

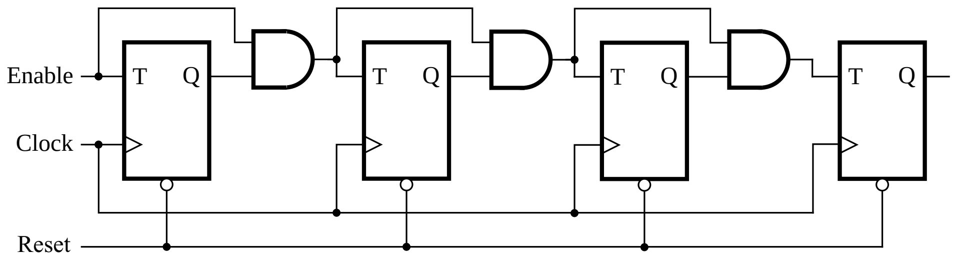


Figure 8. A 4-bit counter.

1. Write a Verilog file that defines a 16-bit counter by using the structure depicted in Figure 8, and compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, *Fmax*, at which your circuit can be operated?
2. Simulate your circuit to verify its correctness.
3. Augment your Verilog file to use the pushbutton $KEY_0$ as the *Clock* input, switches $SW_1$ and $SW_0$ as *Enable* and *Reset* inputs, and 7-segment displays *HEX3-0* to display the hexadecimal count as your circuit operates. Make the necessary pin assignments and compile the circuit.
4. Implement your circuit on the DE2 board and test its functionality by operating the implemented switches.
5. Implement a 4-bit version of your circuit and use the Quartus II RTL Viewer to see how Quartus II software synthesized your circuit. What are the differences in comparison with Figure 8?

**Part VI**

Simplify your Verilog code so that the counter specification is based on the Verilog statement:

$$Q <= Q + 1;$$

Compile a 16-bit version of this counter and compare the number of LEs needed and the *Fmax* that is attainable. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Part V.

**Part VII**

Use an LPM from the Library of Parameterized modules to implement a 16-bit counter. Choose the LPM options to be consistent with the above design, i. e. with enable and synchronous clear. How does this version compare with the previous designs?