ECE241 - Digital Systems
University of Toronto

Lab #4 - Fall 2008
Latches, Flip-flops and Registers

1. Introduction

The purpose of this laboratory exercise is to investigate latches, flip-flops and registers. In addition, you will be introduced to a debugging tool called the *logic analyzer*. With the help of the logic analyzer you will perform a more detailed analysis of the behaviour of the circuits designed in this exercise.

2. Preparation

To prepare for this exercise you will need to design circuits described below as well as read through a logic analyzer tutorial. A logic analyzer tutorial is available at:

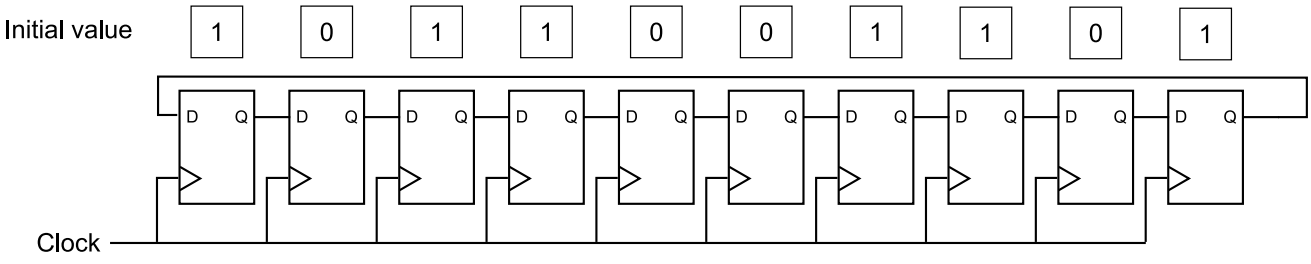
`http://www.eecg.utoronto.ca/~jayar/ece241_08F/Logic_Scope_Tutorial.pdf.`

Your preparation, to be shown to your TA at the beginning of the lab, must consist of the following:

1. The Verilog Hardware Description Language code used to implement each of the circuits described below. The code should be **printed before** you arrive in the lab.

2. Each circuit should be simulated. The simulation must clearly indicate the operation of each circuit. It is insufficient to test just one or two cases - you must use enough test cases to convince yourself (and the TA) that the circuit has been shown to be correct. It will require some thinking on your part to determine if you have done enough simulation. The simulation must be **printed** before the lab begins, with comments that explain your testing procedure.

3. Answer the following question and be prepared to explain your answer to the TA: assume you have a 10-bit shift register connected as a ring, as explained in part VI and shown below. The shift register initially contains the value 1011001101, as illustrated below. What would be the value of the shift register after one clock cycle and then after four more clock cycles?



3. In the lab

In the lab you will have to implement and test circuits described in the sections below. To simplify some of the steps a starter kit has been provided on the website:

`http://www.eecg.utoronto.ca/~jayar/ece241_08F/Lab4_starterkit.zip`

The starter kit is a ZIP archive containing a Quartus II projects for each part of the lab. Retrieve and unzip the archive into a working directory called *lab4*.

The in-lab component of your lab grade will be based on the circuit you create in **parts IV through VI** below. Please inform your TA when you have completed **all parts** and are ready to be graded.

Part I

All 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 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 3a.

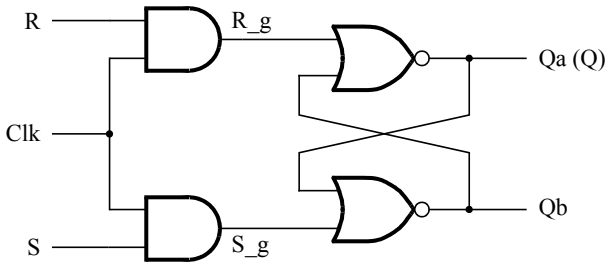


Figure 1. A gated RS latch circuit.

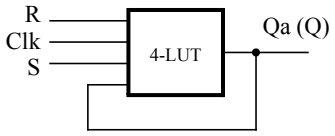
```
// A gated RS latch
module  part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    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 2a. Instantiating logic gates for the RS latch.

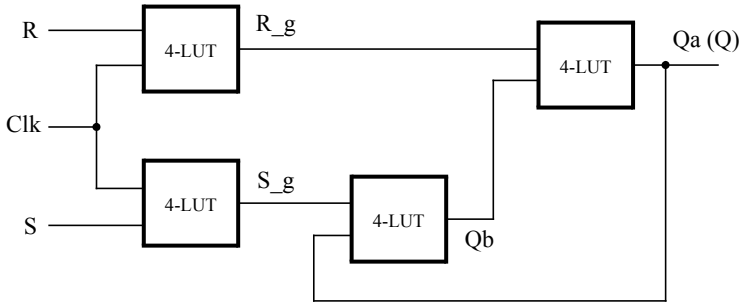
```
// A gated RS latch
module  part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    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 2b. 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 3b.



(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.

Perform the following steps to complete this part of the exercise:

1. The project for this part is provided in the starter kit. Open the project named *part1* in the *part1* subdirectory.
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. You may wish to use the Quartus II RTL Viewer tool to examine the gate-level circuit produced from the code, and the Technology Map Viewer tool to verify that the latch is implemented as shown in Figure 3b. These tools are found under the menu Tools $>$ Netlist Viewers.
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 Quartus II 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.
5. Download the circuit onto the DE2 board and test its functionality. Use the logic analyzer to capture changes of the logic signals *R_g*, *S_g*, *Qa*, and *Qb* as you toggle inputs R and S.

Part II

Figure 4 shows the circuit for a gated D latch.

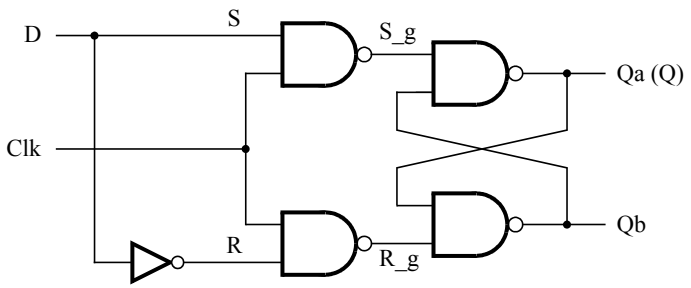


Figure 4. Circuit for a gated D latch.

Perform the following steps:

1. The project for this part is provided in the starter kit. Open the project named *part2* in the *part2* subdirectory to begin your work.
2. Generate a Verilog file using the style of code in Figure 2b for the gated D latch. Include in your Verilog code the /* synthesis keep */ directive to ensure that separate logic elements in the FPGA chip are used to implement the signals $R, S\_g, R\_g, Qa,$ and Qb.
3. Select as the target chip the Cyclone II EP2C35F672C6 and compile the code. You may wish to use the Quartus Technology Map Viewer tool to examine the implemented circuit, by using the command Tools $>$ Netlist Viewer $>$ Technology Map Viewer.
4. Verify that the latch works properly for all input conditions by using functional simulation. Also, examine the timing characteristics of the circuit by using timing simulation.
5. 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 $LEDR_0$.

6. Recompile your project and simulate the compiled circuit.
7. Test the functionality of the circuit by downloading it onto the DE2 board. Toggle the *D* and *Clk* switches and observe the output Q.

Part III

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

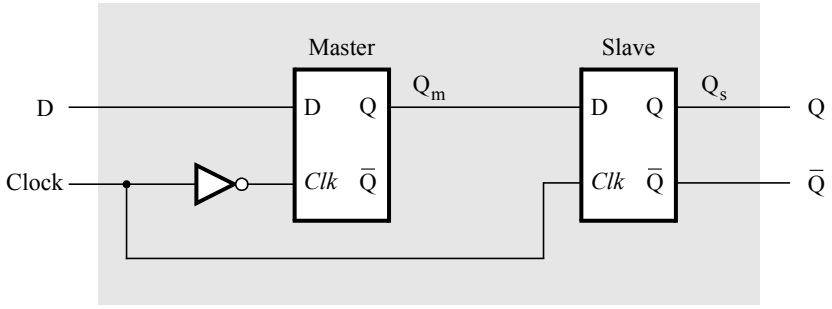


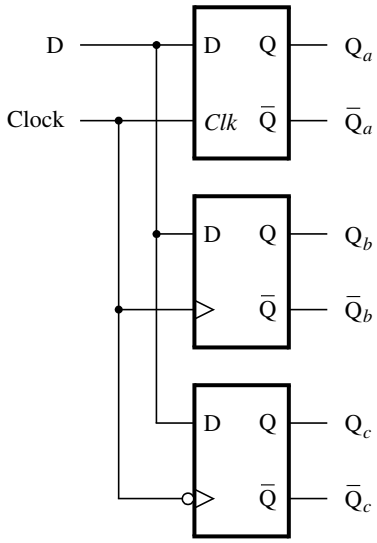
Figure 5. Circuit for a master-slave D flip-flop.

Perform the following steps:

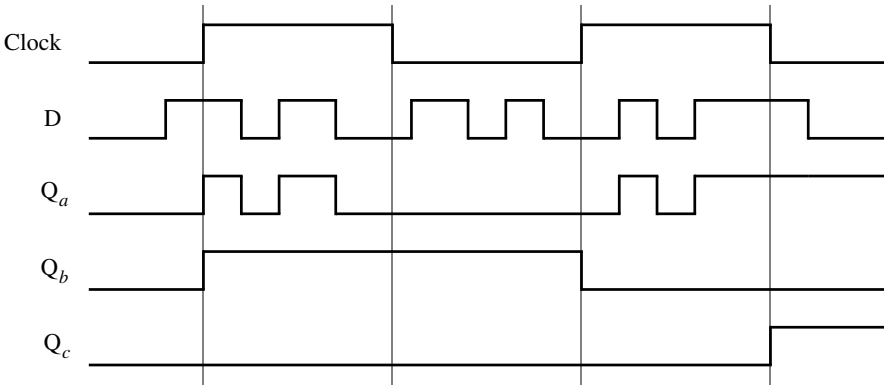
1. The project for this part is provided in the starter kit. Open the project named *part3* in the *part3* subdirectory to begin your work.
2. Generate a Verilog file that instantiates two copies of your gated D latch module from Part II to implement the master-slave flip-flop.
3. Include in your project 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 $LEDR_0$.
4. Compile your project.
5. You may wish to use the Quartus Technology Map Viewer to examine the D flip-flop circuit, using the command Tools > Netlist Viewer > Technology Map Viewer.
6. Use simulation to verify the correct operation of your circuit.
7. Download the circuit onto the DE2 board and test its functionality by toggling the *D* and *Clock* switches and observing the output Q.

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.



(a) Circuit



(b) Timing diagram

Figure 6. Circuit and waveforms for Part IV.

Implement and simulate the circuit in Figure 6 by using the Quartus II software as follows:

1. The project for this part is provided in the starter kit. Open the project named *part4* in the *part4* subdirectory to begin your work.
2. Write a Verilog file that instantiates the three storage elements. For this part you should no longer include the /* synthesis keep */ directive in your Verilog code, because you will not be describing the exact structure of flip-flops like you did in Part III. Instead, you should use a style of Verilog code that will allow the Verilog compiler (in the Quartus II software) to automatically *instantiate* flip-flops that are provided as part of the FPGA logic elements. Such Verilog code is often called *behavioral* code, because it specifies a desired circuit behavior rather than an exact circuit structure. As an example, 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 project.

4. You may wish to use the Quartus Technology Map Viewer to examine the compiled circuit, by using the command Tools > Netlist Viewer > Technology Map Viewer.

5. 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  (D, Clk, Q);
   input D, Clk;
   output reg Q;

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

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

Part V

In this part you will need to work in base 16, also known as hexadecimal, or hex for short. There sixteen digits in hexadecimal, the usual 0-9 and then the letters A through F. It is fairly easy to translate from base 2 to base 16 because each hexademical digit corresponds to exactly 4 bits in binary. A sixteen bit number translates into exactly 4 hexadecimal digits.
The goal for the circuit in this part is to display the hexadecimal value of a 16-bit number, A, on the four 7-segment displays, $HEX7-4$, and a different hexadecimal value of a 16-bit number B on the four 7-segment displays, $HEX3-0$. The values of A and B are to be input to the circuit through switches SW_{15-0}, one value at a time. This is to be done by first setting the switches to the value of A and then setting the switches to the value of B; therefore, the value of A must be stored in the circuit. You will need a 16-bit register (which should be clocked by KEY_1) to store the value of A once it has been set using SW_{15-0}.

Perform the following steps:

1. The project for this part is provided in the starter kit. Open the project named *part5* in the *part5* subdirectory to begin your work.

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 Verilog file in your project and compile the circuit.

4. Assign the pins on the FPGA to connect to the switches and 7-segment displays, as you have done in previous parts of this exercise.

5. Recompile the circuit and download it into the FPGA chip.

6. Test the functionality of your design by toggling the switches and observing the output displays.

Part VI

In this part we will create a waveform generator (a device that creates a rapidly changing digital signal with a

shape over time that can be specified) by using a 10-bit shift register. We will start by building a flip-flop with *asynchronous* set and reset inputs and then creating a circuit with 10 flip-flops connected in a ring, like that pictured in the final question of the preparation. The waveform will be observed by connecting an output of one of the flip-flops to an output pin on the DE2 board. You will then use the logic analyzez to view this signal - because the signal is changing at a high frequency, you need a instrument like the logic analyzer to view it.

Perform the following steps:

1. The project for this part is provided in the starter kit. Open the project named *part6* in the *part6* subdirectory to begin your work.

2. Using Verilog code create a positive-edge triggered flip-flop with *asynchronous* set and reset inputs. When the set input is asserted the flip-flop should immediately assume a logic value 1, while when the reset signal is asserted the flip-flop should change its output to a logic value 0. Add the created Verilog file to your project.

3. Create a *waveform generator* circuit with inputs *reset* and *clock* and an output *q*. The waveform generator will consist of a 10-bit shift register built using 10 positive-edge triggered flip-flops connected in series, forming a ring. That is, each flip-flop output Q must drive an input D of another flip-flop. The *reset* input of the waveform generator should connect to either the set or the reset input of each flip-flop. Connect the *reset* input in such a manner that when reset the 10-bit shift register will contain both 0 and 1 entries. (i.e., each flip-flop is set either to 0 or 1, so that the register will have some value other than 0. This is needed if we want to have a waveform, not just a constant value).

4. Assign pins to the design, ensuring that the clock input is driven by the 50 MHz clock on the DE2 board. Also, the output Q should be driven to a pin on the JP0 or JP1 expansion header on the DE2 board.

5. Compile and download the circuit into the DE2 board. Use the logic analyzer to display the resulting waveform. You may wish to add other outputs (and connect them to pins on the expansion header) to your circuit and display them on the logic analyzer in order to better explain the behaviour of your circuit.