

# 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. A style of VHDL code that uses logic expressions to describe this circuit is given in Figure 2. 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.

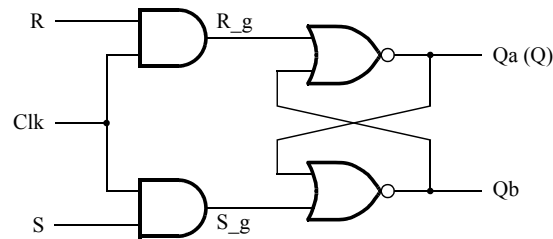


Figure 1: A gated RS latch circuit.

```
-- A gated RS latch described the hard way
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part1 IS
    PORT ( Clk, R, S : IN     STD_LOGIC;
          Q          : OUT   STD_LOGIC);
END part1;

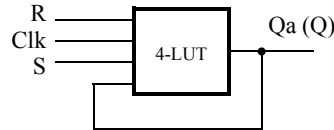
ARCHITECTURE Structural OF part1 IS
    SIGNAL R_g, S_g, Qa, Qb : STD_LOGIC ;
    ATTRIBUTE KEEP : BOOLEAN;
    ATTRIBUTE KEEP OF R_g, S_g, Qa, Qb : SIGNAL IS TRUE;
BEGIN
    R_g <= R AND Clk;
    S_g <= S AND Clk;
    Qa <= NOT (R_g OR Qb);
    Qb <= NOT (S_g OR Qa);

    Q <= Qa;

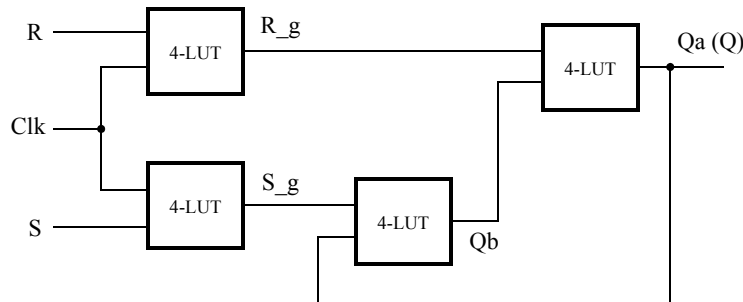
END Structural;
```

Figure 2: 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 `KEEP` is included by using a VHDL `ATTRIBUTE` statement 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 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.

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 VHDL file with the code in Figure 2 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 3b.
4. Simulate the behavior of your VHDL code by using the simulation feature provided in the Quartus software. First, create a vector waveform file (\*.vwf) using the Quartus software to specify the inputs and outputs of your circuit. Then, use the commands available in the Quartus Simulation Waveform Editor tool to run a simulation of the circuit. The procedure for using the Simulation Waveform Editor to perform a simulation is described in the tutorial *Quartus Introduction*, available on the Intel FPGA University Program website (more detailed instructions are provided in the tutorial *Introduction to Quartus Simulation*). An example of a vector waveform file is displayed in Figure 4. 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. If the waveforms provided to the simulation tool do not allow for initialization of all signals in the circuit, then the simulation tool may exit with an error condition.

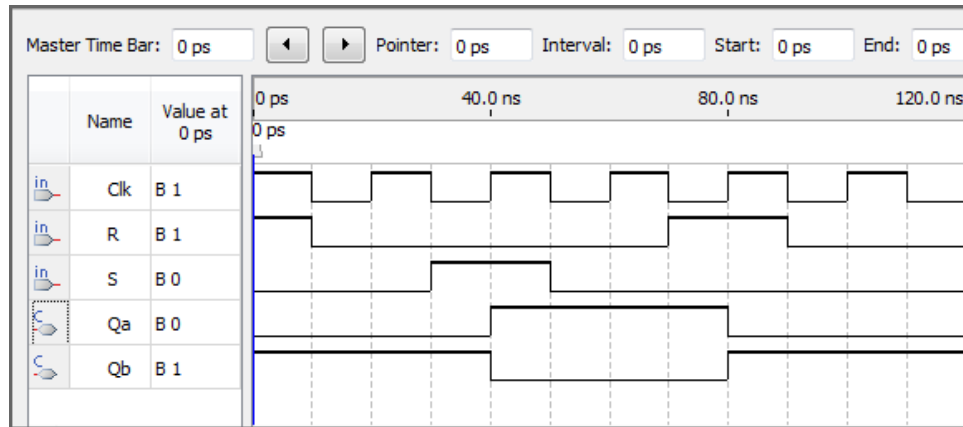


Figure 4: Simulation waveforms for the RS latch.

## Part II

Figure 5 shows the circuit for a gated D latch.

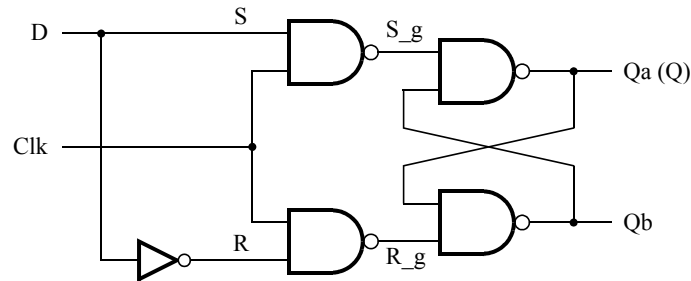


Figure 5: Circuit for a gated D latch.

Perform the following steps:

1. Create a new Quartus project. Generate a VHDL file using the style of code in Figure 2 for the gated D latch. Use the 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 6 shows the circuit for a master-slave D flip-flop.

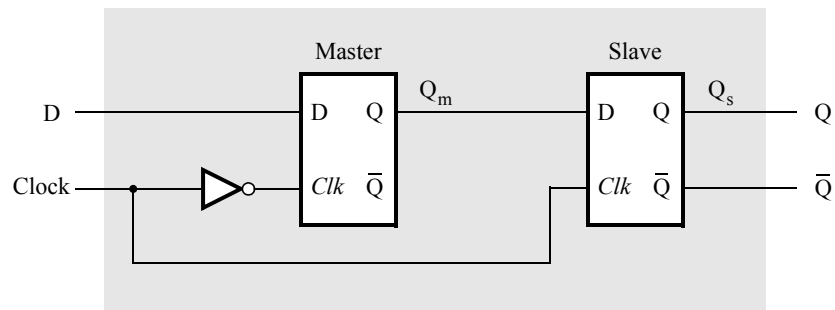


Figure 6: Circuit for a master-slave D flip-flop.

Perform the following:

1. Create a new Quartus project. Generate a VHDL 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 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 7 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.

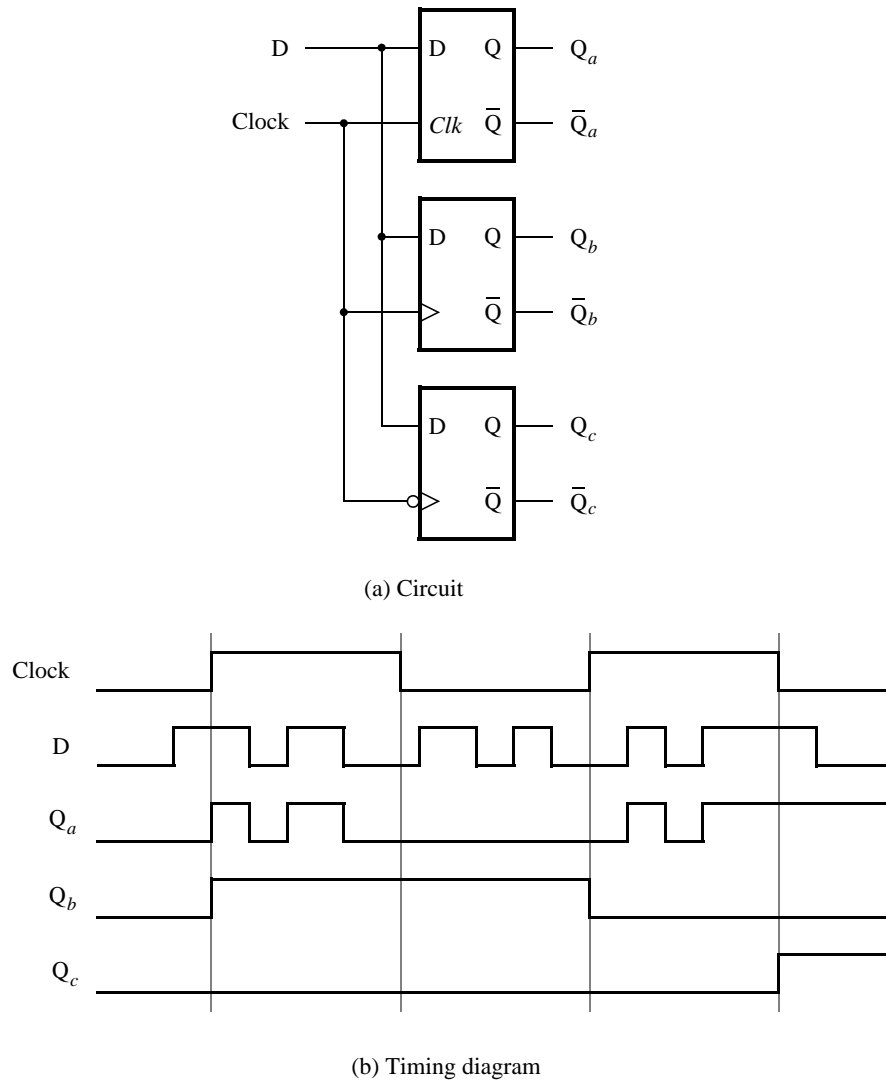


Figure 7: Circuit and waveforms for Part IV.

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

1. Create a new Quartus project.
2. Write a VHDL file that instantiates the three storage elements. For this part you should no longer use the KEEP directive (that is, the VHDL ATTRIBUTE statement) from Parts I to III. Figure 8 gives a behavioral style of VHDL code that specifies the gated D latch in Figure 5. This latch can be implemented in one 4-input lookup table. Use a similar style of code to specify the flip-flops in Figure 7.
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. Create a Vector Waveform File (.vwf) that specifies the inputs and outputs of the circuit. Draw the inputs *D* and *Clock* as indicated in Figure 7. Use functional simulation to obtain the three output signals. Observe the different behavior of the three storage elements.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY latch IS
    PORT ( D, Clk : IN    STD_LOGIC ;
          Q       : OUT  STD_LOGIC ) ;
END latch ;

ARCHITECTURE Behavior OF latch IS
BEGIN
    PROCESS ( D, Clk )
    BEGIN
        IF Clk = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 8: A behavioral style of VHDL 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<sub>7-0</sub>*. 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 VHDL file that provides the necessary functionality. Use *KEY<sub>0</sub>* as an active-low asynchronous reset, and use *KEY<sub>1</sub>* 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.

Copyright © 1991-2016 Intel Corporation. All rights reserved. Intel, The Programmable Solutions Company, the stylized Intel logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Intel Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Intel products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Intel warrants performance of its semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel Corporation. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.