Laboratory 2:
ADDER AND FLIP-FLOP

OBJECTIVES

- The purpose of this lab is to learn how to connect simple input (switches) and output devices (LEDs and 7-segment) to an FPGA chip and implement a circuit that uses these devices.
- Design some combinational circuits that can perform addition and binary-coded-decimal (BCD) addition.
- Besides, investigate latches, flip-flops, and registers.

PREPARATION FOR LAB 2

- Finish Pre Lab 2 at home.
- Students have to simulate all the exercises in Pre Lab 2 at home. All results (codes, waveform, RTL viewer, …) have to be captured and submitted to instructors prior to the lab session. *If not, students will not participate in the lab and be considered absent this session.*

REFERENCE

1. Intel FPGA training

Laboratory 2:

ADDER AND FLIP-FLOP

EXPERIMENT 1

Objective: Known how to program 4-bit adder.

Requirement: Figure 1 shows how four instances of the full adder (FA) module can be used to design a circuit that adds two four-bit numbers. This type of circuit is usually called a *ripple-carry* adder, because of the way that the carry signals are passed from one full adder to the next. Write VHDL code that implements this circuit, as described below.

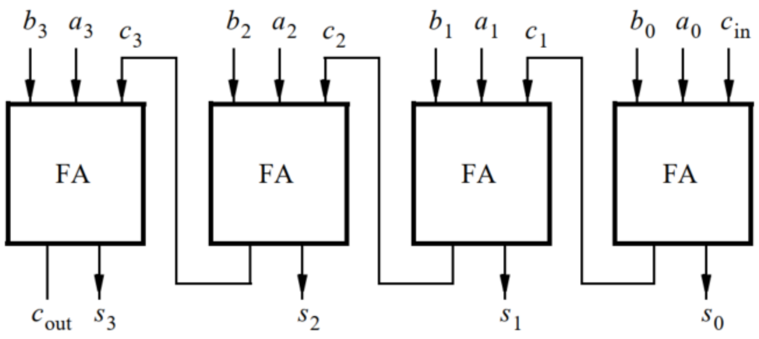


Figure 1: Four-bit ripple-carry adder circuit.

Instruction:

1. In your excersie 1, pre lab 2 , the full adder (FA) subcircuit was written. Write a top-level VHDL entity that instantiates four instances of this FA to describe the circuit given in Figure 1.
2. Use switches *SW*7-4 and *SW*3-0 to represent the inputs *A* and *B*, respectively. Use *SW*8 for the carry-in *cin* of the adder. Connect the outputs of the adder, *cout* and *S*, to the red lights LEDR.
3. Compile the project, and then download the resulting circuit into the FPGA chip. Test your circuit by trying different values for numbers *A*, *B*, and *cin*.

Check: Your report has to show two results:

- ➢ The waveform to prove the circuit works correctly.
- ➢ The result of RTL viewer.

Laboratory 2:

ADDER AND FLIP-FLOP

EXPERIMENT 2

Objective: Known how to program BCD adder.

Requirement: Design a circuit that has two decimal digits, X and Y , as inputs. Each decimal digit is represented as a 4-bit number. In technical literature this is referred to as the *binary coded decimal* (BCD) representation.

You are to design a circuit that adds the two BCD digits. The inputs to your circuit are the numbers X and Y , plus a carry-in, *cin*. When these inputs are added, the result will be a 5-bit binary number. But this result is to be displayed on 7-segment displays as a two-digit BCD sum $S1S0$. For a sum equal to zero you would display $S1S0$ = 00, for a sum of one $S1S0$ = 01, for nine $S1S0$ = 09, for ten $S1S0$ = 10, and so on. Note that the inputs X and Y are assumed to be decimal digits, which means that the largest sum that needs to be handled by this circuit is $S1S0 = 9 + 9 + 1 = 19$.

Instruction:

- Use the four-bit adder circuit from experiment 1 to produce a four-bit sum and carry-out for the operation $X + Y$.
  A good way to work out the design of your circuit is to first make it handle only sums $(X + Y) \leq 15$. With these values, your circuit from experiment 5 (lab 1) can be used to convert the 4-bit sum into the two decimal digits S1S0. Then, once this is working, modify your design to handle values of $15 < (X + Y) \leq 19$. One way to do this is to still use your circuit from experiment 5 (lab 1), but to modify its outputs before attaching them to the 7-segment display to make the necessary adjustments when the sum from the adder exceeds 15.
- Use switches *SW*7-4 and *SW*3-0 for the inputs X and Y , respectively, and use *SW*8 for the carry-in. Connect the four-bit sum and carry-out produced by the operation $X + Y$ to the red lights LEDR. Display the BCD values of X and Y on the 7-segment displays *HEX5* and *HEX3*, and display the result $S1S0$ on *HEX1* and *HEX0*.
- Since your circuit handles only BCD digits, check for the cases when the input X or Y is greater than nine. If this occurs, indicate an error by turning on the red light *LEDR*9.

Check: Your report has to show two results:

Laboratory 2:

ADDER AND FLIP-FLOP

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.

Laboratory 2:

ADDER AND FLIP-FLOP

EXPERIMENT 3

Objective: Known how to program BCD adder.

Requirement: In previous experiment you created VHDL code for a BCD adder. A different approach for describing the adder in VHDL code is to specify an algorithm like the one represented by the following pseudo-code:

```
1    $T_0 = A + B + c_0$
2    **if** ($T_0 > 9$) **then**
3        $Z_0 = 10$;
4        $c_1 = 1$;
5    **else**
6        $Z_0 = 0$;
7        $c_1 = 0$;
8    **end if**
9    $S_0 = T_0 - Z_0$
10   $S_1 = c_1$
```

It is reasonably straightforward to see what circuit could be used to implement this pseudo-code. Lines 1 and 9 represent adders, lines 2-8 correspond to multiplexers, and testing for the condition $T_0 > 9$ requires comparators. You are to write VHDL code that corresponds to this pseudo-code. Note that you can perform addition operations in your VHDL code instead of the subtraction shown in line 9. The intent of this part of the exercise is to examine the effects of relying more on the VHDL compiler to design the circuit by using IF-ELSE statements along with the VHDL > and + operators. Perform the following steps:

Instruction:

- Use switches *SW*7-4 and *SW*3-0 for the inputs *A* and *B*, respectively, and use *SW*8 for the carry-in. The value of *A* should be displayed on the 7-segment display *HEX5*, while *B* should be on *HEX3*. Display the BCD sum, *S*1*S*0, on *HEX1* and *HEX0*.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.
- Use the Quartus RTL Viewer tool to examine the circuit produced by compiling your VHDL code. Compare the circuit to the one you designed in experiment 2.

Laboratory 2:

ADDER AND FLIP-FLOP

EXPERIMENT 4

Objective: Known how to program a master-slave D flip-flop.

Requirement: Write a VHDL code for a master-slave D flip-flop which is shown Figure 2.

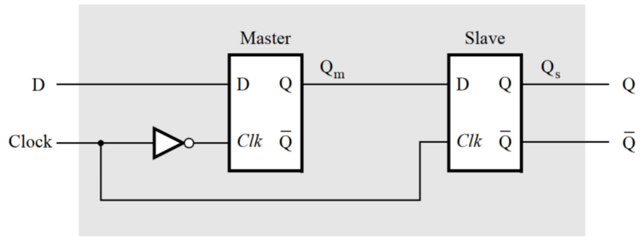


Figure 2: Circuit for a master-slave D flip-flop.

Instruction:

- In your exercise 2, pre lab 2, you wrote the gated D latch module. Write a VHDL code that instantiates two copies of this latch to implement the master-slave flip-flop.
- Use switch SW0 to drive the D input of the flip-flop, and use SW1 as the Clock input. Connect the Q output to LEDR0.
- Use RTL Viewer to examine the D flip-flop circuit, and use simulation to verify its correct operation.
- Download the circuit onto your DE-10 board and test its functionality by toggling the *D* and *Clock* switches and observing the Q output.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.

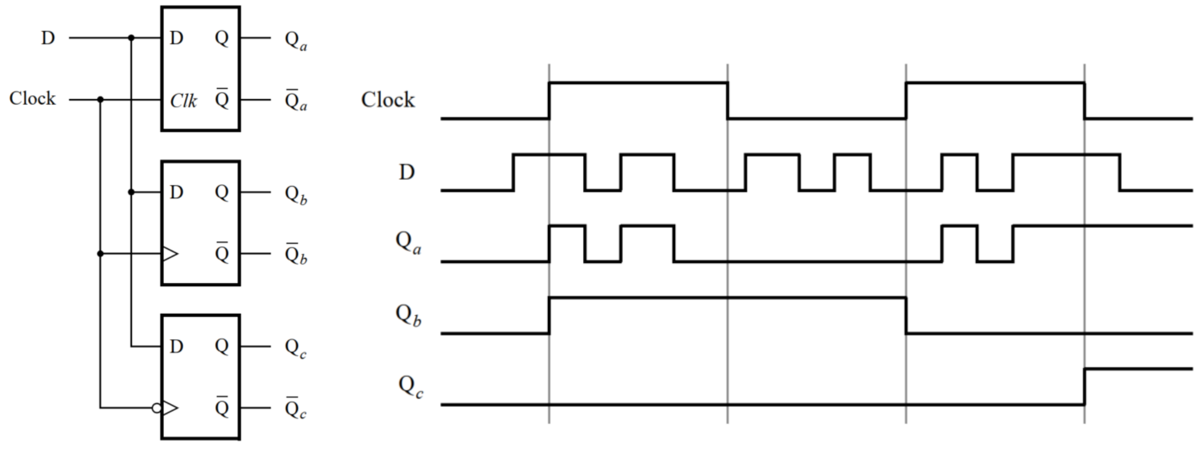
Laboratory 2:

ADDER AND FLIP-FLOP

EXPERIMENT 5

Objective: Compare the different behavior of the three storage elements: a gated D latch, a positive-edge triggered D flip-flop, and a negative-edge triggered D flip-flop.

Requirement: Compare the different behavior of the three storage elements in Figure 3.



(a) Circuit (b) Timing diagram

Figure 3: Circuit and waveforms for experiment 5.

Instruction:

- 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)**. The following code gives a behavioral style of VHDL code that specifies the gated D latch. This latch can be implemented in one 4-input lookup table. Use a similar style of code to specify the flip-flops in Figure 2.

```
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 ;
```

- Compile your code and use the RTL 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.
- Use functional simulation to draw the inputs *D* and *Clock* as indicated in Figure 3b, then obtain the three output signals. Observe the different behavior of the three storage elements.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.
- Show the different between three storage elements.

Laboratory 2:
ADDER AND FLIP-FLOP

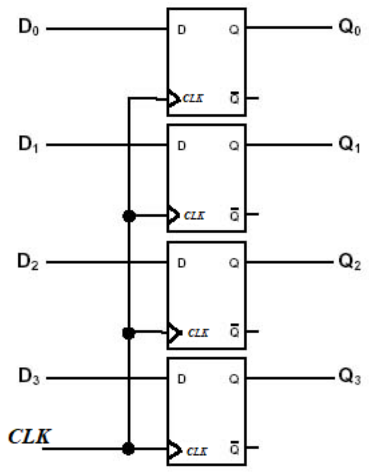
EXPERIMENT 6

Objective: Apply all the previous experiments.

Requirement: We wish to display the hexadecimal value of an 8-bit number *A* on the two 7-segment displays *HEX*3-2. We also wish to display the hex value of an 8-bit number *B* on the two 7-segment displays *HEX*1-0. 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 *HEX*5 - 4. Show the carry-out produced by the adder on LEDR(0).

Instruction:

- A simple structure of a 4-bit register using D-FF is shown below



Modify the VHDL code for D-FF in experiment 5 to construct a register.

- Write a VHDL file that provides the necessary functionality. Use *KEY*0 as an active-low asynchronous reset, and use *KEY*1 as a clock input.
- Include the necessary pin assignments for the pushbutton switches and 7-segment displays, and then compile the circuit.

Check: Your report has to show two results: the waveform and the result of RTL viewer.