Middle East Technical University - Department of Computer Engineering

CENG 232

Logic Design

Spring '2018-2019

Lab 3

Part 1 Due Date: 14 April 2019, Sunday, 23:59
Part 2 Due Date: 21 April 2019, Sunday, 23:59
No late submissions

1 Introduction

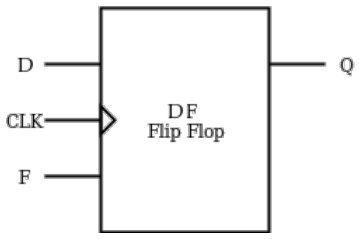
This assignment aims to make you familiar with Verilog language, related software tools, and the FPGA boards. There are two parts in this assignment. The first part is a Verilog simulation of an imaginary flip-flop design, which you are required to implement and test on your own. The second part consists of simulation and implementation (on FPGA) of MarsAdventure system.

2 Part 1: DF - Flip Flop (Warming - 25 pts)

You are given a specification of a new type of flip-flop, and a new chip that uses the flip-flop. This is an individual part. Your task is to implement these in Verilog, and prove that they work according to their specifications by simulation (this part will only be tested with testbenches not with FPGAs).

2.1

Implement the following DF flip-flop in Verilog with respect to the provided truth table. An DF flip-flop has 4 operations when inputs D and F are: 00 (set to 1), 01 (no change), 10 (set to 0), 11 (complement). Please note that the DF Flip-Flop changes its state only at rising clock edges.



2.2

Implement the following chip that contains two DF flip-flops which has output Y.

Use the following module definitions for the modules:

```
module df(input d, input f, input clk, output reg q)
module icplusplus(input d0, input f0, input d1, input f1, input clk, output q0, output q1, output y)
```

Table 1: DF - flip-flop truth table.

D	F	Q	Qnext
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

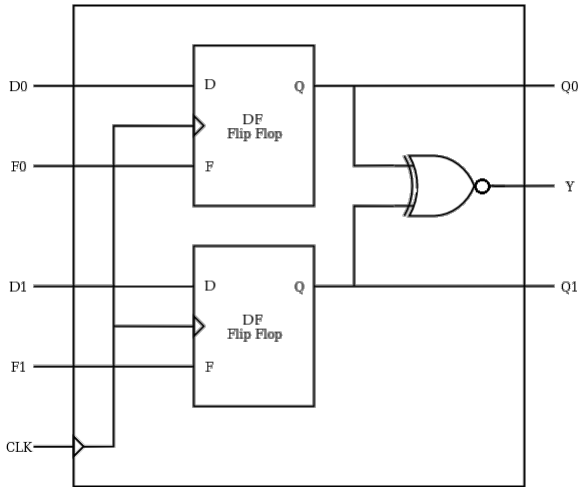


Figure 1: icplusplus module, inputs and outpus.

2.3 Simulation

A sample testbench for df-flipflop module will be provided to you. It is your responsibility to extend the testbench, and also to write a testbench for icplusplus module.

2.4 Deliverables

- Implement both modules in a single Verilog file: lab3_1.v. Do NOT submit your testbenches. You can share your testbenches on the ODTUClass discussion.
- Submit the file through the ODTUClass system before the given deadline. **April 14, 2019, 23:59**
- Any kind of cheating is not allowed.

3 Part 2: F1 Career (75 pts)

In this part of the homework, you will test your implementations on FPGAs (not with testbenches). This is an individual part. You will only share FPGAs for testing. Everyone will send their own verilog file and any kind of cheating is not allowed.

3.1 Problem Definition

Take a seat on F1...

In all around the world, there may be one job for only 20 positions available.The name of this job is f1 driver. Imagine Redbull Racing has canceled the contract with one of their drivers, namely Max Verstappen. After seeing this news, you decide to apply Redbull Racing to become their new young driver. But it is not easy to be appointed.

You should succeed in various tests. After passing all the tests, you come to the last step. That is the intelligence test. In this test, you are going to solve their mathematical test. To impress them, you decided to design a hardware that helps you to solve their problem. They are going to give you a number and you will apply operations to that number.

In this problem the Redbull Racing test committee will provide you some numbers and according to provided options you will apply some mathematical operations to these numbers. Each **number** provided consist of 4 bits. There are two different **operations** to be done on this number. You will apply different operations for **prime numbers** and **non-prime numbers**.

Here is the detailed specification list that Redbull Racing provided to you. To be successful you have to be highly careful to obey all of the specifications:

1. Each **number** given by Redbull racing committee consists of a 4-bit number.
2. If the number is only divisible by itself and one it is a **prime number** and otherwise it is a **non-prime number**.
3. The user can choose the operation he/she wants by using the **selection** switch. If the **selection** is 0, the system will execute the **prime number** operation. If the **selection** is 1, the system will execute the **non-prime number** operation.
4. In the time of executing the **prime number operation**, the system will basically find the *next* or *previous* **prime number** of the given **prime number** according to the **mode** of the system. The provided number should be written to **digit0** variable (*it triggers the right most seven segment display*) and the next or the previous number should be written to **digit1** variable (*it triggers the second right most seven segment display*).
5. If the user *selects* the **prime number operation** and the **mode** is 1, the system will find **the next prime number** *after* the currently provided prime number. If the next prime number is larger than the maximum number that can be represented by a 4-bit number then the system returns the minimum prime number possible. (i,e. if the current prime number is 7 the next prime number will be 11. If the current prime number is 13, the next prime number will be 2 because the next prime number is larger than 15 (*maximum number that can be represented by a 4-bit number*)).
6. After each operation in the **mode** 1 of the prime number operation, the **count1** (*it triggers the left most seven segment display*) variable should be increased by one (we do not check if that number is provided before). **count1** variable basically counts the number of prime number operations executed in the mode 1. Unfortunately **count1** is a 1 digit number, and after it reaches 9 it has to start counting from 0. (i.e, $7, 8, 9, 0, 1 \cdots$).
7. If the user *selects* the **prime number operation** and the **mode** is 0, the system finds **the previous prime number** *before* the currently provided prime number. If there is no previous prime number exists, the system returns the maximum prime number possible. (i,e. if the current prime number is 7 the previous

prime number will be 5. If the current prime number is 2 (*smallest possible prime number*), the next prime number will be 13 (*maximum possible prime number with 4-bits*) because there is no previous prime number).

8. After each operation in the **mode** 0 of the prime number operation, the **count0** (*it triggers the second left most seven segment display*) variable should be increased by one (we do not check if that number is provided before). **count0** variable basically counts the number of prime number operations executed in the mode 0. Unfortunately **count0** is a 1 digit number, and after it reaches 9 it has to start counting from 0. (i.e, $7, 8, 9, 0, 1 \cdots$).

9. If **the prime number operation** (selection=0) is selected and **a non-prime number** is given to the system, then the system should *activate* **the warning led** and the system must not do any operation (*all the other values should stay same without any change*). After the warning situation disappears **the warning led** must be *turned off* again.

10. In the time of executing the **non-prime number operation**, the system will basically shift the provided number to the *left* or *right* according to the **mode** of the system. The provided number should be written to **digit0** variable (*it triggers the right most seven segment display*) and the shifted number should be written to **digit1** variable (*it triggers the second right most seven segment display*).

11. If the user *selects* the **non-prime number operation** and the **mode** is 1, the system will **shift the number to left** by one. After the shift, the left most bit of **the number** will be discarded and the right most bit of **the number** will be instantiated as 0. (i.e, 1010 will become 0100 after the shift).

12. After each operation in the **mode** 1 of the non-prime number operation, the **count1** (*it triggers the left most seven segment display*) variable should be increased by one (we do not check if that number is provided before). **count1** variable basically counts the number of non-prime number operations executed in the mode 1. Unfortunately **count1** is a 1 digit number, and after it reaches 9 it has to start counting from 0. (i.e, $7, 8, 9, 0, 1 \cdots$).

13. If the user *selects* the **non-prime number operation** and the **mode** is 0, the system will **shift the number to right** by one. After the shift, the right most bit of **the number** will be discarded and the left most bit of **the number** will be instantiated as 0. (i.e, 1010 will become 0101 after the shift).

14. After each operation in the **mode** 0 of the non-prime number operation, the **count0** (*it triggers the second left most seven segment display*) variable should be increased by one (we do not check if that number is provided before). **count0** variable basically counts the number of non-prime number operations executed in the mode 1. Unfortunately **count0** is a 1 digit number, and after it reaches 9 it has to start counting from 0. (i.e, $7, 8, 9, 0, 1 \cdots$).

15. Realize that both operations use the same **count** and **digit** variables. To not lose the count information of both operations, you should store them somewhere and initialize to **count** variables when that operation is called. You do not have to store the last values of the **digit** variables.

16. If **the non-prime number operation** (selection=1) is selected and **a prime number** is given to the system, then the system should *activate* **the warning led** and the system must not do any operation (*all the other values should stay same without any change*). After the warning situation disappears **the warning led** must be *turned off* again.

17. If the **selection** is switched from 0 to 1 and **a non-prime number** is provided to the system then the **count** variables should be updated with last values of **non-prime number operation counts**. And the **digit** values must be filled with the non-prime number operation values.

18. If the **selection** is switched from 1 to 0 and **a prime number** is provided to the system then the **count** variables should be updated with last values of **prime number operation counts**. And the **digit** values must be filled with the prime number operation values.

19. Initially, **count1**, **count0**, **digit1**, **digit0** must be set to 0.

20. If the **clear** switch is provided as 1, then all of the variables (**count1**, **count0**, **digit1**, **digit0** and **warning**) must be set to 0. And also the system must not do any operation while the **clear** switch is *on*. If the **clear** switch is set to 0, then the system will behave normally.

3.2 Sample Input/Output

- The values in **Current State** column, which are seperated by "**,**" are defined as: **number**, **clear**, **mode**, **selection**, **count1**, **count0** , **digit1**, **digit0** respectively.
- The values in **Next State** column, which are seperated by "**,**" are defined as: **count1**, **count0**, **digit1**, **digit0**, **warning**, respectively.

Table 2: Sample inputs and outputs.

current state	CLK	next state
1011, 0, 0, 0, 0, 0, 0, 0	↑	0, 1, 7, b, 0
0111, 0, 1, 0, 0, 1, 7, b	↑	1, 1, b, 7, 0
0111, 0, 1, 1, 1, 1, b, 7	↑	1, 1, b, 7, 1
0110, 0, 1, 1, 1, 1, b, 7	↑	1, 0, C, 6, 0
0101, 0, 0, 0, 1, 0, C, 6	↑	1, 2, 3, 5, 0
1111, 1, 1, 1, 1, 2, 3, 5	↑	0, 0, 0, 0, 0

In line 5 we stored a prime number, and the previous values for the prime number counts were 1 1 (we switched from non-prime to prime, their count values are different). Because of that, the new values of the counts become 1 2.

3.3 Input/Output Specifications

- **number** represents 4-bit code.
- **CLK** is the clock input for the module.
- **selection** is used for the selection of operation (prime/non-prime).
  selection = 0 ⇒ prime operation
  selection = 1 ⇒ non-prime operation
- **mode** is used to choose the direction of the operations
  mode = 0 ⇒ the previous prime number will be found and the count0 will be incremented (selection=0)
  mode = 1 ⇒ the next prime number will be found and the count1 will be incremented (selection=0)
  mode = 0 ⇒ the number will be shifted to right and the count0 will be incremented (selection=1)
  mode = 1 ⇒ the number will be shifted to left and the count1 will be incremented (selection=1)
- **digit0** will show the current prime number and **digit1** will show the next or previous prime number. (selection =0)
- **digit0** will show the current non-prime number and **digit1** will show the left shifted or right shifted non-prime number. (selection =1)
- **count0** will show the number of previous prime numbers counted and **count1** will show the number of next prime numbers counted. (selection =0)
- **count0** will show the number of right shifted non-prime numbers and **count1** will show the number of left shifted non-prime numbers. (selection =1)
- **warning** shows the warning situations.
  warning = 0 ⇒ no warning occured.
  warning = 1 ⇒ warning occured.
- **clear** is used for performing clearing operation.
  clear = 0 ⇒ no clear operation will be done.
  clear = 1 ⇒ all of the variables (**count1**, **count0**, **digit1**, **digit0** and **warning**) must be set to 0.

3.4 FPGA Implementation

You will be provided with a Board232.v file (and a ready-to-use Xilinx project), which will bind inputs and outputs of the FPGA board with your Verilog module. You are required to test your Verilog module on the FPGA boards.

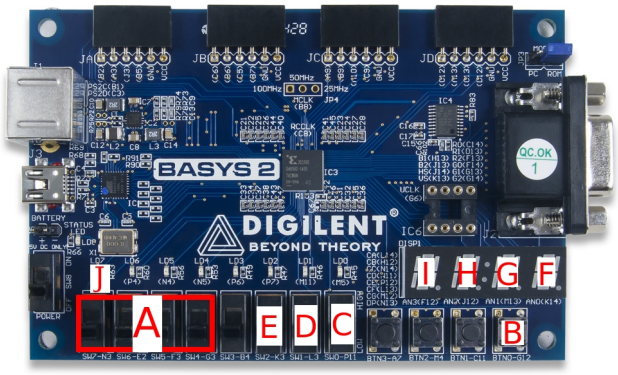
Table 3: Inputs and output variables

Name	Type	Size
number	Input	4 bits
Clock (CLK)	Input	1 bit
selection	Input	1 bit
mode	Input	1 bit
clear	Input	1 bit
digit1	Output	8 bits
digit0	Output	8 bits
count1	Output	8 bits
count0	Output	8 bits
warning	Output	1 bit

Table 4: Button descriptions.

Name	FPGA Board	Description
number	SW7, SW6, SW5, SW4	Left-most 4 switches (A)
Clock (CLK)	BTN0	Right-most button (B)
selection	SW0	Right-most switch (C)
mode	SW1	The switch next to SW0 (D)
clear	SW2	The switch next to SW1 (E)
digit0	7-segment display	Right-most 7-segment display (F)
digit1	7-segment display	Second right-most 7-segment display (G)
count0	7-segment display	Left-most 7-segment display (H)
count1	7-segment display	Second left-most 7-segment display (I)
warning	LD7	Left-most led (J)

Figure 2: Board with the button informations.



3.5 Deliverables

- Implement your module in a single Verilog file: lab3_2.v. Do NOT submit your testbenches. You can share your testbenches on the ODTUClass discussion.
- Submit the file through the ODTUClass system before the given deadline. **April 21, 2019, 23:59**
- **This is not a group part!!! All the work will be done individually. Any kind of cheating is not allowed.**
- Use the ODTUClass discussion for any questions regarding the homework.