Middle East Technical University - Department of Computer Engineering

CENG 232

Logic Design

Spring '2022-2023

Lab 3

Part 1 Due Date: 9 May 2023, Tuesday, 23:55
No late submissions

1 Introduction

This assignment aims to make you familiar with Verilog language and the related software tools. There are two parts in this assignment. The first part is a Verilog simulation of an imaginary flip-flop design. The second part consists of the implementation of a Vending Machine System.

2 Part 1: Warm-up (50 pts)

You are given a specification of a new type of flip-flop, and a new chip that uses this flip-flop. Your task is to implement these in Verilog, and prove that they work according to their specifications by using testbenches.

2.1 BH Flip-Flop Module

Implement the following BH flip-flop in Verilog with respect to the provided characteristic table. BH flip-flop has 4 operations when inputs B and H are: **00 (complement), 01 (set to 1), 10 (set to 0), 11 (no change)**. Please note that the BH flip-flop changes its state **only at rising clock edges**. Initially, the output of the flip-flop should be set to one.

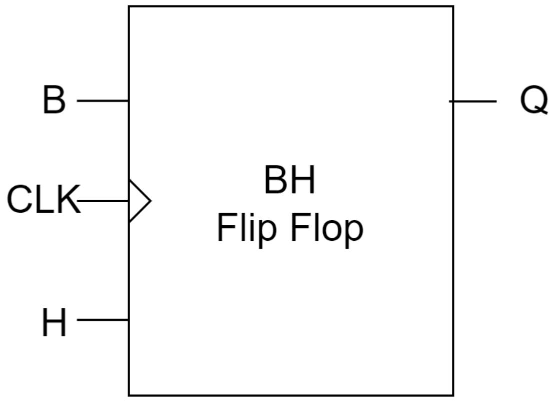

Figure 1: BH Flip-flop diagram

B	H	Q	Q_{next}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 1: BH Flip-flop characteristic table

2.2 ic1337 Module

Implement ic1337 chip given in Figure 2 that contains two BH flip-flops and has A_0, A_1, A_2, and clk as inputs; Q_0, Q_1 and Z as outputs. Please note that A_0', A_1' and A_2' are the complements of A_0, A_1 and A_2, respectively. Use the following module definitions for the modules:

```
module bh(input B, input H, input clk, output reg Q)
module ic1337(input A0, input A1, input A2, input clk, output Q0, output Q1, output Z)
```

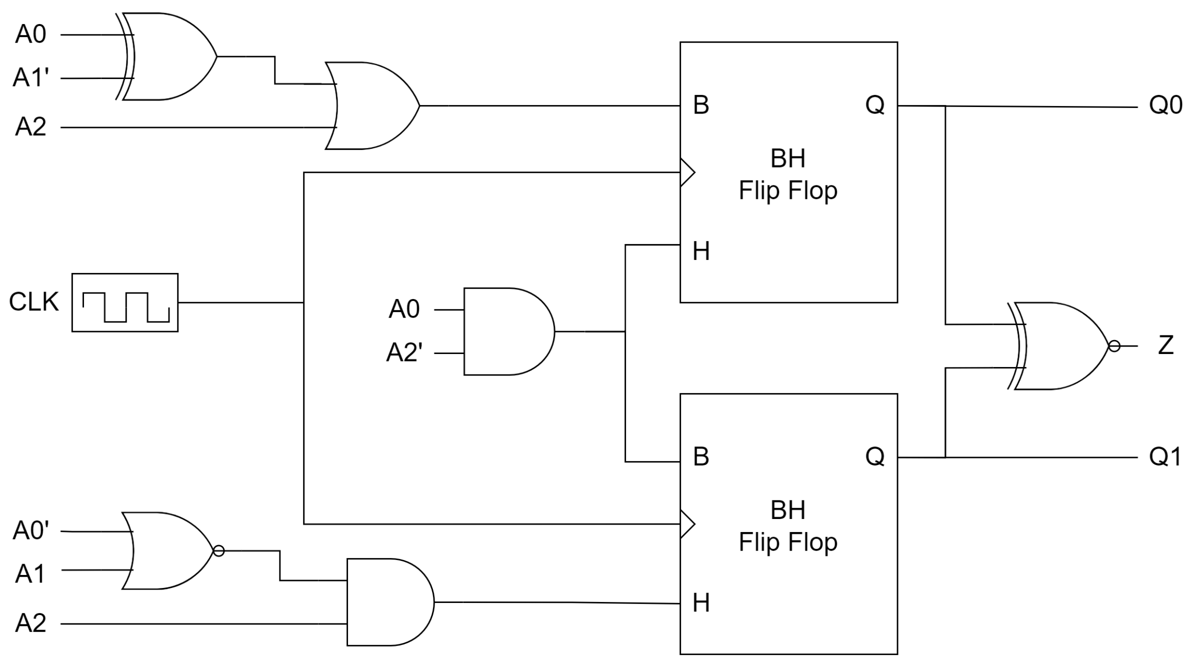

Figure 2: ic1337 module, inputs and outputs.

2.3 Simulation

A sample testbench for BH flip-flop module will be provided to you. You may want to extend the testbench, and also write a testbench for ic1337 module to test different scenarios.

2.4 Deliverables

- Implement both modules in a single Verilog file. Upload only **lab3_1.v** file to ODTUClass system.
  Do **NOT** submit your testbenches. You can share your testbenches on ODTUClass discussion page.
- Submit the file through the ODTUClass system before the deadline given at the top.
- This is an individual work, any kind of cheating is not allowed.
- Implementations will be tested using custom testbenches in a blockbox fashion. We will use **Xilinx** and **iverilog** to conduct these tests. Please make sure that your code compiles over these tools.
- for iVerilog compilation please use **-g2005** flag, it is necessary in order to align the iVerilog tool with the Verilog standard that Xilinx uses.

3 Part 2:Vending Machines (50 pts)

As we might have short breaks between two activities, vending machines can become a life savior. We have two different machines in our department, one includes food such as sandwiches, while the other includes hot beverages.

Figure 3: Vending Machine

You will implement a vending machine system that can be summarized as follows:
There are 2 machines; VM0 and VM1. One can purchase sandwiches, chocolate, and water from VM0. On the other hand, VM1 only offers beverages, namely coffee, tea, and water. The system can be used with cash money represented by 6 bits in the system. After depositing the money, and selecting the machine (VM0 or VM1), one can select the product which is represented with 3 bits. Your aim is to design the system summarized above having the following specifications:

Table 2: Product names, ID's, and their prices

productID	**Product Name**	**Available machine**	**Price**
000	Sandwiches	VM=0	20
001	Chocolates	VM=0	10
010	Water	VM=0 and VM=1	5
011	Coffee	VM=1	12
100	Tea	VM=1	8

1. Initially machines include the following products
   - VM0 includes 10 sandwiches, 10 chocolates, and 5 bottles of water.
   - VM1 includes 10 cups of coffee, 10 cups o tea, and 10 bottles of water.

   You should count these numbers internally, and each time a user makes a purchase, you should update the number of products.
2. Each product is represented by 3 bit-code (*productID*) and has a price, which is shown in table 2
3. If the selected product is not available in the selected machine, the system will give a warning (*invalidProduct*). In other words, the system will raise this warning if

- the user tries to purchase a *productID* with ids 3'b011, 3'b100 from VM0
- the user tries to purchase a *productID* with ids 3'b000, 3'b001 from VM1
- 3'b101, 3'b110 and 3'b111 are invalid for both machines.

If the product is invalid for the given VM, the system gives a warning and stays idle until the next positive edge of the clock.

4. You should internally count the number of available products, if the product is depleted, the system should give a warning (*productUnavailable*). If the product is not available, the system gives this warning and stays idle until the next positive edge of the clock.
5. With VM1, one can add sugar (by setting *sugar* input) to coffee and tea. If the user tries to add sugar to water, the system will give a warning (*sugarUnsuitable*). If the selected machine is VM0, and the user tries to add sugar to the water, system gives this warning as well and stays idle until the next positive edge of the clock.
6. VM0 works with exact money. If the provided money is not an exact match with the selected product's price, the system will give a warning (*notExactFund*).
7. With VM1, if the provided money is less than the price of the selected product, the system will give a warning (*insufficientFund*).
8. If the purchase is successful, the system will;
   - gives success signal (*productReady*)
   - provides the number of items left from the selected product (*itemLeft*)
   - provides the remaining cash left from the purchase (*moneyLeft*). Please note that, since VM0 works with exact money, *moneyLeft* will always be 0 in this machine.
9. System outputs warning in an ordered fashion. This order is defined as follows:
   - *invalidProduct*
   - *productUnavailable*
   - *sugarUnsuitable* (Only valid for VM1)
   - *notExactFund* (Only valid for VM0)
   - *insufficientFund* (Only valid for VM1)

   For example, if an invalid product id is given to a VM, the *invalidProduct* signal will be raised, rest of the warnings will be "don't care". As another example, if *notExactFund* should be raised for VM0 (due to the provided money not exactly matching the item's price); *invalidProduct*, and *productUnavailable*, **must be** zero. Finally, *notExactFund* should be set to one. *sugarUnsuitable* and *insufficientFund* is unrelated to the VM0 thus these signals are still "don't care".

   When a warning occurs, success output *productReady* should be zero. *itemLeft* should be "don't care" and *moneyLeft* should return the exact value of the input *money* (as if the machine returns the provided money to the user).
10. System is triggered by **positive edge** of the clock.

3.1 Sample Input/Output

Sample sequence of inputs and outputs can be seen on Table 3. Light gray variables indicate the internal state of the module.

Table 3: Sample State Flow of the Vending Machines. In **clk** column of table above, "↑" represents the rising edge of the clock.

	Inputs				**CurrentState**															**NextState**														
Line No	**money**	**VM**	**productID**	**sugar**	**moneyLeft**	**itemLeft**	**productUnavailable**	**insufficientFund**	**notExactFund**	**invalidProduct**	**sugarUnsuitable**	**productReady**	**numOfSandviches**	**numofChocolates**	**numOfWatersinVM0**	**numOfCoffee**	**numOfTea**	**numOfWaterinVM1**	**clk**	**moneyLeft**	**itemLeft**	**productUnavailable**	**insufficientFund**	**notExactFund**	**invalidProduct**	**sugarUnsuitable**	**productReady**	**numOfSandviches**	**numofChocolates**	**numOfWatersinVM0**	**numOfCoffee**	**numOfTea**	**numOfWaterinVM1**	**Explanation**
1																				0	0	0	0	0	0	0	0	10	10	5	10	10	10	Initial State
2	111111	1	011	0	0	0	0	0	0	0	0	0	10	10	5	10	10	10	↑	110011	9	0	0	X	0	0	1	10	10	5	9	10	10	User wants 'coffee' (011) from VM1 without sugar, everything is correct, warnings are zero except notExactFund since it is related to VM0.
3	000011	1	010	1	110011	9	0	0	X	0	0	1	10	10	5	9	10	10	↑	000011	X	0	X	X	0	1	0	10	10	5	9	10	10	User now wants water with sugar (from VM0), which is not a correct setting, sugar unsuitable is set and, money is returned.
4	111101	0	100	X	000011	X	0	X	X	0	1	0	10	10	5	9	10	10	↑	111101	X	X	X	X	1	X	0	10	10	5	9	10	10	User wants tea from VM0, product is invalid for that machine, so invalid product is set. Rest of the warnings are not valid.
5	010100	0	000	X	111101	X	X	X	X	1	X	0	10	10	5	9	10	10	↑	000000	9	0	X	0	0	X	1	9	10	5	9	10	10	Proper usage of VM0 with input of sandwich and exact amount of money
6	000001	1	100	0	000000	9	0	0	0	0	X	1	9	10	5	9	10	10	↑	000001	X	0	1	X	0	0	0	9	10	5	9	10	10	User wants tea from VM1, but does not have enough money.
7	000101	0	010	X	000001	X	0	1	X	0	0	0	9	10	5	9	10	10	↑	000000	4	0	0	0	0	X	1	9	10	4	9	10	10	User makes a successful purchase. In order to show the "product unavailable" signal in the following lines as an example, lets assume that the user keeps getting water from VM0.
8	…																																	
9	000101	0	010	X	000000	1	0	0	0	0	X	1	9	10	1	9	10	10	↑	000000	0	0	0	0	0	X	1	9	10	0	9	10	10	One last water is available, user gets it.
10	000101	0	010	X	000000	0	0	0	0	0	X	1	9	10	0	9	10	10	↑	000101	X	1	X	X	0	X	0	9	10	0	9	10	10	Again user tries to get water from VM0. User gets a warning since water on VM0 is depleted. Money is returned.
11	111111	1	111	1	000101	X	1	X	X	0	X	0	9	10	0	9	10	10	↑	111111	X	X	X	X	1	X	0	9	10	0	9	10	10	User tries to set an unknown product id (111). This should set invalidProduct warning.

3.2 Input/Output Specifications

Name	**Type**	**Size**
money	Input	6 bits
Clock (CLK)	Input	1 bit
sugar	Input	1 bit
VM (Machine ID)	Input	1 bit
productID	Input	3 bits
moneyLeft	Output	6 bits
itemLeft	Output	5 bits
productUnavailable	Output	1 bit
insufficientFund	Output	1 bit
notExactFund	Output	1 bit
invalidProduct	Output	1 bit
sugarUnsuitable	Output	1 bit
productReady	Output	1 bit

- **money** is the 6 bits representing the amount of money user uploads.
- **CLK** is the clock input for the module.
- **sugar** is used to add sugar to coffee and tea.
- **VM** is used for the selection of the vending machine.
  - $VM = 0 \Rightarrow$ Machine 0 including sandwiches, chocolates and water.
  - $VM = 1 \Rightarrow$ Machine 1 including coffee, tea and water.
- **productID** is used to indicate the ID of the product. Each product has a price.
  - $productID = 000 \Rightarrow$ sandwiches available in $VM = 0$
  - $productID = 001 \Rightarrow$ chocolates available in $VM = 0$
  - $productID = 010 \Rightarrow$ water available in both machines
  - $productID = 011 \Rightarrow$ coffee available in $VM = 1$
  - $productID = 100 \Rightarrow$ tea available in $VM = 1$
  - $productID =$ otherwise $\Rightarrow$ don't care (*invalidProduct*).
- **invalidProduct**: This warning shows whether the selected machine does not contain the selected product
  - $invalidProduct = 1 \Rightarrow$ Show the warning, the selected machine does not contain the selected product according to Table 2.
  - $invalidProduct = 0 \Rightarrow$ Do not show the warning, the selected machine contains the selected product.
- **productUnavailable**: This warning shows whether the selected machine originally contains the selected product, however, the product is not left on the machine.
  - $productUnavailable = 1 \Rightarrow$ Show the warning, the selected machine did contain the selected product but it is depleted.
  - $productUnavailable = 0 \Rightarrow$ Do not show the warning, the selected machine contains the selected product.
- **sugarUnsuitable**: This warning is specific for VM1. It shows whether the selected product is suitable to add sugar.
  - $sugarUnsuitable = 1 \Rightarrow$ Show the warning if user selected machine 1 and water as product while setting the *sugar* input to 1.

  - *sugarUnsuitable* = 0 ⇒ Do not show the warning if the user selected machine 1 and coffee or tea as the product.

- **notExactFund**: This warning is specific for VM0. This machine does not give the remainder of the money after the purchase, therefore it accepts only the exact amount of money. This warning shows whether the provided money is exact.
  - *notExactFund* = 1 ⇒ Show the warning if the user selected machine 0 and money is not exact based on the selected product. Note that money can be higher or lower than the product prize. If the money is not exact, this warning is shown.
  - *notExactFund* = 0 ⇒ Do not show the warning if user selected machine 0 and money is exact based on the selected product.
- **insufficientFund**: This warning is specific for VM1. This machine can give remainder of the money after the purchase. However if the amount of money is not enough for the purchase this warning is shown.
  - $insufficientFund = 1$ ⇒ Show the warning if user selected machine 1 and money is lower than the prize of the selected product.
  - $insufficientFund = 0$ ⇒ Do not show the warning if user selected machine 1 and money is lower than the prize of the selected product.
- **productReady**: If the selected product can be given to the user, enable this signal.
  - *productReady* = 1 ⇒ Show if the selected product is given to the user.
  - *productReady* = 0 ⇒ Show if the selected product can not be given to the user.
- **itemLeft**: If the users purchase is successful, show the number of items left in the machine.
- **moneyLeft**: If the users purchase is successful, show the amount of money returned to the user. Note that if the user selects VM0, $moneyLeft$ will always be 0, since VM0 requires exact money. If a warning occurs this should be the same value as the *money*.

3.3 Deliverables

- Implement your module in a single Verilog file. Upload only **lab3_2.v** file to ODTUClass system.
  Do **NOT** submit your testbenches, bit files or other project files. You can share your testbenches on ODTU-Class discussion page.
- Submit the file through the ODTUClass system before the deadline given at the top.
- Use the ODTUClass discussion for any questions regarding the homework.
- This is an individual work, any kind of cheating is not allowed.
- Implementations will be tested using custom testbenches in a blockbox fashion. We will use **Xilinx** and **iverilog** to conduct these tests. Please make sure that your code compiles over these tools.
- for iVerilog compilation please use **-g2005** flag, it is necessary in order to align the iVerilog tool with the Verilog standard that Xilinx uses.