Middle East Technical University - Department of Computer Engineering

CENG 232

Logic Design

Spring '2022-2023

Lab 4

Due Date: 01 June 2023, Thursday, 23:55
No late submissions

1 Part 1: Encoded Memory (30 pts)

In this part, you are expected to implement basic memories as 2 Verilog modules. The modules together take a binary number and an operation type (read/write) and then according to the operation type, either calculates and records the absoulte value of the difference between the given number to the given memory index (write mode), or returns the previously computed data from the given index of the memory (read mode).

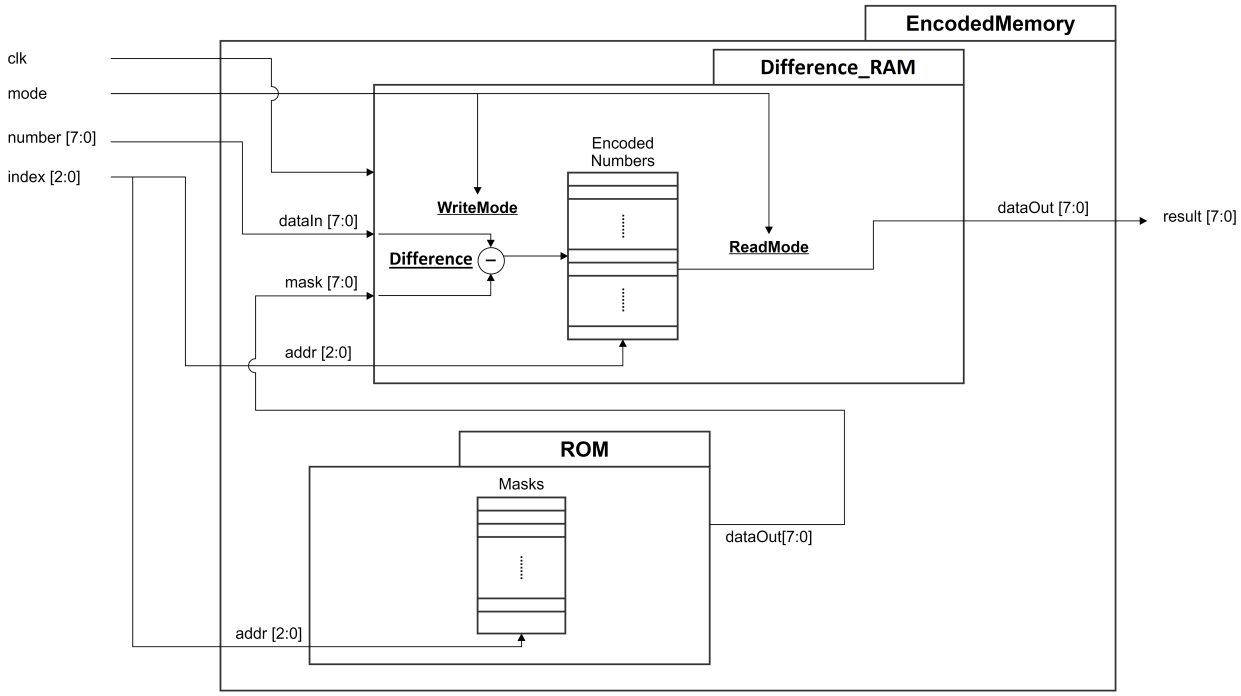

Figure 1: Illustration of the modules.

1.1 ROM Module

The ROM uses 3-bit addresses (**addr**). Each memory location contains 8 bits of data. You should take into consideration the specifications below:

1. This ROM works asynchronously, it is not triggered by a clock pulse.
2. It returns the cell value with the given index (**addr**) as output (**dataOut**) immediately.
3. The values of ROM should be as provided in the following table:

addr	Data Value
0	00000000
1	01010101
2	10101010
3	00110011
4	11001100
5	00001111
6	11110000
7	11111111

Use the following lines as the port definition of the ROM Module:

```
module ROM (
input [2:0] addr,
output reg [7:0] dataOut);
```

1.2 Difference_RAM Module

The RAM uses 3-bit addresses (**addr**). Each memory location contains 8 bits of data. You should take into consideration the specifications below:

1. Initially, the values of all RAM registers will be 0.
2. There are 2 modes in this module:
   - 0 - write mode
   - 1 - read mode
3. Write mode is synchronous. It is triggered by the positive edge of the clock (CLK).
4. Read mode is **asynchronous**. That means it is not triggered by a clock pulse.
5. In write mode, an *absolute value* of the difference between **dataIn** and the **mask** will be computed. The result (abs($dataIn \ominus mask$)) will be written in **addr** location of the RAM.
6. In read mode, no write operation to the memory is conducted. The value stored in the **addr** location of the RAM will be returned in dataOut.
7. The initial value of **dataOut** should be 0. It can only be changed in read mode. It should retain the last read value when the module is in write mode.

Use the following lines as the port definition of the Difference_RAM Module:

```
module Difference_RAM (
input mode,
input [2:0] addr,
input [7:0] dataIn,
input [7:0] mask,
input CLK,
output reg [7:0] dataOut);
```

1.3 EncodedMemory Module

This is the upper module, in which inputs and outputs of other modules are defined. The inputs of this module; mode, CLK, index, number and result are distributed to ROM and Difference_RAM modules. **You should not edit this module.**
Illustration of the overall Encoded Memory system is given in Figure 1.

```
module EncodedMemory (
input mode,
input [2:0] index,
input [7:0] number,
input CLK,
output [7:0] result);
```

1.4 Deliverables

- Implement all modules in a single Verilog file (provided to you as **lab4_1.v**).
  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.

2 Part 2: Election of the Avatar(70 pts)

The Reincarnation cycle is broken more than 200 years ago since the last Avatar has departed. Once, the four nations lived in perfect harmony, but now war is the only constant. After countless negotiations, an agreement has been reached with the spirit world. People will now have the responsibility to choose the next avatar, and the lion turtles will bestow the selected candidate with the power of the Avatar. With this momentous agreement, a new era of peace and prosperity dawns upon us.

Four candidates have emerged for the daunting role of the Avatar, with one representative from each of the four nations: Air, Water, Fire, and Earth. To ensure a fair and transparent selection process, each nation has designated 16 deputies who will cast their votes in the upcoming election, set to take place on Kyoshi Island. Your mission is to assist the local community in ensuring a seamless and efficient electoral process.

In this part, you are expected to implement an election system where there are 4 candidates and 64 voters. The details of the system is described below:

1. Selection process is carried in 3 periods. In the first period (which lasts for 100 clock cycles) voters will enroll to the system and get their ballot box ids. In the second period (which also lasts for 100 clock cycle) the voting will be carried. After 200 clock cycles the winner will be announced.
2. Each candidate is represented by 2 bit *candidate* number as follows;

Ballot Box Id: 00		Ballot Box Id: 01		Ballot Box Id: 10		Ballot Box Id: 11	
Index	Registered?	Index	Registered?	Index	Registered?	Index	Enrolled?
0000	1	0000	0	0000	1	0000	1
0001	0	0001	0	0001	0	0001	0
1000	0	1000	1	1000	1	1000	1
1111	0	1111	1	1111	0	1111	1

Figure 2: Ballot boxes enrollment list

- 00: Candidate for the air nation
- 01: Candidate for the fire nation
- 10: Candidate for the water nation
- 11: Candidate for the earth nation

3. There are 4 ballot boxes. In each box 16 person can vote.
4. Each voter can register and vote for ony **ONCE**. You should internally keep track the registered voters, and whether they have voted or not.
5. Each voter has 6 bit user id that will be used to determine their ballot box ids. In order to do so you should slice the userIds.
   - Most significant 2 bits of the user id represents the users ballot box.

   Hint: You should keep track of the registered and voted users. For that purpose you can use register arrays (memory) for ballot boxes. In order to simplify userId searches in this registers you can use the least significant 4 bits of userId as the index. You can use a structure similar to the one give in Figure 2. In this example we can see users with IDs 00000, 011000, and 110000 are registered to the system, however, 000001, 100001 have not registered yet. You can use a similar structure to keep track of voting information.
6. Mode represents registration (mode=0) vs voting (mode =1) phases of the election process.
7. In the first 100 cycle;
   - Users can only register to the system. (mode = 0)
   - If the user is registering for the first time you should add them to your internal lists, and show the user their ballot box id (output reg ballotBoxId)
   - If the user has registered before, system gives *AlreadyRegistered* warning. You should again show the user ballot box id.
   - If the user selected mode=1 (voting mode) system gives *VotingHasNotStarted* warning.
8. In the second 100 cycle (101-200) users can vote (mode =1) with the following protocol;
   - Only registered users can vote.
   - If the user is registered and voting for the first time, user can vote with candidate ids. (See Item 2 for candidate ids). You should internally count the number of votes per candidate.
     - If the user is not registered, system gives *NotRegistered* warning.

- You should internally keep track of the voted users. If a user tries to vote more than once system gives *AlreadyVoted* warning.

- If the user tries to register during this time period (mode= 0) system gives *RegistrationHasEnded* warning.

9. After 200 cycles, regardless of the inputs, system only shows the winnerID and how many votes the winner had (*numberOfVotesWinner*) as well as *totalNumberOfRegisteredVoters*.

10. System is triggered by **positive edge** of the clock. You can assume there will be an input on every positive edge of the clock.

Table 1: Sample State Flow of the Ballot. In **clk** column of table above, "↑" represents the rising edge of the clock.

		Inputs				**CurrentState**									
Line No	**Clock Cycle**	**mode**	**userID**	**candidate**	**clk**	**ballotBoxId**	**numberOfRegisteredVoters**	**numberOfVotesWinner**	**WinnerId**	**AlreadyRegistered**	**AlreadyVoted**	**NotRegistered**	**VotingHasNotStarted**	**RegistrationHasEnded**	**Explanation**
1	1	0	111111	x	↑	11	1	x	x	0	0	0	0	0	User with ID = 111111 registers to the system
2	2	0	111111	x	↑	11	1	x	x	1	0	0	0	0	User with ID = 111111 tries to registers to the system. Since system gives AlreadyRegistered warning and shows the ballotBoxId
3	3	1	111111	01	↑	11	1	x	x	0	0	0	1	0	User with ID = 111111 tries to vote. Since we are still in the first 100 clock cycles system gives VotingHasNotStarted warning and shows the ballotBoxId
4	4	0	000001	x	↑	00	2	x	x	0	0	0	0	0	User with ID = 000001 registers to the system
5	5	0	101101	x	↑	10	3	x	x	0	0	0	0	0	User with ID = 101101 registers to the system
6	6	0	011100	x	↑	01	4	x	x	0	0	0	0	0	User with ID = 011100 registers to the system
7	...														Assume 100 cycles has passed and 63 users has registered to the system. Assume only the user with id 010101 has not registered to the system.
8	101	1	111101	01	↑	11	63	x	x	0	0	0	0	0	User Id = 111101 votes to Candidate Id = 01
9	102	1	111101	11	↑	11	63	x	x	0	1	0	0	0	User Id = 111101 tries votes to Candidate Id = 01 second time. System gives AlreadyVoted warning.
10	103	0	111101	x	↑	11	63	x	x	0	0	0	0	1	An already registered user tries to register to the system. Since we are in the election phase (clock cycle=103) system gives RegistrationHasEnded warning. System does **NOT** give AlreadyRegistered warning. See Section 2.2 in the main text)
11	104	0	010101	x	↑	01	63	x	x	0	0	0	0	1	Non-registered user 010101 tries to register to the system. System gives RegistrationHasEnded warning.
12	105	0	010101	10	↑	01	63	x	x	0	0	1	0	0	Non-registered user 010101 tries to vote. System gives NotRegistered warning.
13	...														Assume another 100 clock cycle has passed. Air nations candidate (with id 00) get 42 votes and won the election.
14	201	1	010101	10	↑	XX	63	42	00	X	X	X	X	X	Regardless on the input system shows only ballotBoxId, numberOfRegisteredVoters, numberOfVotesWinner and WinnerId after 200 cycles.
15	202	0	001010	001	↑	XX	63	42	00	X	X	X	X	X	
16	203	0	111111	001	↑	XX	63	42	00	X	X	X	X	X	

2.1 Sample State Table

Sample sequence of inputs and outputs can be seen on Table 1. Red column indicates the total clock cycles that passed after the system start-up and, does not relate to any input output.

2.2 Input/Output Specifications

Name	Type	Size
mode	Input	1 bits
Clock (CLK)	Input	1 bit
userID	Input	6 bits
candidate	Input	2 bit
ballotBoxId	Output	2 bits
numberOfRegisteredVoters	Output	6 bits
numberOfVotesWinner	Output	6 bits
WinnerId	Output	2 bit
AlreadyRegistered	Output	1 bit
AlreadyVoted	Output	1 bit
NotRegistered	Output	1 bit
VotingHasNotStarted	Output	1 bit
RegistrationHasEnded	Output	1 bit

- **mode** is used for determining whether user tries to register or vote.
  - mode = 0; User is registering to the system. Users can use this option only in the first 100 clock cycles.
  - mode = 1; User is voting. Users can use this option only in the second 100 clock cycles (101-200).
- **mode** is *don't care* after 200 clock cycles
- **userID** Unique user IDs for 64 that will vote in the election. User id is *don't care* if 200 clock cycles has passed and the election is over.
- **candidate** Unique 2 bit ID of the candidates. User provides this id when they are voting. (00: Candidate of the Air nation. 01: Fire nation, 10:Water Nation, 11:Earth Nation)

  candidate is *dont't care* if mode is 0 (user is registering), or 200 clock cycles has passed and the election is over.
- **ballotBoxId** Ballot box ids of the voters. There are 4 ballot boxes. Most significant 2 bits of userID represents the ballot ids. Its value is shown to the user in the first 200 cycles, based on the id, regardless of other inputs and warnings. It becomes *don't care* after 200 cycles
- **numberOfRegisteredVoters**You should count the number of registered voters as they register to the system. The value is shown after the 100th clock cycle regardless of the inputs.
- **numberOfVotesWinner** shows the number of votes the winner candidate get during the election. *numberOfVotesWinner* should be 0 in the election process (first 200 cycles). System only shows this information after 200 cycles.
- **WinnerId** is the 2 bit ID of the winner candidate. During the election process *WinnerID* will be *don't care*. System only shows this information after 200 cycles.
- **AlreadyRegistered** show this warning if an already registered user tries to register to the system again in the first 100 cycles. Otherwise 0.
- **AlreadyVoted** show this warning if an already voted user tries to vote again in the second 100 cycles. Otherwise 0.
- **NotRegistered** show this warning if a non-registered user tries to vote in the second 100 cycles. Otherwise 0.
- **VotingHasNotStarted** show this warning if an already registered user tries to vote in the first 100 cycles. Otherwise 0.
- **RegistrationHasEnded** show this warning if a non-registered user tries to register in the second 100 cycles. Otherwise 0.

2.3 Deliverables

- Implement the module in a single Verilog file (provided to you as **lab4_2.v**).
  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.
- 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.