Middle East Technical University | Department of Computer Engineering

CENG 232

Logic Design

Spring '2017-2018

Lab 5

Part 1 Due Date: 9 May 2018, Wednesday, 23:59
Part 2 Due Date: 16 May 2018, Wednesday, 23:59
No late submissions

1 Introduction

MM needs you!

Elevators in the MM building are broken down! Rektörlük has hired you to program the elevators. Program the elevators as a computer engineer and they won't ever break down (show our department's class :)). People has to climb for 15 floors every day. This makes them lose weight every single day and they will melt away in approximately three weeks. So, you have to be quick...

2 Part 1: Naive Elevator (Individual Work)

This part of the lab will be performed and submitted individually (NOT WITH YOUR GROUP PARTNER!). In this part of the assignment, you are expected to draw an ASM chart showing the behavior of **naive elevator**.

In the naive elevator problem, you will design a system that consists of **only one elevator**. The system has a **task pool**, in which the incoming call requests are stored. Each task shows the floor number of the request. Each task is 4 bits long. The available floors in the building are from 1 to 15 (The building does not have a 0th floor). The **task pool** works in FIFO (First In First Out) fashion and is capable of holding at most 8 requests at a time. So, when 8 requests are already stored in the task pool, a new incoming request is rejected. The elevator has two states. IDLE state and WORK state. In the IDLE state requests can be *added/deleted* to/from the **task pool** or added requests can be *listed*. The WORK state is the place where the elevator takes the next request from the pool and starts to move to the floor of that request. The detailed explanation of the states are provided below.

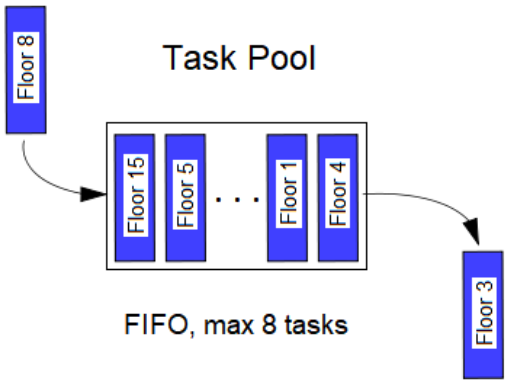

Figure 1: Task Pool

2.1 IDLE State

This is the state the elevator is initially at. In this state: it can *read(add)* new tasks, it can *show* the tasks that are currently in the task pool or it can *cancel(delete)* some of the tasks in the task pool.

The system takes a two bit input named **mode** in the IDLE state. These two bits are defined as follows:

- **mode = 00:** This mode is used to *add* tasks to the task pool. The task pool can take 8 tasks at most. Initially all these places are filled with zeros (ex. each element is 4'b0000). As there is no 0th floor, no tasks will have 4'b0000 as its value.
- **mode = 01:** This mode is used to *list* the tasks in the task pool in order of their addition order. At each rising edge of the clock, the next task is printed to the specified leds. Until the listing is done, the mode variable will not be changed (by the user). While the listing continues **listBusy** led has to be on. After the listing finished, **listBusy** led is turned off and a new mode can be chosen.
- **mode = 10:** This mode is used to *delete* tasks from the task pool. If the provided task exists in the task pool, that task is deleted from the pool and the task order must be preserved. If the given task does not exist in the task pool, no actions will be taken.
- **mode = 11:** This mode is used to trigger the elevator to go to WORK state. Before changing the state, the elevator must read the next task from the **task pool** as its current task and *delete* that task from the pool.

2.2 WORK State

This is the state where elevator moves from one floor to another. In this state elevator moves towards to requested floor at the each rising clock edge. For example, if the elevator is at 3rd floor and the requested floor is 9. With the first rising clock edge elevator closes its door and moves to 4, with the next rising clock edge it moves to 5 etc. When the elevator reaches the requested floor, it opens its door and changes its state to IDLE without changing its floor. Note that, elevator only performed one request and returned back to IDLE state.

2.3 Clarifications

1. Each **task** represents a floor and a task consists of a 4-bit number.
2. No **task** can have a value 0. Task values changes between 1 and 15.
3. Initially elevator is in IDLE state.
4. Initially elevator is in 1st floor and its door is open.
5. Initially **task pool** is filled with 8 zero filled tasks (which means the task pool is empty).
6. To move from IDLE state to WORK state, **mode** must be changed to 11. After **mode** is changed, with the rising clock edge elevator goes to WORK state.

7. To move from WORK state to IDLE state, current job must be finished (Changing modes makes no effect in the WORK state. When elevator finishes its current job it must automatically go to IDLE state with the rising clock edge).
8. When elevator is in IDLE state, changing between modes 00, 01 and 10 does not change the state of the elevator (still IDLE).
9. In IDLE state, **state** bit is 0.
10. In WORK state, **state** bit is 1.
11. In add mode (*mode = 00*) (in IDLE state), requests must be added in the order they made. If a request that already exist in task pool arrives, it must be ignored. If the task pool is full (8 tasks in it), then the incoming requests must be ignored. No task with the value 0 will be added (*the user won't enter a task with value 0*).
12. When in show mode (*mode = 01*) (in IDLE state), listing starts from first added request and continues until the last added element. Empty places (places with 0000) won't be listed. Element to be listed should be written into the variable **listingLeds**. While the listing continues, **listBusy** bit must be set to 1. When the listing finishes, with the rising edge of the clock after the last element is shown **listBusy** must be set to 0. User won't change the mode, until listing finishes.
13. In delete mode (*mode = 10*) (in IDLE state), if the given element is in the list it must be deleted and the order of the tasks must be preserved. If the given element does not exist in the list, no actions will be done. If the task pool is empty and a deletion request arrived, no actions will be done.
14. The building has 15 floors, starting from 1. Floors larger than 9 will be represented as A,B,C,D,E and F.
15. Door is open when the **doorOpen** variable is 1 and door is closed when the **doorOpen** variable is 0.
16. If the mode is changed to 11 and the **task pool** is empty elevator normally should go to WORK state with the clock cycle, then this situation must be checked in WORK state and elevator must return to IDLE state (in the same clock cycle).

2.4 Sample Input/Output

In the tables 1, 2, 3 and 4, the format of current and next states is as follows:
mode, **request**, **state**, **listBusy**, **listingLeds**, **doorOpen**, **currentFloor**

In table-1, add mode is demonstrated with adding 4, 1, 15 and 9.

current state	CLK	next state	description
00, 0100, 0, 0, 0000, x, x	↑	00, 0100, 0, 0, 0000, x, x	added decimal 4 (0100)
00, 0001, 0, 0, 0000, x, x	↑	00, 0001, 0, 0, 0000, x, x	added decimal 1 (0001)
00, 1111, 0, 0, 0000, x, x	↑	00, 1111, 0, 0, 0000, x, x	added decimal 15 (1111)
00, 1001, 0, 0, 0000, x, x	↑	00, 1001, 0, 0, 0000, x, x	suppose pool is full, reject decimal 9

Table 1: Add mode demonstration

In table-2, suppose we have added **4, 1, 15** to the task pool, respectively. Now task pool has only 3 tasks in it. Then in **list** mode:

current state	CLK	next state	description
01, xxxx, 0, 0, 0000, x, x	↑	01, xxxx, 0, 1, 0100, x, x	lists 4
01, xxxx, 0, 1, 0100, x, x	↑	01, xxxx, 0, 1, 0001, x, x	lists 1
01, xxxx, 0, 1, 0001, x, x	↑	01, xxxx, 0, 1, 1111, x, x	lists 15
01, xxxx, 0, 1, 1111, x, x	↑	01, xxxx, 0, 0, 0000, x, x	listing ends

Table 2: List mode demonstration

Suppose we again have **4, 1, 15** in the task pool.
In the table-3, we first delete **1** from the task pool, then **list** the tasks in the current task pool:

current state	CLK	next state	description
10, 0001, 0, 0, 0000, x, x	↑	10, 0001, 0, 0, 0000, x, x	deleted 1
01, xxxx, 0, 0, 0000, x, x	↑	01, xxxx, 0, 1, 0100, x, x	lists 4
01, xxxx, 0, 1, 0100, x, x	↑	01, xxxx, 0, 1, 1111, x, x	lists 15
01, xxxx, 0, 1, 1111, x, x	↑	01, xxxx, 0, 0, 0000, x, x	listing ends

Table 3: Delete mode demonstration

In the table-4, suppose we have **4, 1, 15, 14** in the task pool. The elevator reads the next task, which is 4, removes it from the task pool, then changes to WORK state. It then goes to 4th floor by climbing 1 floor at each rising edge. When it arrives to 4th floor, the door is opened and the state is changed to IDLE. When the user continues in **mode 11**, same procedure is applied. In the $\cdots$ row, 11 times **CLK** rising edge is applied.

current state	CLK	next state	description
11, xxxx, 0, 0, 0000, 1, 1	↑	11, xxxx, 1, 0, 0000, 0, 2	
11, xxxx, 1, 0, 0000, 0, 2	↑	11, xxxx, 1, 0, 0000, 0, 3	
11, xxxx, 1, 0, 0000, 0, 3	↑	11, xxxx, 0, 0, 0000, 1, 4	reached 4th floor
11, xxxx, 0, 0, 0000, 1, 4	↑	11, xxxx, 1, 0, 0000, 0, 3	
11, xxxx, 1, 0, 0000, 0, 3	↑	11, xxxx, 1, 0, 0000, 0, 2	
11, xxxx, 1, 0, 0000, 0, 2	↑	11, xxxx, 0, 0, 0000, 1, 1	reached 1st floor
11, xxxx, 0, 0, 0000, 1, 1	↑	11, xxxx, 1, 0, 0000, 0, 2	
11, xxxx, 1, 0, 0000, 0, 2	↑	11, xxxx, 1, 0, 0000, 0, 3	
$\cdots$	↑	$\cdots$	
11, xxxx, 1, 0, 0000, 0, E	↑	11, xxxx, 0, 0, 0000, 1, F	reached 15th floor
11, xxxx, 0, 0, 0000, 1, F	↑	11, xxxx, 0, 0, 0000, 1, E	reached 14th floor

Table 4: WORK state demonstration

2.5 Input/Output Specifications

- **CLK** is the clock input for the module.
- **mode** is a 2-bit variable:
  mode = 00 ⇒ Add mode
  mode = 01 ⇒ List mode
  mode = 10 ⇒ Delete mode
  mode = 11 ⇒ Work mode
- **request** is a 4-bit variable.
- **state** is a 1-bit variable that defines the state:
  state=0 ⇒ IDLE state
  state=1 ⇒ WORK state
- **listBusy** is a 1-bit variable:
  listBusy=0 ⇒ Currently not listing the tasks.
  listBusy=1 ⇒ Currently listing the tasks.
- **listingLeds** is a 4-bit variable which is used to list the tasks.
- **doorOpen** is a 1-bit variable:
  doorOpen=0 ⇒ Door is closed.
  doorOpen=1 ⇒ Door is opened.
- **currentFloor** is a 4-bit variable which shows the current floor of the elevator.

Name	Type	Size
Clock (CLK)	input	1-bit
mode	input	2-bit
request	input	4-bit
state	output	1-bit
listBusy	output	1-bit
listingLeds	output	4-bit
doorOpen	output	1-bit
currentFloor	output	4-bit

Table 5: Input and output variables for part 1

2.6 Deliverables

- In this part of the assignment, you are expected to draw an ASM chart showing the behavior of the Naive Elevator problem defined in Part 1.
- Don't forget that this is an individual assignment, so any kind of collaboration will be counted as cheating.
- You can use any input-output definitions or abbreviations of them (by stating the exact names of it in your assignment) defined in input/output specifications. Also you can use VARIABLE[a:b] in order to refer a variable while implementing Naive Elevator.
- Please make sure that you cover all the details of Naive Elevator problem and conform to the ASM notation.
- You can draw ASM chart with the any drawing tool you want (we don't want any hand writing). Any files that have hand writing won't be accepted.
- You are supposed to send a file named NaiveElevator_e1234567.pdf/jpg/png any of these file extensions will be accepted.
- Submit the file through the COW system before the given deadline. **May 9, 2018, 23:59**
- Use the newsgroup metu.ceng.course.232 for any questions regarding the homework.

3 Part 2: 2-Elevators (Teamwork)

This part of the lab will be performed and submitted with your group partner. Only submit a single copy per group.

3.1 Problem Definition

You will implement the provided 2-Elevators problem that is defined below in FPGA boards.

2-Elevators problem is an improved version of the Naive Elevator problem. In this problem, we have two elevators instead of one elevator. We have again a single task pool. Elevators take tasks from task pool in an ordered

fashion (check the clarifications part). They are always in the same state (IDLE or WORK). Clarifications of this problem are provided below:

1. There are two elevators in the system. Named elevator1 and elevator2.
2. Elevators are always in the same state, they go from IDLE state to WORK state and return from WORK state to IDLE state at the same time.
3. They share the same task pool.
4. Only mode=11 and WORK state changes from Part1, apart from that all of the explanations from Part 1 applies here.
5. In mode = 11: both of the elevators go to WORK state. Before changing the state, the elevator1 must read the next task from the **task pool** as its current task and *delete* that task from the pool. Then elevator2 reads its task from **task pool** and *deletes* that task from the task pool. This means that when in mode=11 in IDLE state, first request in the task pool is given to elevator1 and the second task in the task pool is given to elevator2.
6. In WORK mode, both elevators start with their own tasks(if there are enough available tasks) and they continue their movement simultaneously with the each rising clock edge.
7. If one of the elevators finish its task before the other. It opens its door (as usual). Then with the next rising clock edge, it takes the next task from task pool and continues with its new task.
8. When all of the tasks in the task pool finishes both of the elevators move to IDLE state.
9. If one elevator finishes its task and there are no available tasks left in the task pool then that elevator waits for the other without changing its state (which is WORK state).
10. Door is open when the **doorOpen1** variable is 1 for elevator1 and **doorOpen2** variable is 1 for elevator2. Door is closed when the **doorOpen1** variable is 0 for elevator1 and **doorOpen2** variable is 0 for elevator2.

3.2 Sample Input/Output

For the add, show and delete modes, the same examples can be used from part-1 (you can check, table-1, table-2 and table-3 for examples) as changing the number of elevators does not change anything in these modes. In table-6, a sample WORK state demonstration of 2-elevators problem is given. The format of current and next states is as follows:
mode, **request**, **state**, **listBusy**, **listingLeds**, **doorOpen2**, **currentFloor2**, **doorOpen1**, **currentFloor1**

In table-6, suppose we have **11, 3, 5, 2** in the task pool in this order. Elevator1 reads the next task, which is 11, removes it from the task pool. Elevator2 reads 3 as its task, and removes it from the pool. Then both elevators change to WORK state simultaneously (as always they change state simultaneously). While elevator1 is climbing to 11th floor, elevator2 finishes its task, reads 5 as its new task and removes it from the pool, climbs to 5th floor and finishes the task, then again elevator2 reads 2 as its task and removes it from the pool, goes down to 2nd floor. As there is no tasks left in the pool, elevator2 now waits in WORK state for elevator1 to finish its task. When elevator1 is finished, as no tasks are left, both of the elevators change to IDLE state simultaneously.

current state	CLK	next state	description
11, xxxx, 0, 0, 0000, 1, 1, 1, 1	↑	11, xxxx, 1, 0, 0000, 0, 2, 0, 2	
11, xxxx, 1, 0, 0000, 0, 2, 0, 2	↑	11, xxxx, 1, 0, 0000, 1, 3, 0, 3	elevator2 reached 3rd floor
11, xxxx, 1, 0, 0000, 1, 3, 0, 3	↑	11, xxxx, 1, 0, 0000, 0, 4, 0, 4	
11, xxxx, 1, 0, 0000, 0, 4, 0, 4	↑	11, xxxx, 1, 0, 0000, 1, 5, 0, 5	elevator2 reached 5th floor
11, xxxx, 1, 0, 0000, 1, 5, 0, 5	↑	11, xxxx, 1, 0, 0000, 0, 4, 0, 6	
11, xxxx, 1, 0, 0000, 0, 4, 0, 6	↑	11, xxxx, 1, 0, 0000, 0, 3, 0, 7	
11, xxxx, 1, 0, 0000, 0, 3, 0, 7	↑	11, xxxx, 1, 0, 0000, 1, 2, 0, 8	elevator2 reached 2nd floor
11, xxxx, 1, 0, 0000, 1, 2, 0, 8	↑	11, xxxx, 1, 0, 0000, 1, 2, 0, 9	
11, xxxx, 1, 0, 0000, 1, 2, 0, 9	↑	11, xxxx, 1, 0, 0000, 1, 2, 0, A	
11, xxxx, 1, 0, 0000, 1, 2, 0, A	↑	11, xxxx, 0, 0, 0000, 1, 2, 1, B	elevator1 reached 11th floor

Table 6: WORK state demonstration of 2-elevators

3.3 Input/Output Specifications

- **CLK** is the clock input for the module.
- **mode** is a 2-bit variable:
  mode = 00 ⇒ Add mode
  mode = 01 ⇒ List mode
  mode = 10 ⇒ Delete mode
  mode = 11 ⇒ Work mode
- **request** is a 4-bit variable.
- **state** is a 1-bit variable that defines the state:
  state=0 ⇒ IDLE state
  state=1 ⇒ WORK state
- **listBusy** is a 1-bit variable:
  listBusy=0 ⇒ Currently not listing the tasks.
  listBusy=1 ⇒ Currently listing the tasks.
- **listingLeds** is a 4-bit variable which is used to list the tasks.
- **doorOpen1** is a 1-bit variable:
  doorOpen1=0 ⇒ Door is closed.
  doorOpen1=1 ⇒ Door is opened.
- **doorOpen2** is a 1-bit variable:
  doorOpen2=0 ⇒ Door is closed.
  doorOpen2=1 ⇒ Door is opened.
- **currentFloor1** is a 4-bit variable which shows the current floor of elevator1.
- **currentFloor2** is a 4-bit variable which shows the current floor of elevator2.

Name	Type	Size
Clock (CLK)	input	1-bit
mode	input	2-bit
request	input	4-bit
state	output	1-bit
listBusy	output	1-bit
listingLeds	output	4-bit
doorOpen1	output	1-bit
currentFloor1	output	4-bit
doorOpen2	output	1-bit
currentFloor2	output	4-bit

Table 7: Input and output variables

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.

3.5 Deliverables

- Implement your module in a single Verilog file: lab5_2.v. Do NOT submit your testbenches.You may share your testbenches on the newsgroup.
- Submit the file through the COW system before the given deadline. **May 16, 2018, 23:59**

Name	FPGA Board	Description
Clock (CLK)	BTN0	Right-most button (A)
mode	SW7, SW6	Left-most 2 switches (B)
request	SW3, SW2, SW1, SW0	Right-most 4 switches (C)
state	LD7	Left-most LED (D)
listBusy	LD6	The LED next to LED7 (E)
listingLeds	LD3, LD2, LD1, LD0	Right-most 4 LEDs (F)
currentFloor1	7-segment display	Right-most 7-segment display (G)
doorOpen1	7-segment display	Second right-most 7-segment display (H)
doorOpen2	7-segment display	Left-most 7-segment display (I)
currentFloor2	7-segment display	Second right-most 7-segment display (J)

Table 8: Button descriptions

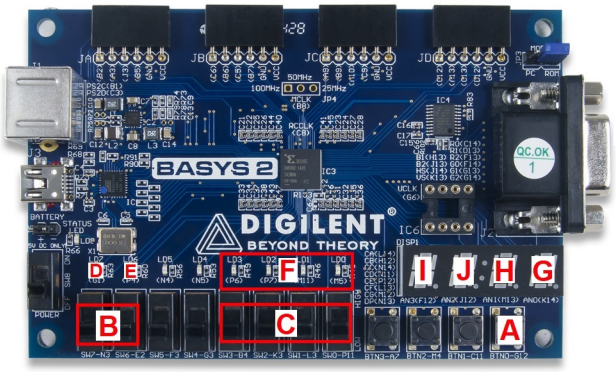

Figure 2: Board with the button informations.

- This part is supposed to be done with your group partner. Make sure both of you take roles in implementation of the project. Any kind of inter-group cheating is not allowed.
- Use the newsgroup metu.ceng.course.232 for any questions regarding the homework.