Middle East Technical University - Department of Computer Engineering

CENG 232

Logic Design

Spring '2020-2021

Lab 3

Part 1 Due Date: 24 May 2021, Monday, 23:55
Part 2 Due Date: 31 May 2021, Monday, 23:55
No late submissions

1 Introduction

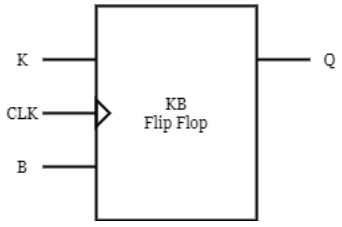
This assignment aims to make you familiar with Verilog language and the related software tools. There are two parts in this assignment. The first part is a Verilog simulation of an imaginary flip-flop design. The second part consists of the implementation of a COVID-19 Vaccination Queue Management System.

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. This is an individual part. Your task is to implement these in Verilog, and prove that they work according to their specifications by using testbenches.

2.1 KB Flip-Flop Module

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



K	B	Q	Qnext
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

2.2 ic232 Module

Implement ic232 chip given in Figure 1 that contains two KB flip-flops and has A0, A1, A2, and clk as inputs; Q0, Q1 and Z as outputs. Please note that A0' and A2' are the complements of A0 and A2, respectively.

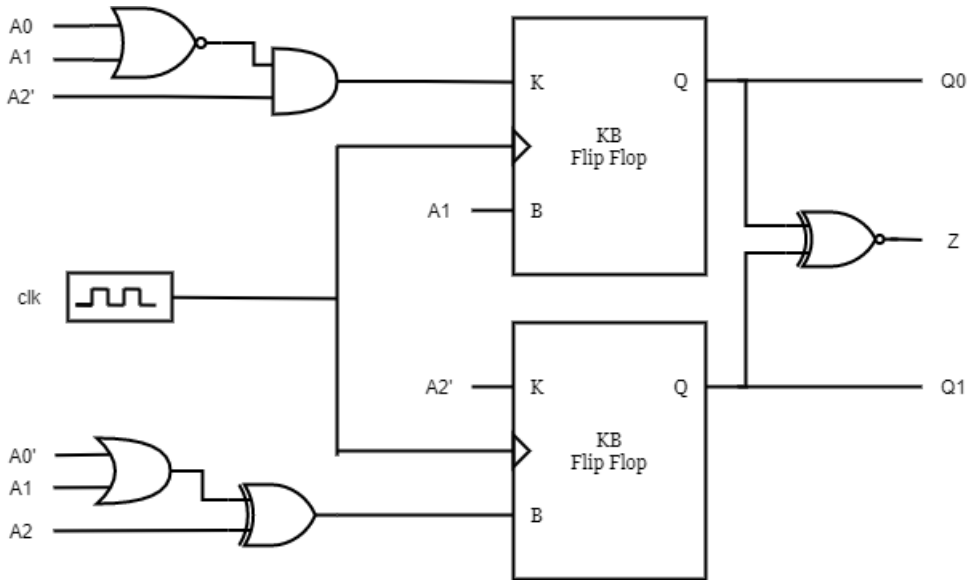


Figure 1: ic232 module, inputs and outputs.

Use the following module definitions for the modules:

```
module kb(input K, input B, input clk, output reg Q)
module ic232(input A0, input A1, input A2, input clk, output Q0, output Q1, output Z)
```

2.3 Simulation

A sample testbench for KB flip-flop module will be provided to you. You may want to extend the testbench, and also write a testbench for ic232 module to test different scenarios.

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 ODTUClass discussion page.
- Submit the file through the ODTUClass system before the deadline given at the top.
- Any kind of cheating is not allowed.

3 Part 2: COVID-19 Vaccination Queue Management System (50 pts)

You and a few friends created a software company called Logictec Inc. An investor comes up to your fresh company with a project. He wants you to create an application for a health clinic that eases queuing residents who come for vaccination. Residents take a number from the numerator to join the queue of a vaccination room; nurses in the rooms serve the next resident waiting in the queue of the room that they are responsible for. As a Verilog expert, your part in the project is to design and implement the system whose definitions are given in the following section.

The project consists of two parts. In the first part, you will design a **queue counter** module, which you will use in the second part to implement the **numerator** of a queuing system of a health clinic.

3.1 Queue Counter Module

The queue counter system that you are going to develop should have the following specifications:

- The queue counter will be synchronous. Increase/decrease operations will be triggered by the **positive edge** of the clock (**clk**).
- The counter performs its operations when enable (**en**) is 1. The counter should not respond to inputs if enable (**en**) is 0.
- Lower limit (**llmt**) and upper limit (**ulmt**) for the counter will be given as input. Their values will stay constant during the lifetime of an instance of the counter. These limits are **inclusive**.
- **tail** represents the number given to the resident who is waiting at the end of the vaccine queue.
- **head** represents the number given to the resident who is being or has been vaccinated by a nurse. ("**head+1**" is the number of the first resident still waiting in the queue).
- The counter will be **cyclic**. There can be states where **tail** is greater than **head** or **head** is greater than **tail**.
- **empty** becomes 1 when there is no resident in the queue. If there are residents in the queue, then **empty** should be 0.
- **full** becomes 1 when the queue is full, otherwise it should be 0.
- Initially, there is no resident in the queue and **empty** is 1. In addition, initially **head** and **tail** must be equal to lower limit (**llmt**). This means that initially the counter starts with a state where the resident with number **llmt** has been served, and now the queue is empty, waiting for new residents to come.
- There are 2 **mode**s in this queue counter:
  - **0**: A new resident wants to take a queue number. Increment **tail**. The incremented tail is the number of the new resident.
  - **1**: A nurse wants to call a waiting resident in a first-come-first-served manner. Increment **head**. The incremented head is the resident who is being called by the nurse.

Table 1: Input/Output Specifications of queue counter module

Name	**Type**	**Size**
llmt	Input	8 bits
ulmt	Input	8 bits
en	Input	1 bit
mode	Input	1 bit
clk	Input	1 bit
head	Output	8 bits
tail	Output	8 bits
empty	Output	1 bit
full	Output	1 bit

Table 2: Sample inputs and outputs for the queue counter module when **llmt=5** and **ulmt=10**.

Current State						clk	Next State				# of residents	Explanation
En	**Mode**	**Head**	**Tail**	**Empty**	**Full**		**Head**	**Tail**	**Empty**	**Full**		
x							5	5	1	0	0	Initial state
1	0	5	5	1	0	↑	5	6	0	0	1	A resident takes a number: 6.
1	0	5	6	0	0	↑	5	7	0	0	2	A resident takes a number: 7.
1	0	5	7	0	0	↑	5	8	0	0	3	A resident takes a number: 8.
1	0	5	8	0	0	↑	5	9	0	0	4	A resident takes a number: 9.
1	0	5	9	0	0	↑	5	10	0	0	5	A resident takes a number: 10.
1	0	5	10	0	0	↑	5	5	0	1	6	A resident takes a number: 5. Queue is full.
1	0	5	5	0	1	↑	5	5	0	1	6	Queue is full. No new number will be given.
1	1	5	5	0	1	↑	6	5	0	0	5	A nurse calls the resident #6
1	1	6	5	0	0	↑	7	5	0	0	4	A nurse calls the resident #7
1	0	7	5	0	0	↑	7	6	0	0	5	A resident takes a number: 6.
1	0	7	6	0	0	↑	7	7	0	1	6	A resident takes a number: 7. Queue is full.
1	0	7	7	0	1	↑	7	7	0	1	6	Queue is full. No new number will be given.
0	x	7	7	0	1	x	7	7	0	1	6	En is 0, do not respond to inputs. Outputs stay unchanged.

3.2 Numerator Module

Using **two** instances of the **queue counter** module that you developed in the previous section, you are required to build a **numerator**. The numerator is designed to be used by both the residents and the nurses of the health clinic. The numerator module that you are going to develop should have the following specifications:

- There are **two** rooms where the range of the queue numbers for each room should be as follows:
  - for the first room ($clinic = 0$), from 5 to 9.
  - for the second room ($clinic = 1$), from 10 to 14.
- **ann** represents the last announced number that was called for the selected room.
- **print** represents the last given number to a resident for the selected room.
- If enable (**en**) is 0, numerator will not respond to clock, **print** and **ann** will store their last values.
- There are 2 **mode**s in this numerator:
  - **0**: A new resident wants to take a queue number (from one of the two rooms).
  - **1**: An nurse wants to call a waiting resident from one of the two rooms.

Table 3: Input/Output Specifications of numerator module

Name	**Type**	**Size**
clinic	Input	1 bit
mode	Input	1 bit
en	Input	1 bit
clk	Input	1 bit
ann	Output	8 bits
print	Output	8 bits

HINT: Connect **clk** to all submodules directly. Build combinational logic circuits (continuous assignment) to connect the enable inputs of submodules.

Table 4: Sample inputs and outputs for the numerator module.

Current State			clk	Next State		Explanation
en	**mode**	**clinic**		**ann**	**print**	
0	X	X	$*$	X (no change)	X (no change)	Initial condition
			$\uparrow$	X (no change)	X (no change)	$\sim$
1	1	0	$*$	5	5	A nurse wants to call a resident waiting for clinic 0, which is empty.
			$\uparrow$	5	5	$\sim$
1	1	1	$*$	10	10	A nurse wants to call a resident waiting for clinic 1, which is empty.
			$\uparrow$	10	10	$\sim$
1	0	0	$*$	5	5	A resident wants to take a queue number for clinic 0.
			$\uparrow$	5	6	$\sim$
1	0	0	$*$	5	6	Another resident wants to take a queue number for clinic 0.
			$\uparrow$	5	7	$\sim$
1	0	1	$*$	10	10	A resident wants to take a queue number for clinic 1.
			$\uparrow$	10	11	$\sim$
1	1	1	$*$	10	11	A nurse calls a resident waiting for clinic 1. Clinic 1 becomes empty after rising edge of the clk.
			$\uparrow$	11	11	$\sim$
1	1	1	$*$	11	11	A nurse wants to call a resident waiting for clinic 1, which is empty.
			$\uparrow$	11	11	$\sim$
1	1	0	$*$	5	7	A nurse calls a resident waiting for clinic 0.
			$\uparrow$	6	7	$\sim$
1	1	0	$*$	6	7	A nurse calls the remaining resident waiting for clinic 0. Clinic 0 becomes empty after rising edge of the clk.
			$\uparrow$	7	7	$\sim$
0	x	x	x	7(no change)	7(no change)	Disable the numerator

In **clk** column of Table 4, "$\uparrow$" represents the rising edge of the clock, "$*$" represents otherwise.

NOTE: While grading, the values of **ann** and **print** will be checked after the rising edges.

3.3 Deliverables

- Implement both modules in a single Verilog file: **lab3_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.
- Any kind of cheating is not allowed.