Middle East Technical University - Department of Computer Engineering

CENG 232

Logic Design

Spring 2024-2025

Lab 3

Due Date: 18 May 2025, Sunday, 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 requires you to implement some simple operations for a transportation firm.

2 Part 1: Warm-up (40 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 MD Flip-Flop Module

Implement the following MD flip-flop in Verilog concerning the provided characteristic table in Table 1. The MD flip-flop has 4 operations when inputs M and D are: **00 (set to 0), 01 and 10 (complement), 11 (set to 1)**.

Please note that the MD flip-flop changes its state **only at rising clock edges**. **Initially, the output of the flip-flop should be set to one**.

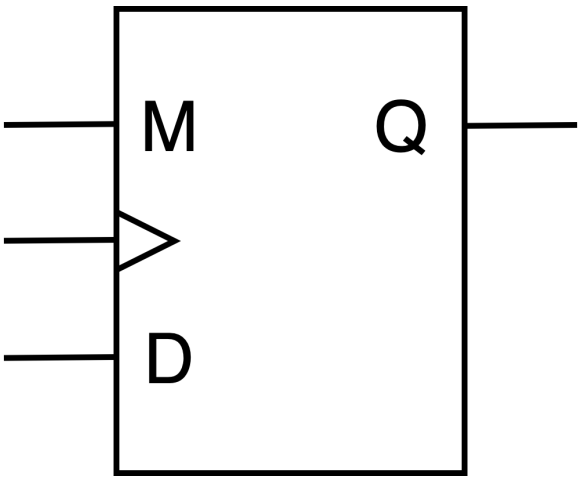

Figure 1: MD Flip-flop diagram

M	D	Q	Q_{next}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 1: MD Flip-flop characteristic table

2.2 ic4706 Module

Implement ic4706 chip given in Figure 2 that contains two MD flip-flops, with A_1, A_2, A_3, and clk as inputs; and S_3, S_4 and Z as outputs.
Use the following module definitions:

```
module md(input M, input D, input clk, output reg Q)
module ic4706(input A1, input A2, input A3, input clk, output S3, output S4, output Z)
```

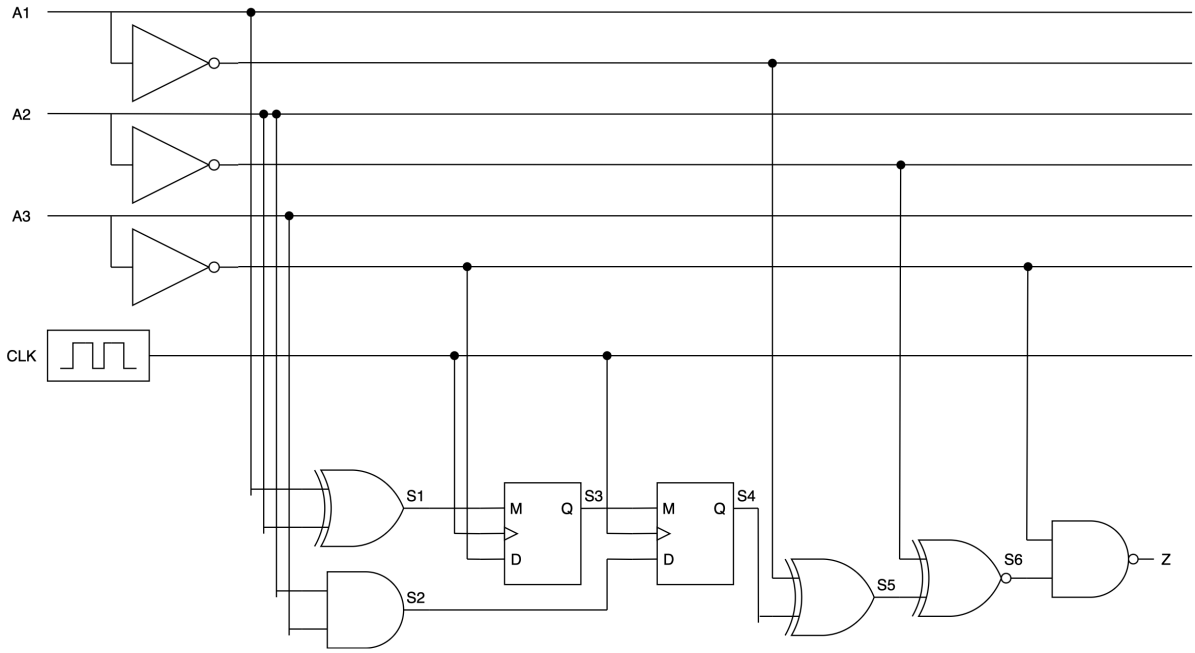

Figure 2: ic4706 module, inputs and outputs.

2.3 Simulation

Sample testbenches for MD flip-flop module and ic4706 module will be provided to you. You may want to extend the testbenches.

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.
- You can work in groups of three (the FPGA groups). A single grade will be given to each group.
- Implementations will be tested using custom test benches in a black-box fashion. We will use **Xilinx**. Please make sure that your code compiles over this tool.

3 Part 2: Transportation Firm (60 Pts)

In this part, you are supposed implement some simple operations for a transportation firm that operates between Ankara and Istanbul. For simplicity your operations will include only one way, i.e. from Ankara to Istanbul. There are three ways of commutation; by bus, train or airplane. The train and the airplane offers business seats along with normal seats.

3.1 System Overview

The specifications of each transportation method is as follows;

1. **Bus:** The most basic transportation method. Has 40 seats. The cost of a seat is *5 Lira*.
2. **Train:** Faster and more comfortable than the bus. Has 30 normal seats and 20 business seats. Normal seats cost *10 Lira* and business seats cost *15 Lira*.
3. **Airplane:** The quickest and the most comfortable method, but more expensive than the other two. Has 30 normal seats and 5 business seats. Normal seats cost *15 Lira* and business seats cost *25 Lira*.

3.2 Operational Overview

- Each **area category** is identified by a unique **2-bit code**.
- The system operates with cash, represented as a **5-bit digital input**.
- Following the insertion of cash and transportation method selection, users may choose a business seat, depending on availability.
- The system must internally **track the occupancy of seats, updating availability after each reservation.**

The following table summarizes the seating options across the different methods, along with the costs:

Table 2: Seating options, costs and selection bits

Area	Select Bits	Normal Seats	Business Seats	Normal Cost	Business Cost
Bus	00	40	0	5 Lira	—
Train	01	30	20	10 Lira	15 Lira
Airplane	11	30	5	15 Lira	25 Lira

3.2.1 Example Transaction

Consider a user intending to travel by train, on a business seat, inserting 25 Lira:

1. User selects the train (`01`) and indicates a preference for a business seat (`1`).
2. System checks if a business seat is available and if 25 Lira suffice for entry.
3. Upon confirmation, the system deducts the fee, reserves the seat, and returns the change.
4. Outputs:
   - `seatReady`: Enabled.
   - `moneyLeft`: 10 Lira.
   - `seatLeft` (business seats on the train): Decremented by 1.

3.3 Input/Output Specifications

The following table summarizes the necessary operations for the system. A more detailed explaination for each item is provided in the next subsection.

Table 3: Hierarchy of System Warnings

Priority	Warning Signal	Description
1	`seatUnavailable`	No more seats available in the selected method.
2	`invalidBusinessSeat`	Attempt to reserve a business seat with where none exists (bus).
3	`selectedSeatUnavailable`	All selected seats for that type (business or normal) are reserved.
4	`insufficientFund`	Inserted money is less than the required fee.

3.4 Input and Warning Descriptions

All system operations are synchronized to the positive edge of the system clock.

The seating system operates based on the following inputs and warnings:

- **money**: A 5-bit input representing the cash amount uploaded by the user.
- **CLK**: The clock input that synchronizes the module's operations.
- **businessAvailability**: A binary input indicating whether a business seat is requested.
- **selectedMethod**: A 2-bit input code for the transportation method:
  - `00` ⇒ Bus
  - `01` ⇒ Train
  - `11` ⇒ Airplane
- **seatUnavailable**: A warning signal indicating that no more seats are available for the selected method.
- **invalidBusinessSeat**: Raised when a business seat is requested for a bus, since no such seats exist.
- **selectedSeatUnavailable**: Indicates that all seats for the selected method and type are already reserved. An example for this warning can be when a customer tries to reserve a business seat when no business seat is available, however there are normal seats available. Note that this is different than "seatUnavailable" warning, where no seat is available for that method.
- **insufficientFund**: Triggered if the inserted money is less than the cost.
- **seatReady**: Enabled when a reservation can be successfully processed.

Additionally, **the system handles warnings based on a predetermined priority**, ensuring that the most critical alerts are addressed first. In the event of any warning, the system withholds processing new reservations until the subsequent positive clock edge.

Table 4: System Operations

Name	Type	Size
money	Input	5 bits
Clock (CLK)	Input	1 bit
businessSeatChoice	Input	1 bit
selectedMethod	Input	2 bits
moneyLeft	Output	5 bits
seatLeft	Output	6 bits
seatUnavailable	Output	1 bit
invalidBusinessSeat	Output	1 bit
selectedSeatUnavailable	Output	1 bit
insufficientFund	Output	1 bit
seatReady	Output	1 bit

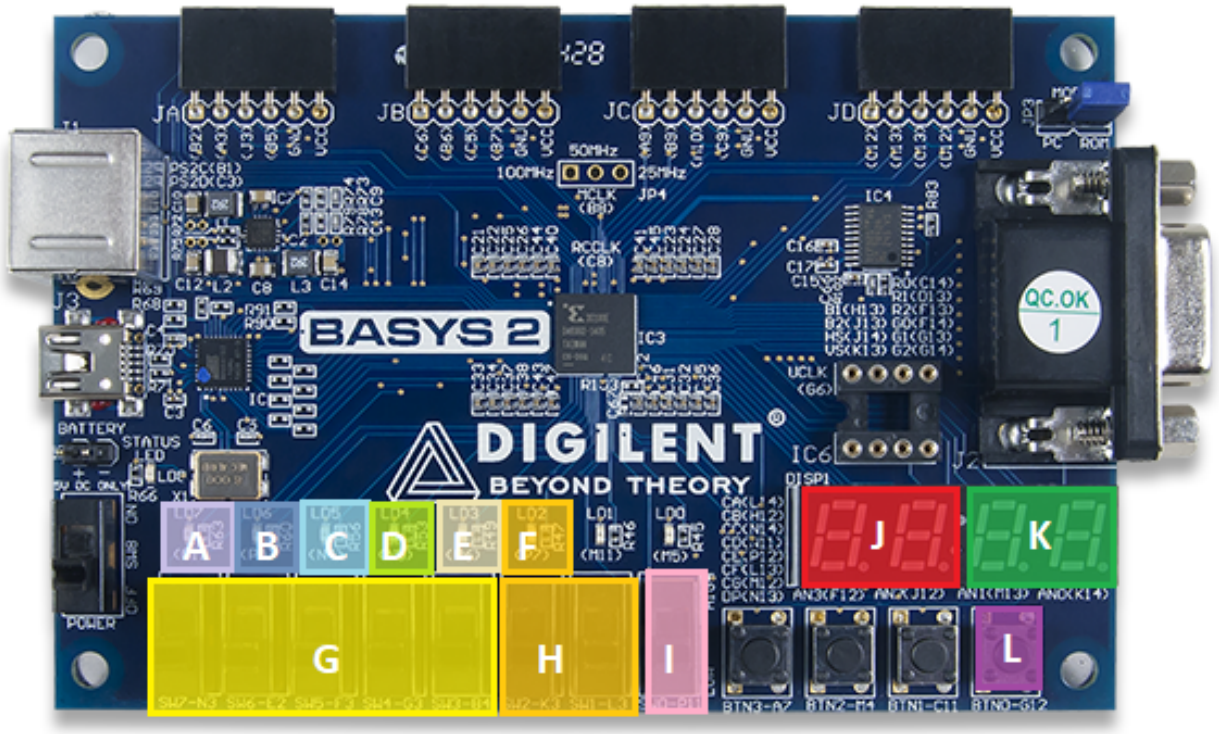

Figure 3: Board figure with labels

3.5 FPGA Implementation

You will be provided with a Board232.v file (and a ready-to-use Xilinx project), which will bind the inputs and outputs of the FPGA board with your Verilog module. You are required to test your Verilog module on the FPGA boards. Note that there is a single gap (E) between warnings and seatReady signal. This is intentional and it helps to easily separate the seatReady signal from the warnings.

Name	**FPGA Board**	**Description**	
seatUnavailable	led[7]	(A)	Top-most LED
invalidBusinessSeat	led[6]	(B)	Second LED from the top
selectedSeatUnavailable	led[5]	(C)	Third LED from the top
insufficientFund	led[4]	(D)	Fourth LED from the top
seatReady	led[2]	(F)	Sixth LED from the top
money	sw[7:3]	(G)	5 left-most switches
selectedArea	sw[2:1]	(H)	Middle switches
businessSeatChoice	sw[0]	(I)	Right-most switch
moneyLeft		(J)	7-segment display left 2 digits
seatLeft		(K)	7-segment display right 2 digits

Table 5: Module I-O to FPGA Board I-O Mappings

3.6 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 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.
- You can work in groups of three (the FPGA groups). A single grade will be given to each group.
- Implementations will be tested using custom test benches in a black-box fashion. We will use **Xilinx**. Please make sure that your code compiles over this tool.

4 Template Files

For each part of the assignment, specific template files are provided to assist you in starting your projects. These files are set up with the necessary configuration.

4.1 Part 1: Flip-Flop and IC Implementation

For the first part of the homework, the following template files are provided:

- **lab3_1.v**: This Verilog file defines the modules for the MD flip-flop and the ic4706. You will implement your solution here.
- **test.v**: Includes a test bench for the MD flip-flop.
- **test_ic4706.v**: Includes a test bench for the ic4706 module.

4.2 Part 2: Seating System Implementation

For the second part, the templates are as follows:

- **Board232.ucf**: Defines the board specifics for the FPGA. This file is provided for FPGA testing and requires no modifications.
- **Board232.v**: Defines the FPGA board implementation for testing purposes. This file should not be changed.
- **lab3_2.v**: This is the Verilog module that you need to implement for the seating system.
- **tester.v**: A simple tester provided for `lab3_2.v`.

5 Grading Criteria

You can work in groups of three (the FPGA groups). A single grade will be given to each group. A single submission for a group will be enough.

Part 1: For this part, your task involves simulating the required circuits using test benches. The grade for this part is based entirely on the correctness of your **simulations**. Each test case will examine specific outputs (i.e., S_3, S_4, and Z). To receive credit for a test case, **all outputs must exactly match the expected results** as specified in the provided tester files. This part will employ **black-box testing** through the test benches.

Part 2: In contrast to Part 1, the second part of the assignment requires your implementations to be **validated both through simulation test benches and physically on FPGA boards**. Each test case for this part will evaluate a subset of outputs. The total points for each test case will be divided equally among these output tests, allowing for **partial credit for each correctly implemented output**. Similarly, FPGA board tests will follow the same methodology, with partial credits awarded where appropriate.

Important Note: The test benches provided are not exhaustive. Successfully passing all the provided test cases does not guarantee full credit, as they do not cover all possible scenarios. You are encouraged to extend these test benches.

6 Plagiarism

For any questions or discussions, if they do not contain specific code snippets etc., please use the discussion thread on `ODTUClass` so that everyone can benefit and be kept up-to-date. In such cases where you need to share specific code/pseudo code (anything not abstract regarding your solution), please do not hesitate to reach out via e-mail (mduymus@ceng.metu.edu.tr & kursad@ceng.metu.edu.tr). When sending an email please CC us both.

You are encouraged to answer each other's questions and discuss among yourselves provided you do not share part of a solution with another group. Such actions of sharing are not to be tolerated. Similarly, submitting someone else's work in part or whole is a direct violation of academic integrity and will be subject to disciplinary action. This also holds for online/AI sources, including Google, ChatGPT, and Copilot. **Make use of them responsibly** - refrain from generating the code you are asked to implement. Remember that we also have access to these tools, making it easier to detect such cases. Your implementations will be subject to similarity checks through advanced tools and those that show high levels of similarity may be considered instances of plagiarism. **In short, please do not resort to practices violating your integrity - we are here to help.**