Middle East Technical University - Department of Computer Engineering

CENG 232

Logic Design

Spring '2024-2025

Lab 4

Part 1 & Part2 Due Date: 3 June 2025, Tuesday, 23:55
No late submissions

1 Part 1: Incremental Memory System (IMS) (30 Points)

1.1 Introduction

In this part, you will implement an Incremental Memory System (IMS) that introduces real-time data accumulation capabilities. This system is designed to support applications that require continuous data updates, such as counters, financial data tracking, and sensor data processing. The IMS combines a Read-Only Lookup Memory (ROLM) for reference values and an Incremental Data Storage (IDS) module for dynamic data accumulation.

1.2 Module Descriptions

1.2.1 Read-Only Lookup Memory (ROLM) Module

The ROLM module is a specialized read-only memory component with the following specifications:

- 3-bit addressing (addr) for a total of 8 memory locations.
- Stores 8-bit data values, predefined at initialization.
- Operates asynchronously, meaning it is not triggered by a clock pulse.
- Returns the data stored at the specified address immediately upon receiving the address input.

Initial Data Values:

Table 1: Initial Values Stored in the ROLM

Address	Data Value (8-bit)
0	11011011
1	10101010
2	01111110
3	00001111
4	10011001
5	11100011
6	01010101
7	00110011

Port Definition:

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

1.2.2 Incremental Data Storage (IDS) Module

The IDS module is a specialized data storage component that performs real-time data accumulation and adjustment based on the inputs provided. It supports both synchronous and asynchronous operations:
Functional Specifications:

- 3-bit addressing (addr) for 8 memory locations, each containing 8 bits of data.
- **Write Mode:** Performs specified operations on the input data (`dataIn`) based on the current operation code and the corresponding ROM data. Write mode **works synchronously**, triggered by the positive edge of the clock signal (`CLK`).
- **Read Mode:** Simply retrieves and outputs data stored at a specified memory address. Read mode **works asynchronously**.

Operation Codes:

- 00 ⇒ Increment (Adds the input data to the ROM data at the specified address)
- 01 ⇒ Decrement (Subtracts the ROM data from the input data)
- 10 ⇒ Reset (Sets the memory location to 0, effectively clearing the stored data)
- 11 ⇒ Preset (Sets the memory location to a predefined constant, e.g., 8'b11111111)

Port Definition:

```
module IDS (
input mode,                     // 0 for write, 1 for read
input [2:0] addr,               // Memory address
input [1:0] operation,          // Operation code
input [7:0] dataIn,             // Input data
input [7:0] romData,            // Data from ROM used in operations
input CLK,                      // Clock signal
output reg [7:0] dataOut        // Output data
);
```

1.2.3 Incremental Memory System (IMS) Module

This top-level module integrates the ROLM and IDS modules, coordinating the data flow between these components to achieve efficient memory operations.
Port Definition:

```
module Incremental_Memory_System (
input mode,                         // W/R mode of IDS
input [2:0] systemAddr,             // System address
input [1:0] operation,              // Operation code
input [7:0] dataIn,                 // System data input
input CLK,                          // Clock signal
output [7:0] systemOutput           // System output
);
```

1.3 Deliverables

- Implement both modules in a single Verilog file. Upload only **Lab4_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.

2 Part 2: Transportation System (70 Points)

2.1 Introduction

In this part of the assignment, your task is to implement a unified Transportation System for a firm operating between Ankara and Istanbul. The system supports four modes of transportation: Bus, Train, Airplane, and High-Speed Train, each with different seating capacities, and seat numbering. The goal is to manage seat reservations, passenger check-ins, and seat searches efficiently. This system is designed to operate in multiple modes, each tailored for specific operations, similar to the seating system introduced in Lab 3.

2.2 System Overview

The specifications of each transportation method are as follows:

- **Bus:** The most basic transportation method. Has 16 seats, numbered from 0000 to 1111.
- **Train:** Faster and more comfortable than the bus. Has 8 normal seats (0000 to 0111) and 8 business seats (1000 to 1110).
- **Airplane:** The quickest and most comfortable method, but more expensive than the other two. Has 8 normal seats (0000 to 0111) and 8 business seats (1000 to 1110).
- **High-Speed Train:** Balances speed and comfort, with 16 seats, numbered from 0000 to 1111.

Seating Options:

Table 2: Seating options, costs, and selection bits

Method	Select Bits	Normal Seats	Business Seats
Bus	00	0000 to 1111	-
Train	01	0000 to 0111	1000 to 1110
Airplane	11	0000 to 0111	1000 to 1110
High-Speed Train	10	0000 to 1111	-

2.3 Module Definition

```
module TransportationSystem(
input  [1:0] mode,         // Modes: 00 for Registration, 01 for Check-In, 10 for Search
input  [1:0] method,       // Transportation method (Bus, Train, Airplane, High-Speed Train)
input  [3:0] seatNumber,   // 4-bit seat number (0000 to 1111)
input CLK,                // Clock signal
output reg  [1:0] selectedMethod, // Shows user-selected method
output reg  [5:0] seatLeft,        // Number of available seats in the selected method
output reg registered,            // Indicates if a passenger is registered
output reg checkInStatus,         // Indicates if a passenger has checked in
output reg alreadyRegistered,     // Indicates if a passenger is already registered
output reg notRegistered,         // Indicates if a passenger is not registered
output reg seatUnavailable        // Indicates if the selected seat is unavailable
output reg alreadyCheckedIn       // Indicates if a passenger has already checked in
);
```

2.4 System Specifications

1. **I/O Specifications:**
   (a) `Inputs:`
   - `mode`: Indicates the mode of the system (00 for Registration, 01 for Check-In, 10 for Search).
   - `method`: A 2-bit code representing the selected transportation method (00 for Bus, 01 for Train, 10 for High-Speed Train, 11 for Airplane).

- `seatNumber`: A 4-bit number representing the seat number (0000 to 1111).
- `CLK`: System clock signal.

(b) `Outputs:`

- `selectedMethod`: 2-bit output indicating the transportation method selected by the passenger.
- `seatLeft`: 6-bit output showing the current number of available seats in the selected method.
- `registered`: Flag indicating if a passenger is registered.
- `checkInStatus`: Flag indicating if a passenger has checked in.
- `alreadyRegistered`: Flag indicating if a passenger is already registered.
- `notRegistered`: Flag indicating if a passenger is not registered.
- `seatUnavailable`: Flag indicating if the selected seat is unavailable.
- `alreadyCheckedIn`: Flag indicating if a passenger is already checked in.

2.4.1 Modes of Operation

(a) **Registration Mode (00):**

- Allows a passenger to register for a seat if it is available.
- Updates the seat availability and registration status.
- Issues an `alreadyRegistered` warning if the passenger is already registered.

(b) **Check-In Mode (01):**

- Allows a registered passenger to check in to their reserved seat.
- Updates the check-in status for the passenger.
- Issues a `notRegistered` warning if the passenger is not registered.
- Issues an `alreadyCheckedIn` warning if the passenger is already checked in.

(c) **Search Mode (10):**

- Allows the system to check if a specific seat is registered and occupied.
- Updates `alreadyRegistered`, `notRegistered`, and `alreadyCheckedIn` accordingly.

2. **Seat Numbering:** Each transportation method has up to 16 seats. Bus and High-Speed Train use the full range from 0000 to 1111, while Train and Airplane use seats from 0000 to 1110 (15 seats).
3. **Initial Conditions:** All seats are initially unregistered and unoccupied.

2.5 Input/Output Specifications

Type	Name	Description
Input	`mode`	2-bit mode selector
Input	`method`	2-bit method selector
Input	`seatNumber`	4-bit seat number (0000 to 1111)
Input	`CLK`	Clock signal
Output	`selectedMethod`	2-bit output showing method
Output	`seatLeft`	6-bit output showing available seats
Output	`registered`	Flag for passenger registration
Output	`checkInStatus`	Flag for passenger check-in
Output	`alreadyRegistered`	Flag for existing registration
Output	`notRegistered`	Flag for missing registration
Output	`seatUnavailable`	Flag for unavailable seats
Output	`alreadyCheckedIn`	Flag for already checked-in passengers

Table 3: Input/Output Specifications of the Transportation System Module

2.6 Warning Priorities

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.

Priority	Warning Signal	Description
1	`seatUnavailable`	No more seats available in the selected method.
2	`alreadyRegistered`	Attempt to register a seat that is already reserved.
3	`notRegistered`	Attempt to check in or search for a seat that is not registered.
4	`alreadyCheckedIn`	Attempt to check in to a seat that is already checked in.

Table 4: Hierarchy of System Warnings

2.7 FPGA Implementation

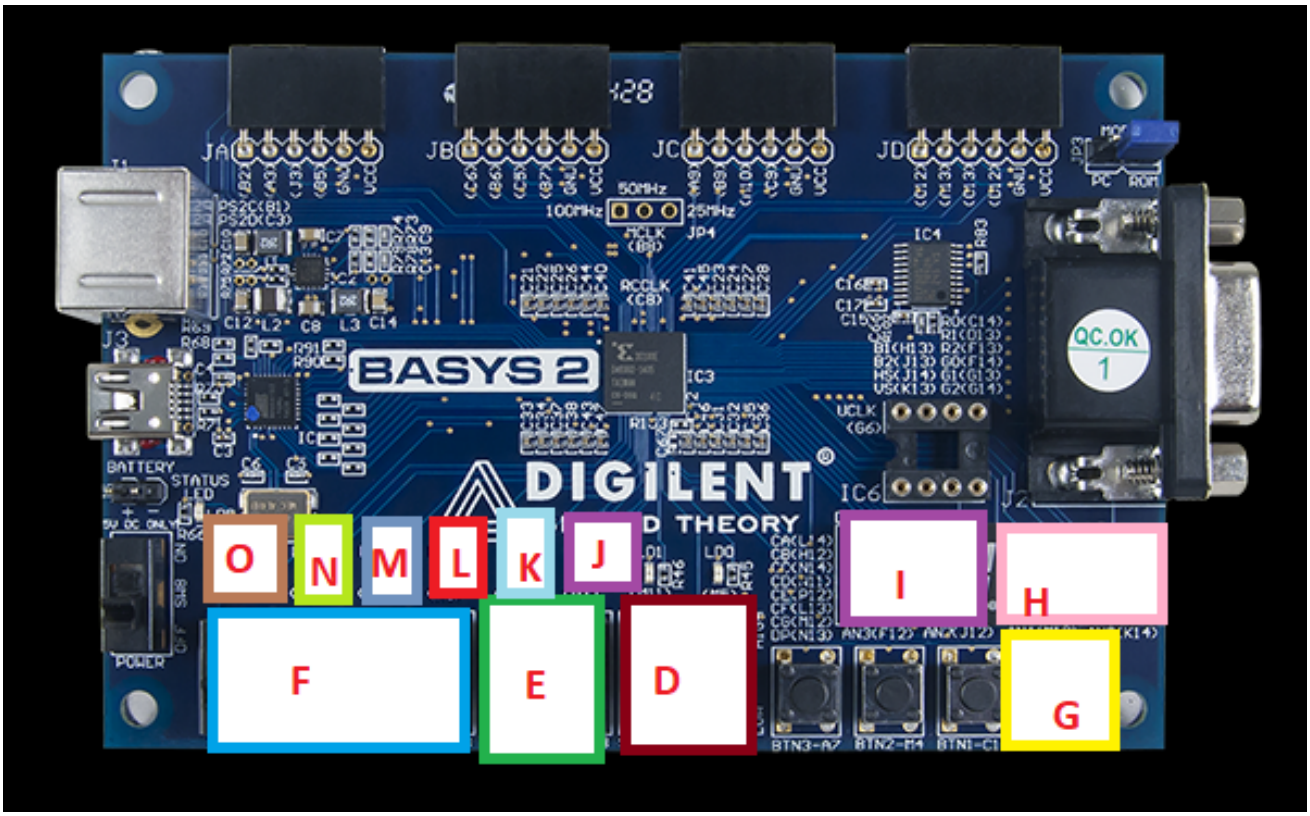

Figure 1: Board figure with labels

Name	FPGA Board	Description	Alphabets
mode	sw[1:0]	Mode Selector (Middle switches)	D
method	sw[3:2]	Transportation Method Selector	E
seatNumber	sw[7:4]	Seat Number (0000 to 1111)	F
CLK	btn[0]	Clock Signal (Button)	G
seatLeft	7-seg (right 2 digits)	Available seats count	H
selectedMethod	7-seg (left 2 digits)	Selected transportation method	I
alreadyCheckedIn	led[2]	Already checked-in flag	J
seatUnavailable	led[3]	Unavailable seats flag	K
notRegistered	led[4]	Missing registration flag	L
alreadyRegistered	led[5]	Existing registration flag	M
checkInStatus	led[6]	Passenger check-in flag	N
registered	led[7]	Passenger registration flag	O

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

Note: The right bottom button is CLK.

2.8 Deliverables

- Implement your module in a single Verilog file. Upload only **Lab4_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.

3 Testbenches & Simulation

You are provided with testbenches to test your implementations in a simulation environment, however, please note that they do not cover all possible cases. That is, passing all test cases provided does not guarantee a full credit. Hence, you are recommended to extend the testbenches provided.

4 Grading Criteria

The grading for this assignment is structured as follows:

1. **Part 1 (30 Points):** Part 1 of the assignment will be evaluated through black box test benches in a simulation environment.
2. **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.

5 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 (ctekinbas@ceng.metu.edu.tr & selim@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.**