# COL215 – Digital Logic and System Design
# Department of Computer Science & Engineering, IIT Delhi
# Semester I, 2025-26
# Lab Assignment 6

Memory Read and Write

## 1 Introduction

In this assignment, we will be working with three memory blocks: a Read-Only Memory (ROM) and two Random-Access Memories (RAMs). The core task is to create a control unit that handles reading from and writing to these memory blocks based on user inputs.
Our task is to design a module that performs vector addition of two given input vectors, defined as follows.

$$C[i] = A[i] + B[i],\ 0 \leq i \leq 1023$$

where,
A: Input vector of 1024 4-bit elements stored as $ROM$
B: Input vector of 1024 4-bit elements stored as $RAM0$
C: Output vector of 1024 5-bit elements stored as $RAM1$

Along with addition, an increment operation should also be supported over input vector elements. Also give a proper indication on the four 7-segment display in case of reset condition, and display the content when read.

## 2 Problem Description

### 2.1 Switch and pin functions

The functionality of different IO pins and switches is listed in Table 1.

Table 1: Pin and Switch Function Assignments

| Pin / Switch | Function |
|---|---|
| $SW3 - SW0$ | Used to provide 4-bit input number, $B[i]$ |
| $SW13 - SW4$ | Address of the input number, $i$ |
| $SW15 - SW14$ | 2-bit index to select the element to be displayed:<br>`00`: Nothing. Display is inactive.<br>`01`: Read $ROM$, $RAM0$, $RAM1$:<br>- Seven Segment Digit 0: Display value of $ROM$<br>- Seven Segment Digit 1: Display value of $RAM0$<br>- Seven Segment Digit 2 and 3: Display value of $RAM1$<br>`10`: Write $RAM0$: Value on $SW3 - SW0$ should get written at $SW13 - SW4$ address.<br>`11`: Increment $RAM0$: Increment value stored at $SW13 - SW4$ address by 1.<br>Use your knowledge about storing data onto registers to ensure the write and increment operations happen once and not multiple times. |
| $BTNC$ | Reset. Displays '`-rSt`' on the 7-segment display for 5 seconds. |

As an example, a setting with

- $BTNC$ not pushed, i.e., OFF
- $SW15$–$SW14$ as ON, OFF, respectively, SW13–SW4 as ON, ON, OFF, OFF, ON, ON, OFF, OFF, OFF, OFF, respectively, (i.e., 10'b1100110000 or 'd816),
- SW3–SW0 as OFF, ON, OFF, ON, respectively, (i.e., 4'b0101 or 'd5)

stores 5 at the $816^{th}$ index of vector B.
If the value stored in vector A at the $816^{th}$ index is $F_{16}$. Then, the value stored in the vector C at $816^{th}$ index would be $5_{16} + F_{16} = 14_{16}$.
Now, if the position of SW15–SW14 is changed to OFF, ON, respectively, while the others are kept the same, the 7-segment display should display as shown in Figure 1.

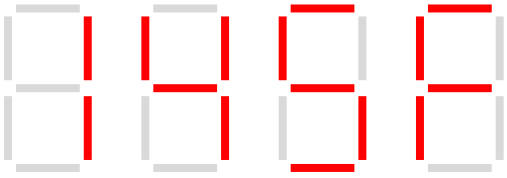

Figure 1: Seven Segment Display output of the example input

## 2.2 Modules

The overall design would consist of the following:

1. A Read-Write Memory (RWM, more popularly known as RAM or Random-Access Memory) - You will be using this memory to store the input vector and modify that based on user inputs via switches. A second similar RAM will be used to store the outputs.
2. A Read-Only Memory (ROM) - You will be using this memory to store the input vector.
3. Registers - You will be using these to store temporary results across clock cycles.
4. Seven Segment Module - You will be using this module to display the values in the seven segment displays. To be reused from the previous assignments.
5. An overall controller that stitches all modules together to perform the vector addition.

## 2.3 Memories

We need memories to store the input and output vectors.

- The input vector, $A$, is stored in a $ROM$.
- The input vector, $B$, is stored in a $RAM0$.
- The output vector, $C$, is stored in a $RAM1$.

### 2.3.1 Read-Only Memory (ROM)

We will use RAM0 for storing:

- Vector $A$, of length 1024 and consisting of 4-bit elements. These will be stored at address 0 ($000_{16}$) onwards. The address width of the memory should be at least 10 bits.

### 2.3.2 Random-Access Memory or Read-Write Memory (RAM)

We will use two RAMs, RAM0 and RAM1, to store the following:

- RAM0: Vector $B$, of length 1024 and consisting of 4-bit elements. These will be stored at address 0 ($000_{16}$) onwards. The address width of the memory should be at least 10 bits.
- RAM1: Vector $C$, of length 1024 and consisting of 5-bit elements. These will be stored at address 0 ($000_{16}$) onwards. The address width of the memory should be at least 10 bits.

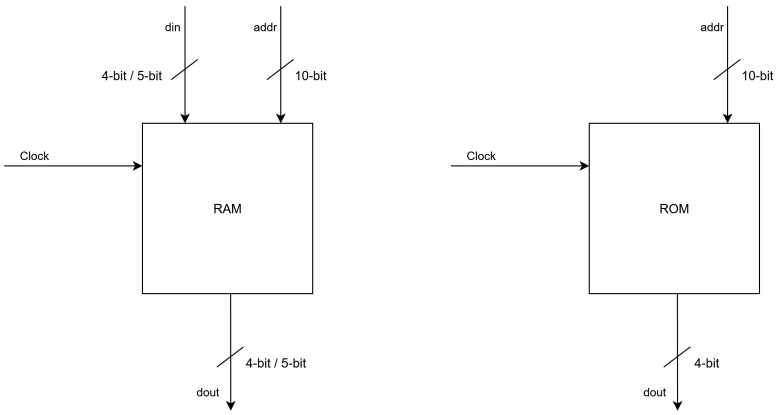


Figure 2: RAM and ROM block diagrams

### 2.3.3 Generate the memory block using Vivado IP catalog

We will use Vivado's memory generator to design program and data memories. Instructions are as follows:

- In Vivado, go to the Project Manager window on the left side of the screen as shown in Fig. 3.
- Open IP catalog. You can also open IP Catalog through a drop down menu after clicking on Window in the main menu on the top of the screen. "IP" (Intellectual Property) refers to pre-designed modules available for use. The IP catalog lists various IPs category-wise.

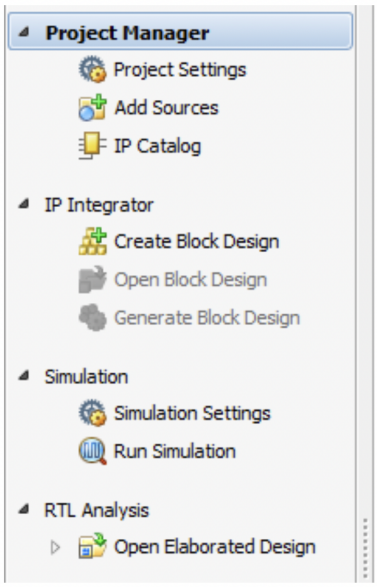


Figure 3: Project Manager

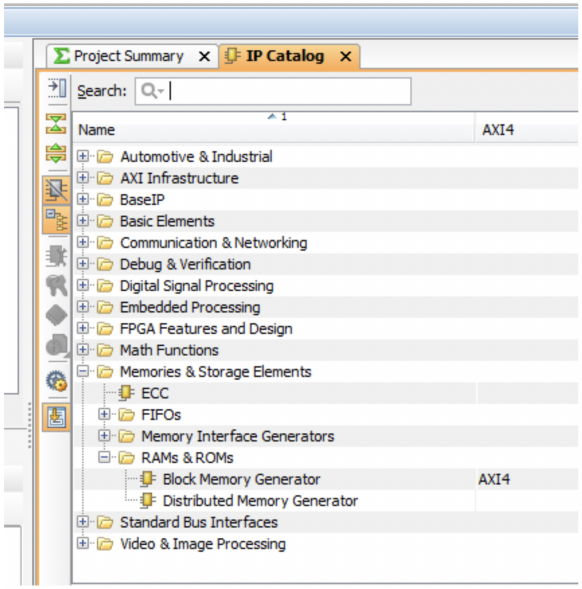


Figure 4: IP Catalog

- Select the category **Memories & Storage Elements** and sub-category **RAMs & ROMs**, as shown in Fig. 4.
- Now choose **Distributed Memory Generator**, to create a ROM / RAM as per your design requirement.

  The Distributed Memory Generator opens a window with three tabs. In the **memory config tab**, specify memory size and type. The Fig. 5 shown below is for a ROM with 256 words, each of size 32 bits. You should change these parameters based on the input vector size that is ROM with 1024 words with size of 4 bits.

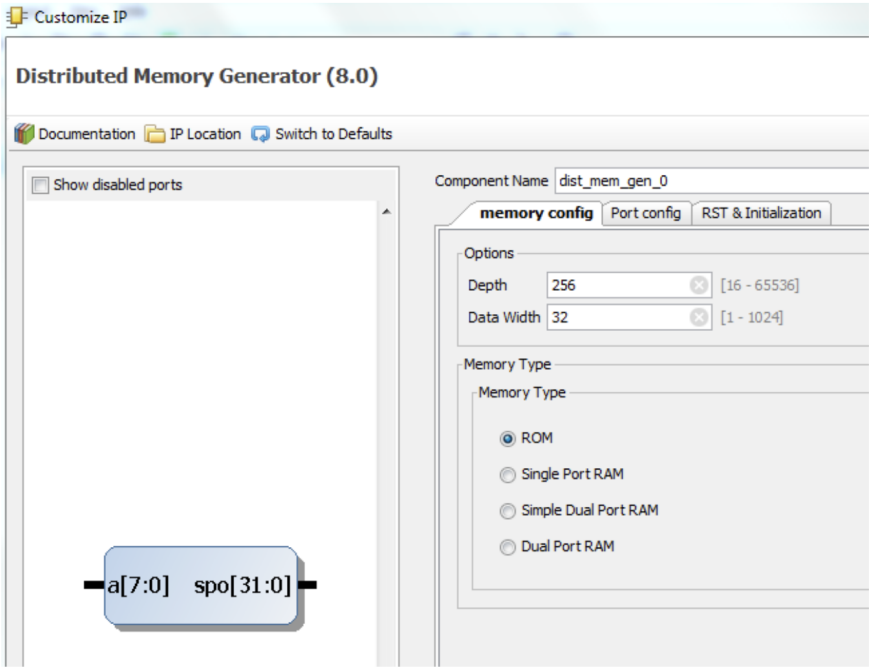


Figure 5: Customize IP: Distributed Memory Generator

- In the **Port config** tab, select output registered option. This will add a 'clk' port to the block generated.
- In the **RST & Initialization** tab, specify name of the file that defines the memory contents. Sample files named *ram_vector.coe* and *rom_vector.coe* are available on Moodle.

  Now instead of passing the inputs from the test bench or using *force value* construct, initialize the inputs using a *coe* file and pass input to the Design Under Test (DUT) by connecting it to the memory module.

  Once you synthesize and implement your design, you will see a change in the number of LUTs used in your design.
- Use the following test bench to simulate the memory IP block in Vivado. Below is a snippet of how to import the generated memory block into your Verilog design file.

To use the ROM module generated,

```
// Signals in your top-level module
reg  [9:0] address;   // 2^10 = 1024, so we need a 10-bit address
wire [3:0] data_out;  // 4-bit data output

// Instantiate the Distributed Memory Generator IP
// Note: Replace 'dist_mem_gen_0' with the actual name of your IP instance.
dist_mem_gen_0 rom_instance (
  .clk(clk),        // Connect to your system clock
  .a(address),      // Connect to your 10-bit address bus
  .qspo(data_out)   // Connect to the data bus
);
```

To use the RAM module generated,

```
// Signals in your top-level module
reg         clk;
reg  [9:0]  address;    // 10-bit address for 1024 locations
reg         write_en;   // Your logic's write enable signal
reg  [3:0]  data_in;    // 4-bit data to be written
wire [3:0]  data_out;   // 4-bit data read from the RAM

// Instantiate the Distributed Memory Generator IP
// Note: Replace 'dist_mem_gen_0' with the actual name of your IP instance.
```

```
dist_mem_gen_0 ram_instance (
  .clk(clk),        // Clock
  .we(write_en),    // Write Enable (1 to write, 0 to read)
  .a(address),      // Address Bus
  .d(data_in),      // Data In Bus
  .qspo(data_out)   // Data Out Bus
);
```

By default the ROM and RAM0 both will be loaded with their respective coe files and RAM1 reading RAM1 should indicate the appropriate values. Once the values in RAM0 are updated using switches, the RAM1 should indicate the updated values.

### 2.4 Four 7-segment display

Use your knowledge about the 7-segment display from previous assignments for this assignment. The 7-segment display can be used to display numbers in hexadecimal format as in Fig. 6

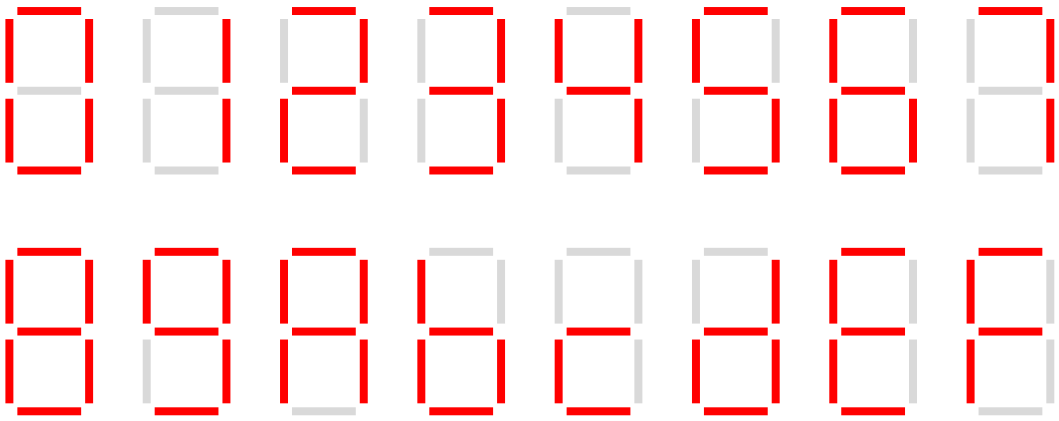

Figure 6: Seven Segment Display output of the hexadecimal values, 0-9 and A, F

When the reset button is pressed, the 7-segment display should display '`-rst`' as shown in Fig. 7 for 5 seconds.

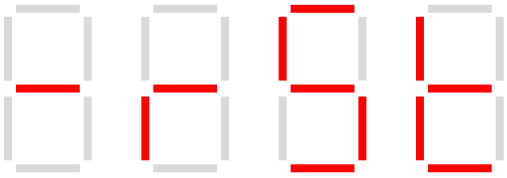

Figure 7: -rst

**The diagrams in this document are for illustration and explanation of the problem statement and do not represent the exact design to be implemented. The design choices can vary for each group and should be carefully explained in the assignment report.**

### 2.5 Output

- You need to verify the output using simulation and show waveforms in your report.
- The input vectors have been provided over Moodle (*ram_vector.coe* and *rom_vector.coe*). You will be loading these into the memory. You will be given 2/4 random vector indices and you will have to display the hexadecimal value of the values at that index on the 7-segment display on the board. You will also need to update the RAM0 values using switches and then display the values on the 7-segment dispaly unit.

## 3 Submission and Demo Instructions

1. Demo should be given in the assigned lab slot itself.

2. You are required to submit the following on Gradescope.

   - 20 points: Verilog files for all the designed modules. Only the modules written by you are needed. The IPs generated, i.e., distributed memories must not be submitted.
   - 20 points: Verilog files for the testbench and the simulation of reset, addition of default coe contents, addition after updating the values from switches, and increment operation.
   - 10 points: Warning-clean constraint file (.xdc), bit file.
   - 30 points: Questionnaire corresponding to the assignment.
   - 20 points: A short report (3-4 pages) outlining simulation snapshots, utilization data (BRAMs, DSPs, LUTs, and FFs), and generated schematics. Explain your design decisions. Having only snippets in the report (without explanation) will result in penalty.

   **NOTE:**

- Ensure all the files in the zip folder are correctly visible on gradescope. Later requests about technical glitches with gradescope and files being in zip folder and not visible on gradescope and being part of report wouldn't be considered anymore.
- The report must be in pdf format only (docx will not be considered anymore) and should consist of the points mentioned in the Submission section. Not mentioning utilization data specifically in the report (and relying on synthesis screenshots) will not be considered anymore.
- The submission must have only the above listed items. Submitting additional files such as video of board functionality, separate utilization report, vivado xpr or dcp files, etc. will result in penalty.

We advise you to be ready with your design before the lab session, and during the session, perform validation by downloading it into the FPGA board.

# 4 Resources references

- IEEE document: `https://ieeexplore.ieee.org/document/1620780`
- Basys 3 board reference manual: `https://digilent.com/reference/_media/basys3:basys3_rm.pdf`
- Online Verilog simulator: `https://www.edaplayground.com`