# Lab Member

Table Number : <24>

| Name | Student Number |
| --- | --- |
| Nantapong Ruangsuksriwong | 6632107621 |
| Worralop Srichainont | 6632200221 |
| Weerapat Lerdcharoenvorakul | 6632209021 |
| Pakapim Eua-anant | 6632162021 |

# 

# Objectives

1. Introducing the workflow of the FPGA development.
2. Understand the combinational logic on Verilog.

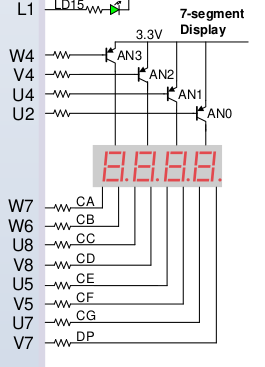
# 7-Segments Display on Basys3 Board

The Basys 3 board features a four-digit common anode seven-segment LED display. Each digit consists of 8 LEDs, corresponding to segments A, B, C, D, E, F, G, and DP, as illustrated in Figure 1. All LEDs within the same digit share a common anode. Additionally, identical segments across all digits share the same cathode; for instance, the "A" segments in all digits are connected to the same cathode pin, labeled CA. This configuration minimizes the number of pins required to control the 7-segment display.

# 

**Figure 1 : 7-Segment Display Configuration**

To illuminate each segment, the corresponding anode (AN) must be set to 0. As shown in Figure 2, each anode is connected to a transistor, and setting the corresponding cathode (CX) to 0 allows current to flow from the anode to the cathode, thereby turning on the segment.

****

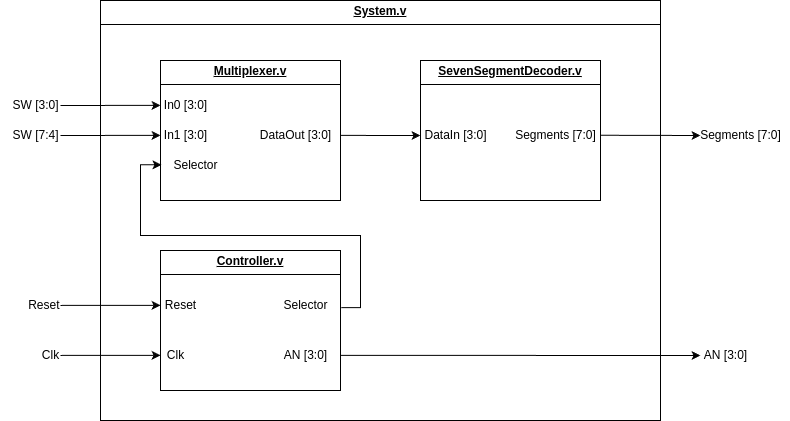
**Figure 2 : 7-Segment Display Pin**

To illuminate multiple digits simultaneously, a technique called multiplexing is used. This involves rapidly turning each digit on and off at a rate imperceptible to the human eye. However, this functionality is not the focus of this lab exercise and the module handling this functionality will be provided.

# 

# Lab Exercise

In this lab exercise, you are tasked with developing a system that takes input from 8 onboard switches (SW0 to SW7) to represent two hexadecimal values. SW0 to SW3 form the first hexadecimal value, with SW0 as the least significant bit (LSB), while SW4 to SW7 form the second hexadecimal value, with SW4 as the LSB. The system should display the first hexadecimal value on the first digit of the 7-segment display and the second hexadecimal value on the second digit. The third and fourth digits of the 7-segment display will not be used.



**Figure 3: System Overview.**

Figure 3 illustrates the System Overview to be implemented in this lab exercise. The system comprises four modules: **System**, **Multiplexer**, **SevenSegmentDecoder**, and **Controller**. The roles of each module are as follows:

1. **System Module**: Serves as the top-level module, integrating and connecting the other submodules.
2. **Multiplexer Module**: Selects which hexadecimal value is sent to the SevenSegmentDecoder module for translation into control signals for the 7-segment display.
3. **SevenSegmentDecoder Module**: Converts a 4-bit binary input into control signals for the segments of the 7-segment display.
4. **Controller Module**: Controls the on/off cycling of each digit on the 7-segment display to ensure all digits appear visible to the observer. It does this by sending a selector signal to the Multiplexer to determine the active switch data and manipulating the AN signal to turn each digit on or off. This module is pre-implemented for you.

## Part 1 : Multiplexer

In this part of the lab, you are tasked with completing the **Multiplexer** module. The module has 3 input ports: **In0[3:0]**, **In1[3:0]**, and **Selector**, and 1 output port: **DataOut[3:0]**. The module must be implemented as combinational logic according to the truth table provided below.

| Selector | DataOut [3:0] |
| --- | --- |
| 0 | In0 |
| 1 | In1 |

**Table 1: Multiplexer Truth table**

### Instruction

1. Create a new project and import all the necessary files from the following source: <https://github.com/2110363-HW-SYN-LAB/lab/tree/main/Lab1> or MCV. Follow the instructions provided in the Lab1 Guide file for setup.

2. Modify the **Multiplexer** module to implement the behavior described in the truth table above

Insert your modified Multiplexer module here

|  |
| --- |

3. Run the testbench to verify that your module functions correctly.

Insert your Xilinx testbench (MultiplexerTB) result here.

|  |
| --- |

Insert your Cocotb testbench result (from folder /cocotb/MultiplexerTB/) here.

|  |
| --- |

Insert your testbench result waveform here. (Either Cocotb or Xilinx Testbench)

|  |
| --- |

4. Call TA to inspect your work.

## 

## Part 2 : 7-Segment Decoder

In this part of the lab, you are tasked with completing the **SevenSegmentDecoder** module. This module has 1 input port: **DataIn[3:0]** and 1 output port: **Segments[7:0]**. The output segments are assigned as follows: **[CA, CB, CC, CD, CE, CF, CG, DP]**, where **CA** corresponds to **Segments[7]** and **DP** corresponds to **Segments[0]**.

This module must be implemented as combinational logic. The **DP** signal controls the decimal point (dot) on the 7-segment display. For this lab exercise, the decimal point should remain turned off.



**Figure 4 : 7-Segment Display of hexadecimal value**

### Instruction

1. Complete the truth table below, which converts 4 bit **DataIn** to the corresponding **Segments[7:0]**.

| DataIn [3:0] | Segments [7:0] |
| --- | --- |
| 0000 | 00000011 |
| 0001 | 10011111 |
| 0010 | 00100101 |
| 0011 | 00001101 |
| 0100 | 10011001 |
| 0101 | 01001001 |
| 0110 | 01000001 |
| 0111 | 00011111 |
| 1000 | 00000001 |
| 1001 | 00001001 |
| 1010 | 00010001 |
| 1011 | 11000001 |
| 1100 | 01100011 |
| 1101 | 10000101 |
| 1110 | 01100001 |
| 1111 | 01110001 |

3. Create a testbench for this module to verify that it works correctly. You will need to create both **Xilinx** and **Cocotb** testbenches:

Insert your Xilinx testbench file (SevenSegmentDecoderTB) here.

| `timescale 1ns / 1ps  *//////////////////////////////////////////////////////////////////////////////////*  *// Create Date: 12/23/2024 05:07:14 AM*  *// Design Name: Exercise1*  *// Module Name: SevenSegmentDecoderTB*  *// Project Name: Exercise1*  *// Target Devices: Basys3*  *// Tool Versions: 2023.2*  *// Description: Testbench for the SevenSegmentDecoder module*  *//////////////////////////////////////////////////////////////////////////////////*  module SevenSegmentDecoderTB ();  *// Declare the reg/wire*  reg [3:0] DataIn; *// Input to the SevenSegmentDecoder*  wire [7:0] Segments; *// Output from the SevenSegmentDecoder*  reg [7:0] segment\_values[15:0]; *// Expected segment patterns for each DataIn value*  *// Instantiate the SevenSegmentDecoder module*  SevenSegmentDecoder SevenSegmentDecoderInst (  .DataIn (DataIn),  .Segments(Segments)  );  *// Instantiate variable*  integer flag = 0; *// Flag to track if any test case fails*  integer TestCaseNo = 0; *// Counter for test cases*  integer i; *// Loop variable*  *// Task to check the output*  task check\_output;  input integer TestCaseNo;  input reg [7:0] expected\_Segments; *// Expected output*  begin  if (Segments !== expected\_Segments) begin  $error("ERROR: TestCaseNo %0d | Time = %0t | DataIn = %b | Segments = %b (Expected: %b)",  TestCaseNo, $time, DataIn, Segments, expected\_Segments);  flag = 1;  end  end  endtask  *// Test cases*  initial begin  *// Initialize the expected segment values*  segment\_values[4'b0000] = 8'b00000011; *// '0'*  segment\_values[4'b0001] = 8'b10011111; *// '1'*  segment\_values[4'b0010] = 8'b00100101; *// '2'*  segment\_values[4'b0011] = 8'b00001101; *// '3'*  segment\_values[4'b0100] = 8'b10011001; *// '4'*  segment\_values[4'b0101] = 8'b01001001; *// '5'*  segment\_values[4'b0110] = 8'b01000001; *// '6'*  segment\_values[4'b0111] = 8'b00011111; *// '7'*  segment\_values[4'b1000] = 8'b00000001; *// '8'*  segment\_values[4'b1001] = 8'b00001001; *// '9'*  segment\_values[4'b1010] = 8'b00010001; *// 'A'*  segment\_values[4'b1011] = 8'b11000001; *// 'b'*  segment\_values[4'b1100] = 8'b01100011; *// 'C'*  segment\_values[4'b1101] = 8'b10000101; *// 'd'*  segment\_values[4'b1110] = 8'b01100001; *// 'E'*  segment\_values[4'b1111] = 8'b01110001; *// 'F'*  *// Apply test cases for all DataIn values (0–15)*  for (i = 0; i < 16; i = i + 1) begin  DataIn = i[3:0]; *// Set the input*  #10; *// Wait for 10 time units*  check\_output(TestCaseNo, segment\_values[i]); *// Check output*  TestCaseNo = TestCaseNo + 1; *// Increment test case number*  end  *// Final test results*  if (flag == 0) begin  $display("All test cases pass");  end else begin  $display("Some test cases fail");  end  $finish;  end  endmodule |
| --- |

Insert your Cocotb testbench file (from folder /cocotb/SevenSegmentDecoderTB/) here.

| import cocotb  from cocotb.triggers import Timer  **@cocotb.test**()  async def SevenSegmentDecoderTB(dut):  """Try accessing the design."""  dut.\_log.info("Running test!")  *# create a testbench here*  test\_cases = {  0x0: 0b00000011, *# 0*  0x1: 0b10011111, *# 1*  0x2: 0b00100101, *# 2*  0x3: 0b00001101, *# 3*  0x4: 0b10011001, *# 4*  0x5: 0b01001001, *# 5*  0x6: 0b01000001, *# 6*  0x7: 0b00011111, *# 7*  0x8: 0b00000001, *# 8*  0x9: 0b00001001, *# 9*  0xA: 0b00010001, *# A*  0xB: 0b11000001, *# B*  0xC: 0b01100011, *# C*  0xD: 0b10000101, *# D*  0xE: 0b01100001, *# E*  0xF: 0b01110001, *# F*  }  for DataIn, expected\_segments in test\_cases.items() :  *#assign data input*  dut.DataIn.value = DataIn  *#wait to change*  await Timer(1, units='ns')  assert dut.Segments.value == expected\_segments    dut.\_log.info("Test Complete") |
| --- |

4. Modify the **SevenSegmentDecoder** module to work according to the truth table.

Submit your modified 7SegmentDecoder module here.

| `timescale 1ns / 1ps  *//////////////////////////////////////////////////////////////////////////////////*  *// Create Date: 12/23/2024 04:17:24 AM*  *// Design Name: Exercise1*  *// Module Name: SevenSegmentDecoder*  *// Project Name: Exercise1*  *// Target Devices: Basys3*  *// Tool Versions: 2023.2*  *// Description: Decoder for 7-segment display*  *//////////////////////////////////////////////////////////////////////////////////*  module SevenSegmentDecoder (  input wire [3:0] DataIn, *// 4-bit input*  output wire [7:0] Segments *// 8-bit output (active low)*  );  *// Declare intermediate reg to hold the value*  reg [7:0] Segments\_reg;  *// Continuous assignment from reg to wire*  assign Segments = Segments\_reg;  *// Combinational logic for 7-segment decoding*  always @(\*) begin  case (DataIn)  4'b0000: Segments\_reg = 8'b00000011; *// Display '0'*  4'b0001: Segments\_reg = 8'b10011111; *// Display '1'*  4'b0010: Segments\_reg = 8'b00100101; *// Display '2'*  4'b0011: Segments\_reg = 8'b00001101; *// Display '3'*  4'b0100: Segments\_reg = 8'b10011001; *// Display '4'*  4'b0101: Segments\_reg = 8'b01001001; *// Display '5'*  4'b0110: Segments\_reg = 8'b01000001; *// Display '6'*  4'b0111: Segments\_reg = 8'b00011111; *// Display '7'*  4'b1000: Segments\_reg = 8'b00000001; *// Display '8'*  4'b1001: Segments\_reg = 8'b00001001; *// Display '9'*  4'b1010: Segments\_reg = 8'b00010001; *// Display 'A'*  4'b1011: Segments\_reg = 8'b11000001; *// Display 'b'*  4'b1100: Segments\_reg = 8'b01100011; *// Display 'C'*  4'b1101: Segments\_reg = 8'b10000101; *// Display 'd'*  4'b1110: Segments\_reg = 8'b01100001; *// Display 'E'*  4'b1111: Segments\_reg = 8'b01110001; *// Display 'F'*  default: Segments\_reg = 8'b11111111; *// Default: All segments OFF*  endcase  end  *// cocotb dump waveforms*  `ifdef COCOTB\_SIM  initial begin  $dumpfile("waveform.vcd"); *// Name of the dump file*  $dumpvars(0, SevenSegmentDecoder); *// Dump all variables for the top module*  end  `endif  endmodule |
| --- |

5. Run the testbench and report the result here.

Insert your Xilinx testbench (SevenSegmentDecoderTB) result here.

|  |
| --- |

Insert your Cocotb testbench (from folder /cocotb/SevenSegmentDecoderTB/) result here.

|  |
| --- |

Insert your testbench result waveform here. (Either Cocotb or Xilinx Testbench)

|  |
| --- |

6. Call TA to inspect your work.

## Part 3 : System Integration

Now that you have completed all the submodules, it's time to integrate them into the **System** module, which will serve as the top module.

### Instruction

1.Modify the System module to connect all the submodules according to figure 3.

Submit your Modify System Module here

| `timescale 1ns / 1ps  *//////////////////////////////////////////////////////////////////////////////////*  *// Create Date: 12/23/2024 05:07:36 AM*  *// Design Name: Exercise1*  *// Module Name: SystemTB*  *// Project Name: Exercise1*  *// Target Devices: Basys3*  *// Tool Versions: 2023.2*  *// Description: Testbench for the System module*  *//////////////////////////////////////////////////////////////////////////////////*  module SystemTB ();  *// declare the reg/wire*  reg [7:0] SW;  reg Reset;  reg Clk;  wire [7:0] Segments;  wire [3:0] AN;  reg [7:0] segment\_values[15:0];  *// instantiate the SevenSegmentDecoder module*  System SystemInst (  .SW(SW),  .Reset(Reset),  .Clk(Clk),  .Segments(Segments),  .AN(AN)  );  *// instantiate variable*  integer flag = 0;  integer TestCaseNo = 0;  integer sw;  *// task to check the output*  task check\_output;  input integer TestCaseNo;  input reg [7:0] expected\_Segments; *// Expected output*  input reg [3:0] expected\_AN; *// Expected output*  begin  if (Segments !== expected\_Segments | AN !== expected\_AN) begin  $error(  "ERROR: TestCaseNo %0d | Time = %0t | SW = %b, Reset = %b | Segments = %b (Expected: %b) | AN = %b (Expected: %b)",  TestCaseNo, $time, SW, Reset, Segments, expected\_Segments, AN, expected\_AN);  flag = 1;  end  end  endtask  localparam CLK\_PERIOD = 2;  always #(CLK\_PERIOD / 2.0) Clk = ~Clk;  *// test cases*  initial begin  segment\_values[4'b0000] = 8'b00000011;  segment\_values[4'b0001] = 8'b10011111;  segment\_values[4'b0010] = 8'b00100101;  segment\_values[4'b0011] = 8'b00001101;  segment\_values[4'b0100] = 8'b10011001;  segment\_values[4'b0101] = 8'b01001001;  segment\_values[4'b0110] = 8'b01000001;  segment\_values[4'b0111] = 8'b00011111;  segment\_values[4'b1000] = 8'b00000001;  segment\_values[4'b1001] = 8'b00001001;  segment\_values[4'b1010] = 8'b00010001;  segment\_values[4'b1011] = 8'b11000001;  segment\_values[4'b1100] = 8'b01100011;  segment\_values[4'b1101] = 8'b10000101;  segment\_values[4'b1110] = 8'b01100001;  segment\_values[4'b1111] = 8'b01110001;  Clk = 0;  Reset = 1;  SW = 8'b0;  #(CLK\_PERIOD) Reset = 0;  for (sw = 0; sw < 16; sw = sw + 1) begin  SW = {4'b0, sw};  #CLK\_PERIOD;  check\_output(TestCaseNo, segment\_values[sw], 4'b1110);  TestCaseNo = TestCaseNo + 1;  end  #(CLK\_PERIOD\*150000);  for (sw = 0; sw < 16; sw = sw + 1) begin  SW = {sw, 4'b0};  #CLK\_PERIOD;  check\_output(TestCaseNo, segment\_values[sw], 4'b1101);  TestCaseNo = TestCaseNo + 1;  end  if (flag == 0) begin  $display("All test cases pass");  end else begin  $display("Some test cases fail");  end  $finish;  end  endmodule |
| --- |

2. Run the testbench to verify your module (either Xilinx or Cocotb one).

Insert your testbench (SystemTB) result here.

|  |
| --- |

3. Call TA to inspect your work.

## Part 4 : Programming the Hardware

Now that the entire system is complete and functions correctly according to the testbench, it is time to generate the bitstream and program it onto the **Basys3** board. To do this, we first need to create a **constraint file** to map the system's inputs and outputs to the appropriate pins on the **Basys3** board. Afterward, we will run synthesis, implementation, and generate the bitstream.

### Instruction

1. Complete this mapping table below to map your system's inputs and outputs onto the **Basys3** pins. This system uses **BTNU** for resetting purposes, and **AN[x]** is used to enable the corresponding digit of the 7-segment display.

Hint, look up the Basys3 reference manual page 6 and 15.

| **System.v input/output** | **Basys3 Pins** |
| --- | --- |
| Reset | T18 |
| Clk | W5 |
| SW[0] | V17 |
| SW[1] | V16 |
| SW[2] | W16 |
| SW[3] | W17 |
| SW[4] | W15 |
| SW[5] | V15 |
| SW[6] | W14 |
| SW[7] | W13 |
| AN[0] | U2 |
| AN[1] | U4 |
| AN[2] | V4 |
| AN[3] | W4 |
| Segments[0] | V7 |
| Segments[1] | U7 |
| Segments[2] | V5 |
| Segments[3] | U5 |
| Segments[4] | V8 |
| Segments[5] | U8 |
| Segments[6] | W6 |
| Segments[7] | W7 |

1. Create the constraint file. Follow the instructions in the Lab1 Guide.

Submit your constraint file here.

| set\_property PACKAGE\_PIN W4 [get\_ports {AN[3]}]  set\_property PACKAGE\_PIN V4 [get\_ports {AN[2]}]  set\_property PACKAGE\_PIN U4 [get\_ports {AN[1]}]  set\_property PACKAGE\_PIN U2 [get\_ports {AN[0]}]  set\_property PACKAGE\_PIN W7 [get\_ports {Segments[7]}]  set\_property PACKAGE\_PIN W6 [get\_ports {Segments[6]}]  set\_property PACKAGE\_PIN U8 [get\_ports {Segments[5]}]  set\_property PACKAGE\_PIN V8 [get\_ports {Segments[4]}]  set\_property PACKAGE\_PIN V5 [get\_ports {Segments[2]}]  set\_property PACKAGE\_PIN U5 [get\_ports {Segments[3]}]  set\_property PACKAGE\_PIN U7 [get\_ports {Segments[1]}]  set\_property PACKAGE\_PIN V7 [get\_ports {Segments[0]}]  create\_clock -period 10.000 -name Clk -waveform {0.000 5.000} -add [get\_ports Clk]  set\_property PACKAGE\_PIN W13 [get\_ports {SW[7]}]  set\_property PACKAGE\_PIN W14 [get\_ports {SW[6]}]  set\_property PACKAGE\_PIN V15 [get\_ports {SW[5]}]  set\_property PACKAGE\_PIN W15 [get\_ports {SW[4]}]  set\_property PACKAGE\_PIN W17 [get\_ports {SW[3]}]  set\_property PACKAGE\_PIN W16 [get\_ports {SW[2]}]  set\_property PACKAGE\_PIN V16 [get\_ports {SW[1]}]  set\_property PACKAGE\_PIN V17 [get\_ports {SW[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[7]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[7]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {Segments[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {AN[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {AN[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {AN[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {AN[0]}]  set\_property PACKAGE\_PIN W5 [get\_ports Clk]  set\_property PACKAGE\_PIN T18 [get\_ports Reset]  set\_property IOSTANDARD LVCMOS33 [get\_ports Clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports Reset] |
| --- |

1. Run the synthesis and then open the Schematic.

Explain what you observed in the schematic and submit a picture of the schematic here.

| This schematic represents a 4-digit 7-segment display controller implemented on an FPGA. Inputs include an 8-bit switch (SW[7:0]), a clock (Clk), and a reset (Reset), all buffered with IBUF. The central Controller module processes these inputs to generate Segments[7:0] (to control the segments of the active digit) and AN[3:0] (to select which digit is active). Outputs are buffered with OBUF to drive the physical display. The design likely multiplexes the 4 digits using the clock for sequential activation, enabling the display of user-provided values via switches. |
| --- |

1. Run the implementation, generate the bitstream, and program the **Basys3** device.

Take a picture of your Board working after programming.

|  |
| --- |

1. Call TA to inspect your work.
2. Submit this sheet to the MCV.