THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 7
7-Segment Displays
Nexys A7 board
(Revisions by Roy Kravitz)

1. INTRODUCTION

This lab describes how the RVfpga System was extended to work with 7-segment displays and shows how to modify the 7-segment display controller. The Nexys A7 board has eight 7-segment displays. We first describe how they work (Section 2) and then analyse the high-level specification of the 8-digit 7-segment display controller included in the RVfpga System and provide some fundamental exercises (Sections 3 and 4). Finally, we analyse the low-level implementation of this controller, perform a Verilator simulation and provide additional exercises where you will modify and experiment with the controller implementation (Sections 5 and 6).

2. 7-SEGMENT DISPLAYS ON THE BOOLEAN BOARD

The Nexys A7 board includes two 4-digit seven-segment displays (8 total digits) that use a common anode configuration (see Figure 1). A lot of information can be found at https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

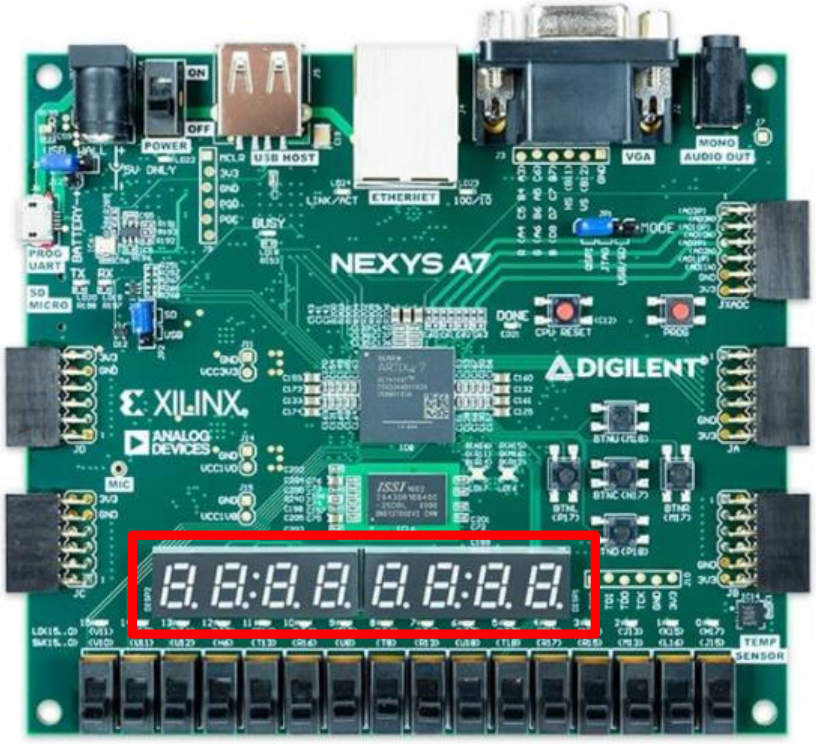

Figure 1. 8-digit 7-segment displays on the Nexys A7 board

A seven-segment display is built from individual LED's arranged in a figure-8 pattern as shown. Any LED/segment can be individually illuminated, so any one of 128 different patterns can be shown; specifically, among these 128 patterns, the decimal digits can be displayed as shown in Figure 2.

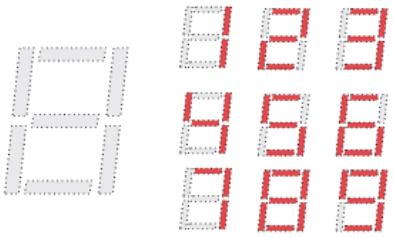

Figure 2. Patterns corresponding to decimal digits

The anodes of the seven LEDs for a single digit are tied together into one “common anode” circuit node, but the LED cathodes remain separate (see Figure 3). The four common anode signals, one for each digit, act as a “digit enable”. The cathodes of the same segment on all four digits are connected into seven signals. (Note that an eighth signal exists for the decimal point, *DP*, but we will not use it in this lab.) This signal connection scheme creates a multiplexed display, where the cathode signals are common to all digits, but they can only illuminate the segments of the digit whose corresponding anode signal is asserted. All these signals are driven low when active.

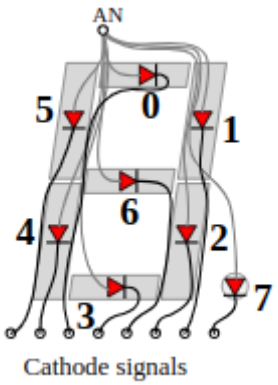

Figure 3. Connection of one 7-segment Display Digit on the Boolean board

A scanning display controller circuit can be used to show an 8-digit number on the 8-digit 7-segment displays. This circuit drives the cathodes with the pattern of each digit in a repeating continuous succession at an update rate that is faster than the human eye can detect; at the same time the circuit drives the anodes one at a time. Thus, each digit is illuminated just one-eighth of the time, but, because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears to be continuously illuminated.

For each of the 8 digits to appear bright and continuously illuminated, all eight digits should be driven once every 1 to 16 ms, and each digit would be illuminated for 1/8 of the refresh cycle (e.g., for a 16ms refresh cycle, each digit is illuminated for 2ms). As explained above, the controller must drive the cathodes of a digit low with the correct pattern while the corresponding anode signal is also driven low.

3. HIGH-LEVEL SPECIFICATION OF THE 8-DIGIT 7-SEGMENT DISPLAY CONTROLLER

In this section, we first describe and analyse the high-level specification of the 8-digit 7-segment displays controller used in the RVfpga System, and then we provide exercises for using it.

A. High-level specification

The 8-digit 7-segment display controller used in this course has been custom-designed for the RVfpga System. It includes two registers, called *Enables_Reg* and *Digits_Reg*, that are mapped to addresses 0x80001038 and 0x8000103C respectively (note that these addresses are unused addresses within the address range reserved for the System Controller, which you can view at https://github.com/chipsalliance/VeeRwolf).

TASK: Locate the declaration of registers *Enables_Reg* and *Digits_Reg*, as well as the place where they are assigned a value. The 8-digit 7-segment displays is implemented in file: *[RVfpgaNexysA7-DDRPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv*

Enables_Reg is an 8-bit register where each bit determines if the corresponding digit is *ON* (0) or *OFF* (1). *Digits_Reg* is a 32-bit register where each 4-bit group represents the hexadecimal value to show in the corresponding digit. For example, to show *71* on the two right-most digits, the programmer would assign the following values to the registers:

- *Enables_Reg = 0xFC* (two right-most digits enabled)
- *Digits_Reg = 0x00000071* (value = 71)

4. FUNDAMENTAL EXERCISES

Exercise 1. Write a RISC-V assembly program and a C program that shows the value of the switches on the four right-most digits of the 7-segment displays.

Exercise 2. Write a RISC-V assembly program and a C program that shows the string “0-1-2-3-4-5-6-7-8” moving from the right to the left of the 8-digit 7-segment displays. That is, 0 should show up on the right-most digit first. Then it should move to the left and 1 should show up on the right-most digit, and so on.

5. 8-DIGIT 7-SEGMENT DISPLAY CONTROLLER: LOW-LEVEL IMPLEMENTATION

Up until this point, we have shown how to use the 8-digit 7-segment displays only. In this section, we describe their low-level implementation and then we provide exercises for modifying the 8-digit 7-segment display controller.

Like previous general-purpose I/O (GPIO) labs, we divide the analysis of the 8-digit 7-segment display controller into three phases:

1. Connection between the SoC and the I/O device on the board (left shadowed region in Figure 4);
2. Integration of the new controller, which is included inside the VeeRwolfX System Controller contained in the SoC (middle shadowed region in Figure 4).

3. Connection between the new controller and the VeeR EL2 Core (right shadowed region in Figure 4).

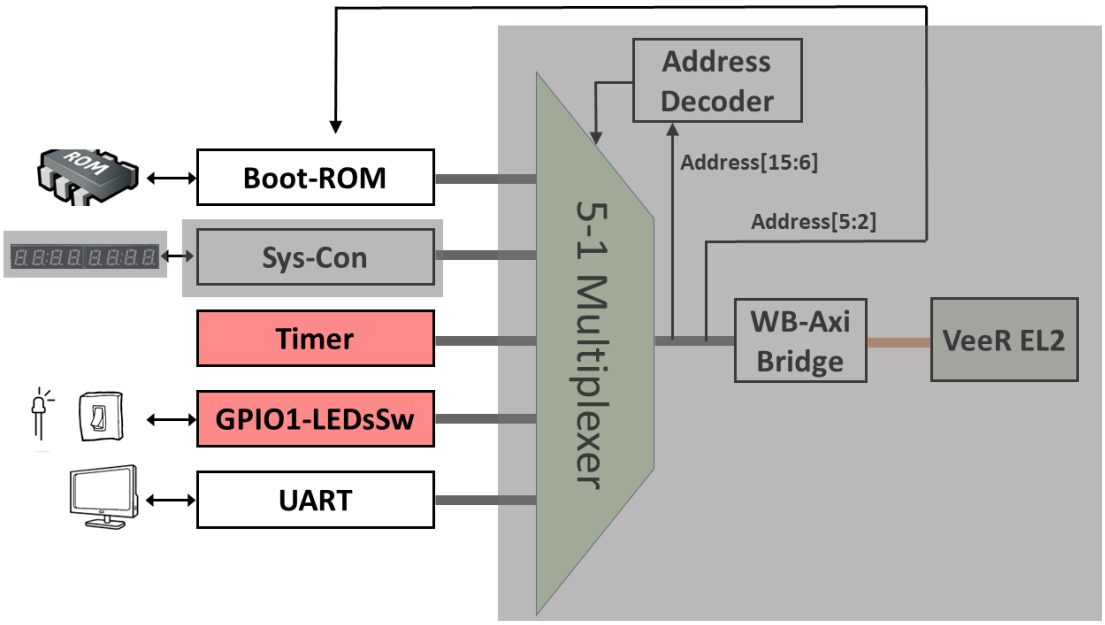

Figure 4. 8-digit 7-segment displays controller analysis in 3 phases

1. Connection of the 7-segment display to the SoC

The constraints file of the project (*[RVfpgaNexysA7-DDRPath]/src/rvfpganexys.xdc*) defines the connection between the input/output SoC signals and the board. Each I/O device on the Nexys A7 board FPGA board is connected to a specific FPGA pin. The signal that connects the eight anodes (see Figure 3) is called *AN[i]* (with *i* ranging from 0-7), and the signals that connect the cathodes of similar segments on all 4 digits (see Figure 3) are called *CA, CB, CC, CD, CE, CF* and *CG*. Figure 5 shows the snippet of the constraints file where these connections are defined.

```
##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { CA }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { CB }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { CC }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { CD }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { CE }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { CF }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { CG }]; #IO_L4P_T0_D04_14 Sch=cg
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

Figure 5. Connection of the two 4-digit 7-segment displays inputs (file *rvfpganexys.xdc*).

In the top-module of our system (module **rvfpganexys.sv**, implemented in file *[RVfpgaNexysA7-DDRPath]/src/rvfpganexys.sv*) you can find the 8-digit 7-segment displays input signals connected to the SoC and at the end of that module you can find their connection to the **veerwolf_core** module (see Figure 6).

```
output reg [15:0]  o_led,
output reg [7:0]   AN,
output reg         CA, CB, CC, CD, CE, CF, CG,

  .io_data        ({i_sw[15:0],gpio_out[15:0]}),
  .AN (AN),
  .Digits_Bits ({CA,CB,CC,CD,CE,CF,CG}),
  .o_accel_sclk   (accel_sclk),
```

Figure 6. Connection of the two 4-digit 7-segment displays to the SoC (file: *rvfpganexys.sv*).

Finally, the two signals are inserted from the **veerwolf_core** module into the System Controller module (**veerwolf_syscon**) (see Figure 7), where the 8-digit 7-segment display controller is implemented.

```
veerwolf_syscon
  #(.clk_freq_hz (clk_freq_hz))
syscon
  (.i_clk            (clk),
   .i_rst            (wb_rst),

   .gpio_irq         (gpio_irq),
   .ptc_irq          (ptc_irq),
   .o_timer_irq      (timer_irq),
   .o_sw_irq3        (sw_irq3),
   .o_sw_irq4        (sw_irq4),
   .i_ram_init_done  (i_ram_init_done),
   .i_ram_init_error (i_ram_init_error),
   .o_nmi_vec        (nmi_vec),
   .o_nmi_int        (nmi_int),

   .i_wb_adr         (wb_m2s_sys_adr[5:0]),
   .i_wb_dat         (wb_m2s_sys_dat),
   .i_wb_sel         (wb_m2s_sys_sel),
   .i_wb_we          (wb_m2s_sys_we),
   .i_wb_cyc         (wb_m2s_sys_cyc),
   .i_wb_stb         (wb_m2s_sys_stb),
   .o_wb_rdt         (wb_s2m_sys_dat),
   .o_wb_ack         (wb_s2m_sys_ack),
   .AN (AN),
   .Digits_Bits (Digits_Bits));
```

Figure 7. Connection of the 8-digit 7-segment displays to the System Controller (file: *veerwolf_core.v*).

TASK: Follow these signals (*CA-CG, CA_1-CG_1* and *AN*) from the constraints file to the System Controller module (where *CA-CG* are merged into array *Digits_Bits*). You will need to inspect the following files: *[RVfpgaNexysA7-DDRPath]/src/rvfpganexys.xdc* *[RVfpgaNexysA7-DDRPath]/]/src/rvfpganexys.sv* *[RVfpgaNexysA7-DDRPath]/src/VeeRwolf/veerwolf_core.v*
[RVfpgaNexysA7-DDRPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv

2. Integration of the 8-digit 7-segment display controller into the SoC

In module **veerwolf_syscon** *([RVfpgaNexysA7-DDRPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv*) the 8-digit 7-segment display controller is instantiated and integrated in the SoC (see Figure 8).

```
module SevSegDisplays_Controller(
                input wire          clk,
                input wire          rst_n,
                input wire    [ 7:0] Enables_Reg,
                input wire    [31:0] Digits_Reg,
                output wire   [ 7:0] AN,
                output wire   [ 6:0] Digits_Bits);
```

Figure 8. 8-digit 7-segment displays controller instantiation (file: *veerwolf_syscon.sv*).

The **SevSegdisplays_Controller** module receives, in addition to the clock signal (*i_clk*, renamed as *clk*) and the reset signal (*i_rst*, renamed as *rst_n*), two input signals (*Enables_Reg* and *Digits_Reg*), which are the two memory-mapped control registers already described. This module outputs two signals, *AN* and *Digits_Bits*, which are connected to the 7-segment displays on the board. For the example showing *71* on the two right-most digits, the **SevSegdisplays_Controller** would assign the following values to signals *AN* and *Digits_Bits*:

- From 0 to 2ms: Signal *AN[0]* is low to enable digit 0 (the right-most digit) to display. Signals *Digits_Bits[5]* and *Digits_Bits[4]* (that correspond to *CB* and *CC*) are also low to display "1" on digit 0 (the right-most digit). All other signals are high.
- From 2 to 4ms: Signal *AN[1]* is low to enable digit 1 to display. *Digits_Bits[6], Digits_Bits[5]* and *Digits_Bits[4]* (that correspond to *CA, CB,* and *CC*) are high to display "*7"* on digit 1. All other signals are high.
- From 4 to 16ms: *AN[2]…AN[7]* are high in 2 ms intervals so that they do not display values. The segments are also high for the remaining digits, digits 2-7.

The **SevSegdisplays_Controller** module is implemented in file *[RVfpgaNexysA7-DDRPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv*. It contains the following subunits:

- Two multiplexers select the value to send to the *AN* and *Digits_Bits* signals every 2ms. The multiplexer is implemented inside module **SevSegMux**.
- For creating the 2ms period, we use a **counter** module provided in files *counter.sv* and *delta_counter.sv*, both included in folder *[RVfpgaNexysA7-DDRPath]/src/OtherSources/pulp-platform.org__common_cells_1.20.0/src*. The counter is configured to count from 0 to 2^{19}, and the 3 most significant bits, which change approximately every 2ms, are used as the select signals for the two multiplexers described above.
- A decoder is implemented in module **SevenSegDecoder**, which outputs the segment values for a given 4-bit hexadecimal value.

TASKS: Analyse the **SevSegdisplays_Controller** module in detail. The simulation performed in the next section can help you on this task. You can also extend the simulation with new signals if necessary.

3. Connection between the 8-digit 7-segment displays controller and the VeeR EL2 Core

As described in Lab 6, the device controllers are connected to the VeeR EL2 Core using a multiplexer (see Figure 4). Remember that the 5:1 multiplexer (Figure 9) is instantiated in file *[RVfpgaNexysA7-DDRPath]/src/VeeRwolf/Interconnect/WishboneInterconnect/wb_intercon.v*. Then, the **wb_intercon** module is instantiated in file *[RVfpgaNexysA7-DDRPath]/src/VeeRwolf/Interconnect/WishboneInterconnect/wb_intercon.vh*. This latter file is included in the **veerwolf_core** module located here: *[RVfpgaNexysA7-DDRPath]src/VeeRwolf/veerwolf_core.v*.

The multiplexer selects which peripheral to read or write, connecting the CPU (*wb_io_** signals) with the Wishbone Bus of one peripheral, depending on the address. For example, if the address generated by the CPU is in the range 0x80001000-0x8000103F, the System Controller is selected, and thus signals *wb_io_** will be connected with signals *wb_sys_**.

```
wb_mux
  #(.num_slaves (5),
    .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001200, 32'h00001400, 32'h00002000}),
    .MATCH_MASK ({32'hfffff000, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hfffff000}))
wb_mux_io
  (.wb_clk_i  (wb_clk_i),
    .wb_rst_i  (wb_rst_i),
    .wbm_adr_i (wb_io_adr_i),
    .wbm_dat_i (wb_io_dat_i),
    .wbm_sel_i (wb_io_sel_i),
    .wbm_we_i  (wb_io_we_i),
    .wbm_cyc_i (wb_io_cyc_i),
    .wbm_stb_i (wb_io_stb_i),
    .wbm_cti_i (wb_io_cti_i),
    .wbm_bte_i (wb_io_bte_i),
    .wbm_dat_o (wb_io_dat_o),
    .wbm_ack_o (wb_io_ack_o),
    .wbm_err_o (wb_io_err_o),
    .wbm_rty_o (wb_io_rty_o),
    .wbs_adr_o ({wb_rom_adr_o, wb_sys_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o}),
    .wbs_dat_o ({wb_rom_dat_o, wb_sys_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o}),
    .wbs_sel_o ({wb_rom_sel_o, wb_sys_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o}),
    .wbs_we_o  ({wb_rom_we_o,  wb_sys_we_o,  wb_ptc_we_o,  wb_gpio_we_o,  wb_uart_we_o}),
    .wbs_cyc_o ({wb_rom_cyc_o, wb_sys_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o}),
    .wbs_stb_o ({wb_rom_stb_o, wb_sys_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o}),
    .wbs_cti_o ({wb_rom_cti_o, wb_sys_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o}),
    .wbs_bte_o ({wb_rom_bte_o, wb_sys_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o}),
    .wbs_dat_i ({wb_rom_dat_i, wb_sys_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i}),
    .wbs_ack_i ({wb_rom_ack_i, wb_sys_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i}),
    .wbs_err_i ({wb_rom_err_i, wb_sys_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i}),
    .wbs_rty_i ({wb_rom_rty_i, wb_sys_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i}));
```

Figure 9. 7-1 multiplexer that selects the peripheral connected with the CPU (file: *wb_intercon.v*).

TASK: Inspect module **veerwolf_syscon** in order to understand how addresses are mapped in the System Controller. Focus on registers *Enables_Reg* and *Digits_Reg* (as we mentioned before, the addresses assigned to these two registers are 0x80001038 and 0x8000103C respectively).

```
14 : begin
     if (i_wb_sel[0]) Enables_Reg[7:0]  <= i_wb_dat[7:0];
end
15 : begin
     if (i_wb_sel[0]) Digits_Reg[7:0]   <= i_wb_dat[7:0];
     if (i_wb_sel[1]) Digits_Reg[15:8]  <= i_wb_dat[15:8];
     if (i_wb_sel[2]) Digits_Reg[23:16] <= i_wb_dat[23:16];
     if (i_wb_sel[3]) Digits_Reg[31:24] <= i_wb_dat[31:24];
end
```

Figure 10. Connection between the 8-digit 7-segment displays and the Core (file *veerwolf_syscon.sv*).

6. ADVANCED EXERCISES

Exercise 3. Modify the controller described in this lab so that the 8-digit 7-segment displays can show any combination of ON/OFF LEDs.

- You do not need an enable register now. Instead, you need eight 7-bit registers. Call them: Segments_Digit0 – Segments_Digit7, one for each of the eight 7-segment displays. In each of these registers, each bit indicates if the corresponding segment is ON (0) or OFF (1). For example, if all the bits of the first register (Segments_Digit0) are 0, all segments in the right-most digit will be ON, whereas if all the bits of the first register are 1, all segments of the right-most digit will be OFF.
- You can map these two new registers to the same addresses that we used before (first remove the two previous registers *Enables_Reg* and *Digits_Reg*):
  - Segments_Digit0 ←→ Address 0x80001038
  - Segments_Digit1 ←→ Address 0x80001039
  - …
  - Segments_Digit7 ←→ Address 0x8000103F
- Note that you do not need the 4-7 decoder anymore (module **SevenSegDecoder**), as the information provided by the program is already decoded.

Exercise 4. Use the new controller for printing the following on the 8-digit 7-segment displays: “I SAY HI”. As usual, implement both RISC-V assembly and C versions of the program.