THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 7
7-Segment Displays

1. Introduction

This lab describes how the RVfpgaEL2 System was extended to work with 7-segment displays and shows how to modify the 7-segment display controller. The Basys 3 Board has four 7-segment displays. We first describe how they work (Section 2) and then analyse the high-level specification of the 4-digit 7-segment display controller included in the RVfpgaEL2 System and provide some fundamental exercises (Sections 3 and 4). Finally, we analyse the low-level implementation of this controller, and provide additional exercises where you will modify and experiment with the controller implementation (Sections 5 and 6).

2. 7-Segment Displays on the Basys 3 Board

The Basys 3 board contains 4-digit common-anode 7-segment LED displays (see Figure 1). Each of the four digits is composed of seven segments arranged in a “figure 8” pattern (see Figure 2), with an LED for each segment. Each of these segments can be switched on or off, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark; specifically, among these 128 patterns, the decimal digits can be displayed as shown in Figure 2.

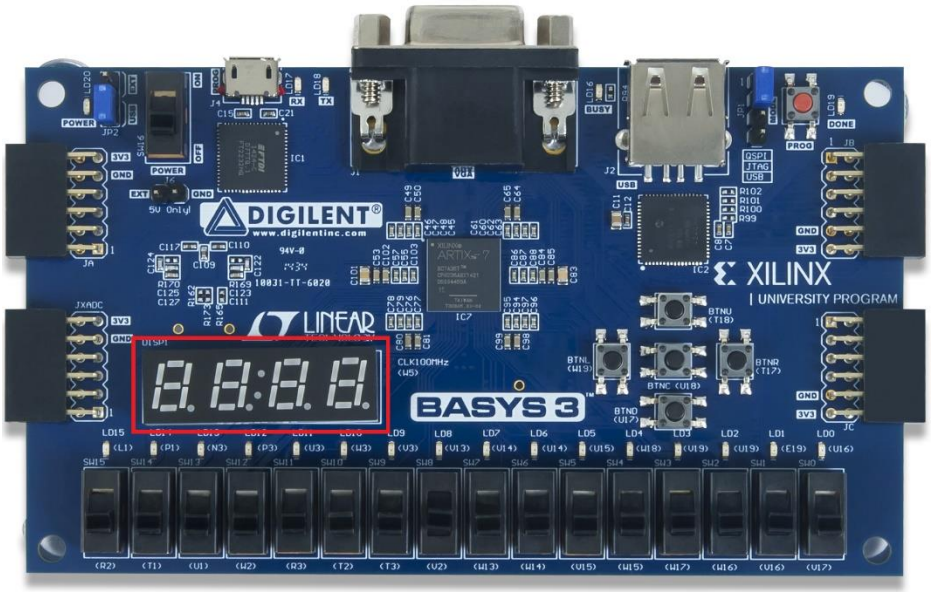

Figure 1. 4-digit 7-segment displays on the Basys 3

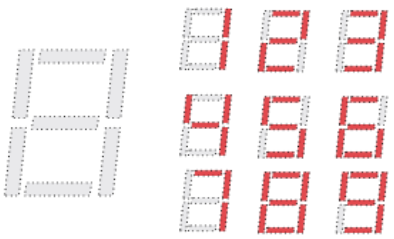

Figure 2. Patterns corresponding to decimal digits

The LED segments of a single digit are labelled *A*-*G*, as shown on the right of Figure 3. 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 (*AN0*-*AN3*), act as a “digit enable”. The cathodes of the same segment on all four digits are connected into seven signals, *CA*-*CG* (see Figure 3). (Note that an eighth signal exists for the decimal point, *DP*, but we will not use it in this lab.) For example, the cathode of segment *D* from the four digits are grouped together into a single circuit node called *CD*. 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; thus, to illuminate a segment, for example, segment *D* on digit *2*, both the anode *AN2* and the cathode *CD* must be driven low.

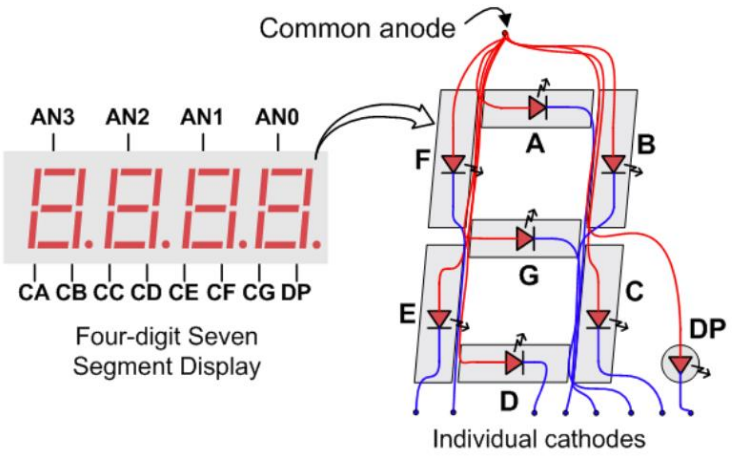

Figure 3. Connection of the 4-digit 7-segment Display on the Basys 3

A scanning display controller circuit can be used to show a four-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession at an update rate that is faster than the human eye can detect. Each digit is illuminated just one-fourth of the time, but because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears continuously illuminated. If the update, or “refresh”, rate is slowed to around 45 hertz, a flicker can be noticed in the display.

For each of the four digits to appear bright and continuously illuminated, all four digits should be driven once every 1 to 16ms, for a refresh frequency of about 1KHz to 60Hz. For example, in a 62.5Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for 1/4 of the refresh cycle. The controller must drive low the cathodes with the correct pattern when the corresponding anode signal is driven high. To illustrate the process, if AN0 is asserted while CB and CC are asserted, then a “1” will be displayed in digit position 1. Then, if AN1 is asserted while CA, CB, and CC are asserted, a “7” will be displayed in digit position 2. If AN0, CB, and CC are driven for 4ms, and then AN1, CA, CB, and CC are driven for 4ms in an endless succession, the display will show “71” in the first two digits.

3. High-Level Specification of the 4-Digit 7-Segment Display Controller

In this section, we first describe and analyse the high-level specification of the 4-digit 7-segment displays controller used in the RVfpgaEL2 System, and then we provide exercises for using it.

The 4-digit 7-segment display controller used in this course has been custom-designed for the RVfpgaEL2 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 4-digit 7-segment displays is implemented in file: *[RVfpgaBasysPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv*

`Enables_Reg` is an 4-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` = 0xC (two right-most digits enabled)
- `Digits_Reg` = 0x0071 (value = 71)

You can test the program provided at *[RVfpgaBasysPath]/Labs/Lab07/71_7SegDispl_C-Lang*

4. Fundamental Exercises

Exercise 1. Write a RISC-V assembly program and/or a C program that shows the value of the switches on the four right-most digits of the 7-segment displays. Recall that you can run the same program either in hardware on RVfpgaEL2-Basys3 or in simulation on RVfpgaEL2-ViDBo. For example, Figure 4 shows the program running on RVfpgaEL2-ViDBo.

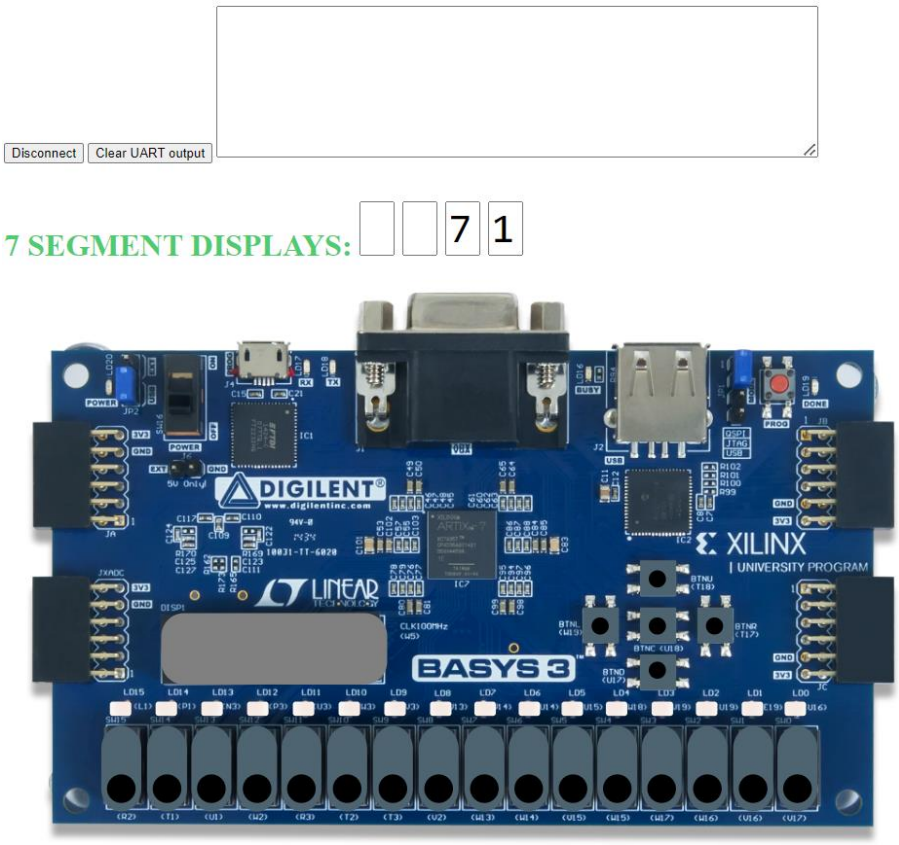

Figure 4. Running Exercise 2 on RVfpgaEL2-ViDBo

Exercise 2. Write a RISC-V assembly program and/or 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 4-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. Recall that you can run the same program on either RVfpgaEL2-Basys3 or RVfpgaEL2-ViDBo.

5. 4-Digit 7-Segment Display Controller: Low-Level Implementation

Up until this point, we have shown how to use the 4-digit 7-segment displays only. In this section, we describe their low-level implementation and then we provide exercises for modifying the 4-digit 7-segment display controller.

Similar to previous general-purpose I/O (GPIO) labs, we divide the analysis of the 4-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 5);
2. Integration of the new controller, which is included inside the VeeRwolfX System Controller contained in the SoC (middle shadowed region in Figure 5);
3. Connection between the new controller and the VeeR EL2 Core (right shadowed region in Figure 5).

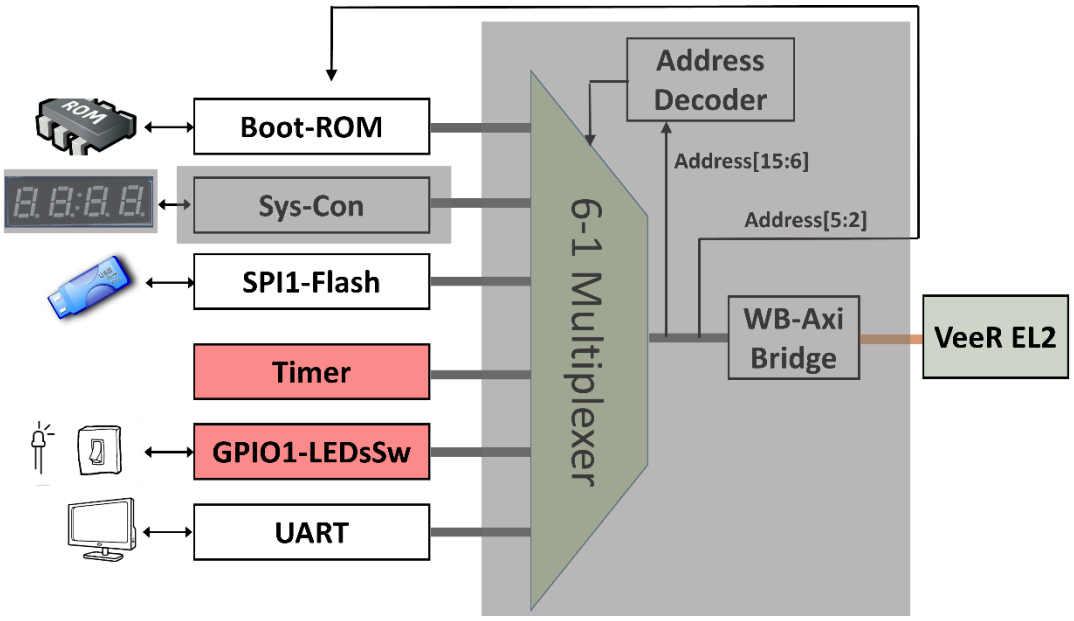

Figure 5. 4-digit 7-segment displays controller analysis in 3 phases

1. Connection of the LEDs/Switches to the SoC

The constraints file of the project (*[RVfpgaBasysPath]/src/rvfpgabasys3.xdc*) defines the connection between the input/output SoC signals and the board. Each I/O device on the Basys 3 Board FPGA board is connected to a specific FPGA pin. The signal that connects the four anodes (see Figure 3) is called *AN[i]* (with *i* ranging from 0-3), 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 6 shows the snippet of the constraints file where these connections are defined.

```
#7 segment display

set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports { CA }]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports { CB }]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports { CC }]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports { CD }]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports { CE }]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports { CF }]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports { CG }]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]
```

Figure 6. Connection of the two 4-digit 7-segment displays inputs (file *rvfpgabasys3.xdc*)

In the top-module of our system (module **`rvfpgabasys3`**, implemented in file *[RVfpgaBasysPath]/src/rvfpgabasys3.sv*) you can find the 4-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 7).

```
`default_nettype none
module rvfpgabasys3
  #(parameter bootrom_file = "boot_main.mem")
   (input wire          clk,
    output wire         o_flash_cs_n,
    output wire         o_flash_mosi,
    input wire          i_flash_miso,
    input wire          i_uart_rx,
    output wire         o_uart_tx,
    inout wire [15:0]   i_sw,
    output reg [15:0]   o_led,
    output reg [3:0]    AN,
    output reg          CA, CB, CC, CD, CE, CF, CG);
```

```
.i_ram_init_error    (1'b0),
.io_data             ({i_sw[15:0],gpio_out[15:0]}),
.AN (AN),
.Digits_Bits ({CA,CB,CC,CD,CE,CF,CG}));
```

Figure 7. Connection of the two 4-digit 7-segment displays to the SoC (file: *rvfpgabasys3.sv*).

Finally, the two signals are inserted from the **`veerwolf_core`** module into the System Controller module (**`veerwolf_syscon`**) (see Figure 8), where the 4-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 8. Connection of the 4-digit 7-segment displays to the System Controller (file: *veerwolf_core.v*).

TASK: Follow these signals (*CA-CG* 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:

[RVfpgaBasysPath]/src/rvfpgabasys3.xdc
[RVfpgaBasysPath]/src/rvfpgabasys3.sv

[RVfpgaBasysPath]/src/VeeRwolf/veerwolf_core.v
[RVfpgaBasysPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv

2. Integration of the 4-digit 7-segment display controller into the SoC

In module **`veerwolf_syscon`** (*[RVfpgaBasysPath]/src/VeeRwolf/Peripherals/SystemController/veerwolf_syscon.sv*) the 4-digit 7-segment display controller is instantiated and integrated in the SoC (see Figure 9).

```
// Eight-Digit 7 Segment Displays

reg  [ 3:0]  Enables_Reg;
reg  [31:0]  Digits_Reg;

SevSegDisplays_Controller SegDispl_Ctr(
  .clk              (i_clk),
  .rst_n            (i_rst),
  .Enables_Reg      (Enables_Reg),
  .Digits_Reg       (Digits_Reg),
  .AN               (AN),
  .Digits_Bits      (Digits_Bits)
);
```

Figure 9. 4-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 4 ms: 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 4 to 8 ms: 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 8 to 16 ms: `AN[2]…AN[3]` are high in 4 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 *[RVfpgaBasysPath]/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 4 ms. The multiplexer is implemented inside module **`SevSegMux`**.
- For creating the 4 ms period, we use a **`counter`** module provided in files *counter.sv* and *delta_counter.sv*, both included in folder *[RVfpgaBasysPath]/src/OtherSources/pulp-platform.org__common_cells_1.20.0/src*. The counter is configured to count from 0 to 2^{17}, and the 2 most significant bits 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 4-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 5). Remember that the 6:1 multiplexer (Figure 10) is instantiated in file *[RVfpgaBasysPath]/src/VeeRwolf/Interconnect/WishboneInterconnect/wb_intercon.v*. Then, the **`wb_intercon`** module is instantiated in file *[RVfpgaBasysPath]/src/VeeRwolf/Interconnect/WishboneInterconnect/wb_intercon.vh*. This latter file is included in the **`veerwolf_core`** module located here: *[RVfpgaBasysPath]/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 (6),
    .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001200, 32'h00001400, 32'h00002000}),
    .MATCH_MASK ({32'hfffff000, 32'hffffffc0, 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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_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_spi_flash_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i}));
```

Figure 10. 6:1 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[3:0]  <= i_wb_dat[3:0];
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 11. Connection between the 4-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 4-digit 7-segment displays can show any combination of ON/OFF LEDs.

- You do not need an enable register now. Instead, you need four 7-bit registers. Call them: `Segments_Digit0` – `Segments_Digit7`, one for each of the four 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. Use the segment numbering shown in Figure 12 (as shown for `Segments_Digit0`, but it will be the same numbering for all digits).

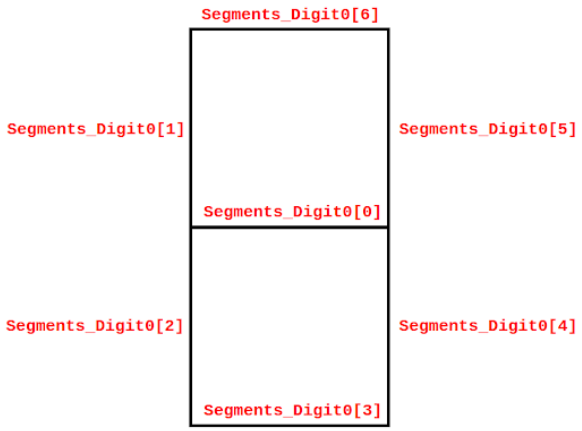

Figure 12. Segment numbering

- 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
  - …

- Note that you do not need the decoder anymore (module **`SevenSegDecoder`**), as the information provided by the program is already decoded.

- The outputs of the controller are the same as before:
  - The 4-bit `AN` output from the controller connects with `AN0` … `AN3`.
  - The 7-bit `Digits_Bits` output from the controller connects with `CA` … `CG` (`DP` is left unconnected on the board).

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

Recall that you can run the same program in either RVfpgaEL2-Basys3 or RVfpgaEL2-ViDBo. For example, Figure 13 shows the program running on the modified RVfpgaEL2-ViDBo developed in Exercise 3.

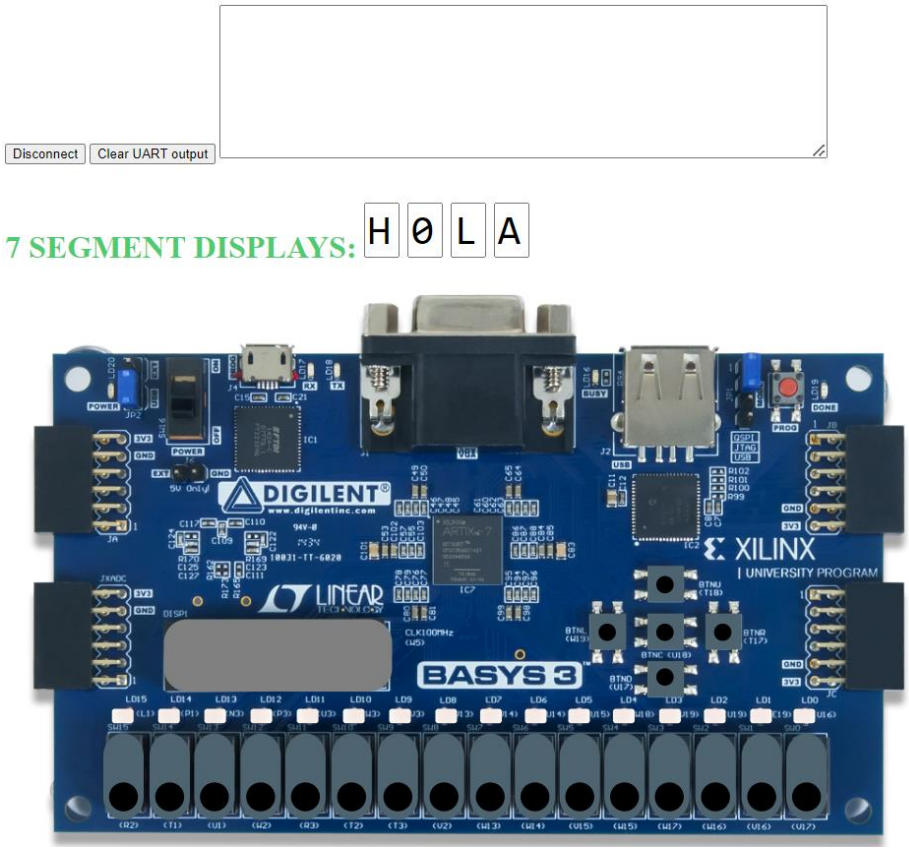

Figure 13. RVfpgaEL2-ViDBo running Exercise 4

IMPORTANT: The virtual board simulates the same behaviour of this device, thus it receives the same inputs from the SoC: signals `AN[0:3]` and `CA-CG`. However, some things must be taken into account when using RVfpgaEL2-ViDBo:

- The physical board supports any combination of LEDs in the 7-segment displays. However, the virtual board only supports the hexadecimal digits plus the following

characters:

Any other 7-segment display combination generated by the controller will show the digit as off.

- The symbols shown for number 0 and letter O, number 1 and letter I, and number 5 and letter S, are the same.