Queen’s University
Department of Electrical and Computer Engineering
ELEC 271 Digital Systems
Fall 2018

Lab 5: Code Conversion for Hex (Seven-Segment) Displays, Decoders, and Multiple Registers

Objectives

This laboratory activity for *ELEC 271 Digital Systems* provides the opportunity for students to:

- experiment with decoder and code-conversion logic,
- understand the operation of circuits containing multi-bit registers,
- learn to use hex (seven-segment) LED displays for output, and
- gain additional experience with clock inputs to logic circuits.

This activity involves the use of the worksheet included in the description. *Each* student must have a paper copy of the worksheet ready at the beginning of the laboratory session, and each student must complete it (with reasonable neatness) during the session, *otherwise no credit will be granted.*

Each student must read the indicated sections of technical documentation for relevant details.

Each student must also prepare VHDL code in advance as specified in the description.

Preparation BEFORE Your Scheduled Laboratory Session

- *You will individually prepare a file on your network storage or USB drive so that you are fully prepared to make effective use of your in-lab time and to address a possible partner absence.*
- In a file entitled `lab5.vhd`, prepare the following template for later modification.

```
library ieee;
use ieee.std_logic_1164.all;

entity lab5 is
  port (
        -- clock and reset connections that will use pushbuttons
        clk, reset_n : in  std_logic;

        -- 3-bit pattern choice input to feed hex (7-segment) code converter
        pattern : in std_logic_vector(2 downto 0);

        -- 2-bit hex display selection input to feed 2-to-4 decoder
        display : in std_logic_vector(1 downto 0);

        -- enable input for 2-to-4 decoder
        enable : in std_logic;

        -- four 7-bit pin outputs, one vector for each hex (7-segment) display
        hex_d0, hex_d1, hex_d2, hex_d3 : out std_logic_vector(6 downto 0)
       );
end entity;


architecture logic of lab5 is

-- register output vectors to hold current pattern for each hex display
signal q_d0, q_d1, q_d2, q_d3 : std_logic_vector(6 downto 0);

-- output vector of code converter to generate hex (7-segment) display pattern
signal code_conv_out : std_logic_vector(6 downto 0);

-- output vector of 2-to-4 decoder to select register that will change
signal load_enables : std_logic_vector(3 downto 0);

begin

-- combined process for behavior of four registers, one for each hex display;
-- load enables are separate, but data input vector is same for all registers
hex_display_registers : process (clk, reset_n)
begin
  if (reset_n = '0') then
    q_d0 <= (others => '0');
    q_d1 <= (others => '0');
    q_d2 <= (others => '0');
    q_d3 <= (others => '0');
  elsif (clk'event and clk = '1') then
    if (load_enables(0) = '1') then q_d0 <= code_conv_out; end if;
    if (load_enables(1) = '1') then q_d1 <= code_conv_out; end if;
    if (load_enables(2) = '1') then q_d2 <= code_conv_out; end if;
    if (load_enables(3) = '1') then q_d3 <= code_conv_out; end if;
  end if;
end process;
```

Type these hex... signal names *exactly as shown here.*

(continued on next page)

The continuation of `lab5.vhd` is shown below. The two pages of code should be in one file.

```
-- hex display output connections (inversion makes '1' mean 'LED is on')
hex_d0 <= not q_d0;
hex_d1 <= not q_d1;
hex_d2 <= not q_d2;
hex_d3 <= not q_d3;

-- gate-level specification of 2-to-4 decoder for load-enable signals
load_enables(0) <= ???????;
load_enables(1) <= ???????;
load_enables(2) <= ???????;
load_enables(3) <= ???????;

-- high-level specification of code converter for hex (7-segment) display
code_conv_out <=   "???????" when (pattern = "???")
                   else .....

end architecture;
```

- Additional preparation for this exercise involves consulting the Terasic DE0 User Manual to learn more about the connections between the Cyclone III FPGA chip on the DE0 board and the various input/output features of the board.

  Specifically for the `lab5.vhd` file and the corresponding circuit whose schematic is shown later in this document, **read Section 4.2 and Section 4.3** in the Terasic DE0 User Manual:

  You must understand the operation of the buttons and seven-segment LED displays that you will be using in this exercise, and you must understand the naming conventions for the four displays and the individual LED segments for each display. **The diagram in Figure 4-10 is important.** When using a logic bit vector of the form `(6 downto 0)` to connect a seven-segment display, the top LED segment labelled ‘0’ in Figure 4-10 would correspond to bit 0 of the vector, segment ‘1’ would correspond to bit 1 of the vector, and so on.

  Also read Section 4.4, which describes the connection of the external 50-MHz clock source.

- **Study the `lab5.vhd` file and the corresponding circuit whose schematic is shown later in this document to understand the internal connections, the use of the inversion at the outputs, the control and data connections for the registers, etc.**

Part 1: Implementation of Circuit for Controlling Seven-Segment Displays

- Create a folder `Z:\ELEC271\LAB5`. (Or use a USB key for faster access without the network.)
- Place your template file `lab5.vhd` in the above folder.
- Start Altera Quartus II, and create a new project called `lab5` in the folder you created above.
- At the appropriate screen of the New Project Wizard, include the `lab5.vhd` file into the project.
- As done in other lab sessions, use the New Project Wizard to complete the setup for the correct Cyclone III chip on the Altera DE0 board. After completing the steps in the New Project Wizard, remember to also use *Assignments → Device…* to change settings for *unused pins* and *voltage*.
- *On your worksheet*, use the space provided to derive the output code vectors for the collection of hex-display patterns in the specifications given to you at the start of your scheduled session.
- In the Quartus II main menubar, select *File → Open...* and select your `lab5.vhd` file for editing.
- Use your worksheet to prepare the signal assignment statement in `lab5.vhd` for the signal named `code_conv_out` using `when…else…` syntax in VHDL.
- Generate a *gate-level* specification in VHDL for the `load_enables` output vector in `lab5.vhd`.
- From the main menubar, select *File → Save* to save the modified file.
- Select *Processing → Start Compilation* from the main menubar in order to synthesize the circuit.
- After successful synthesis without errors, select *Assignments → Pin Planner*.
- Use the Terasic DE0 User Manual to complete the pin assignments as follows.

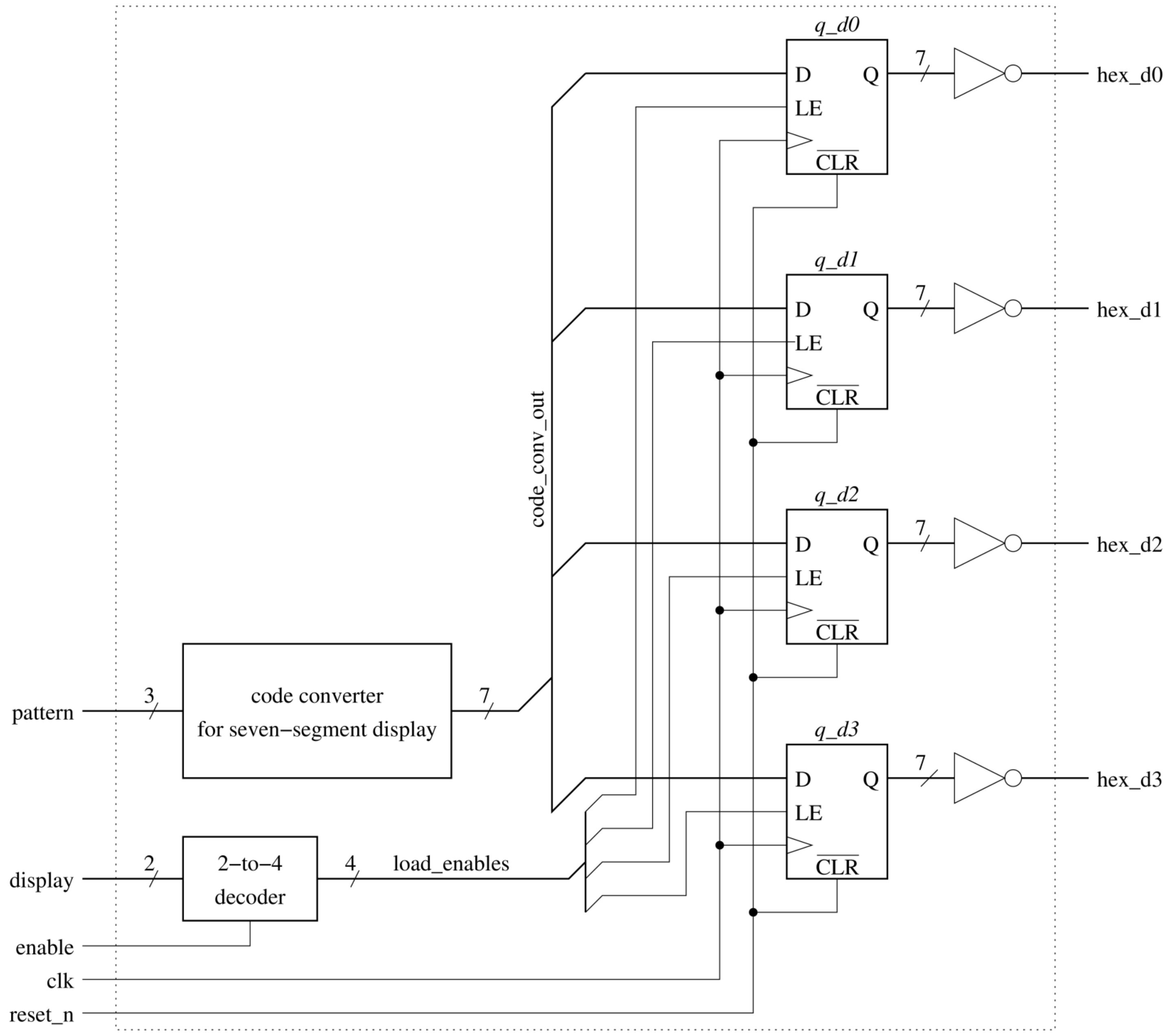
`reset_n`: BUTTON0	`clk`: BUTTON1	
`pattern[2]`: SW9	`pattern[1]`: SW8	`pattern[0]`: SW7
`enable`: SW2	`display[1]`: SW1	`display[0]`: SW0

- *Keep the Pin Planner window open*.
- From the course Website, download the file provided for the hex display pin assignments and save it into your `Z:\ELEC271\LAB5` folder (or the folder on your USB key, if using one).
- In the *Quartus II window*, select *Assignments → Import Assignments* from the main menubar.
- In the Import Assignments dialog box that appears, click on the "…" button and in the file selection dialog box, locate and select the file for the hex display pin assignments, then click on *Open*.
- Check the *System* messages at the bottom of the Quartus II window to confirm that the file was read successfully. Then, check the Pin Planner window for the hex display pin assignments.
- Select *Processing → Start Compilation* **again** to resynthesize with the specified pin assignments.
- With the USB cable connected and the DE0 board on, select *Tools → Programmer.* Ensure that "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- Use the switches for `pattern` and `display` along with the `clk` pushbutton to load the desired values into the registers for the hex displays to show one of the words given in your specifications.
- Experiment with the `enable` input to observe the resulting behavior from pressing the `clk` button.
- Return to your `lab5.vhd` file. Use '`--`' characters to comment out the logic for `load_enables`.
- Below the commented-out statements, introduce a *high-level* `when…else…` statement to perform the decoding with *vectors*, rather than at gate-level . **The `enable` input should be handled first**.
- Save the modified file, recompile the project, reprogram the chip, and test the circuit again to confirm the equivalence between the gate-level and high-level specifications for the decoder.

Part 2: Dynamic Patterns on the Hex Displays using the 50-MHz Clock Input

- Return to your `lab5.vhd` in the Quartus II editor window.
- Use the mouse to select all text from `architecture logic of…` to `end architecture;`
- Copy the selected text, then paste it after `end architecture;` at the end of the file.
- Change the name of the new architecture to `part2` or `logic2` – something different than the first.
- This new architecture is the *last* one in the file, so the VHDL compiler will – by default – use it as the architecture for circuit synthesis. The first architecture will be needed in the *next* lab exercise.
- The circuit for this part will still have four multi-bit registers with clock and reset inputs.
- For the *load-enable signals* in the *NEW* architecture for `lab5.vhd`, however, simply *force all of them to logic-1* so that the registers *always* accept new data on each clock edge.
- In the VHDL process for registers in the *NEW* architecture, change the input data for `q_d`**1** and `q_d`**3** to use a new signal vector `other_code_conv_out`. (Leave `q_d0`, `q_d2` the same.)
- In the list of internal signal definitions for the *NEW* architecture, add `other_code_conv_out` to the existing signal definition for `code_conv_out`.
- *On your worksheet*, use the specifications provided to you for Part 2 to derive the output vectors that must be generated by the logic for `code_conv_out` and `other_code_conv_out`.
- Using the information on your worksheet, modify the *NEW* architecture so that the existing signal assignment statement for `code_conv_out` uses *register contents* to determine the output, rather than the pattern switches. Use the `q_d0` signal as the input in the `when` conditions.
- Introduce a new signal assignment statement for `other_code_conv_out` to similarly reflect the information on your worksheet. Use the `q_d1` signal as the input in the `when` conditions.
- In the *NEW* architecture, change all `(others => '0')` outputs in the reset case for the four-register process. Initial outputs for registers `q_d`**0**, `q_d`**2** should be the `code_conv_out` initial pattern. Registers `q_d`**1**, `q_d`**3** should use the initial pattern for `other_code_conv_out`.
- *Save the modified file* so that the changes you have made up to this point are not lost.
- In your Web browser, open the description for Lab 1 in a new tab for reference.
- At the top of `lab5.vhd`, introduce a `use` statement for the `std_logic_unsigned` library to allow the VHDL compiler to accept specifications with a '+' symbol. See Part 4 of Lab 1.
- Based on Part 7 of Lab 1, add a signal def'n for a 32-bit `count` vector in the *NEW* architecture. Also introduce a new process to increment the value of `count` on each edge of the `clk` input. Finally, modify the *existing* four-register process in the *NEW* architecture to use `count(25)` as the clock, which means modifying the sensitivity list and the `if` condition for the clock event/value.
- *Save the modified file*.
- Use the Terasic DE0 User Manual to identify the pin for CLOCK_50, which will provide a 50-MHz continuous clock signal to the circuit on the chip.
- Open the Pin Planner. Change the pin assignment of the `clk` signal to use the CLOCK_50 pin.
- Select *Processing* → *Start Compilation* to synthesize the new architecture with the 50-MHz clock.
- Program the chip with the new circuit. *As soon as the chip is programmed*, the hex displays should immediately begin cycling through the patterns that were specified for you to implement. If the pattern sequence is incorrect, review your work, make VHDL corrections, and try again.
- Using bit 25 of the `count` vector provides a speed slow enough to aid in your verification of correct behavior. You can try bit 23 or bit 22 by modifying the file in the two appropriate locations, then resynthesizing the circuit and reprogramming the chip. Also try the reset pushbutton.

➡ The schematic diagram below corresponds to the description in the `lab5.vhd` file for Part 1.



Note that the inversion of all lines before the hex-display output pins is needed to make a '1' output from a flip-flop cause a segment to turn on. The manner in which the external connections are made on the DE0 is such that a '0' output ***at the pin*** turns on a segment. Hence, the inversion adapts our circuit to the board.

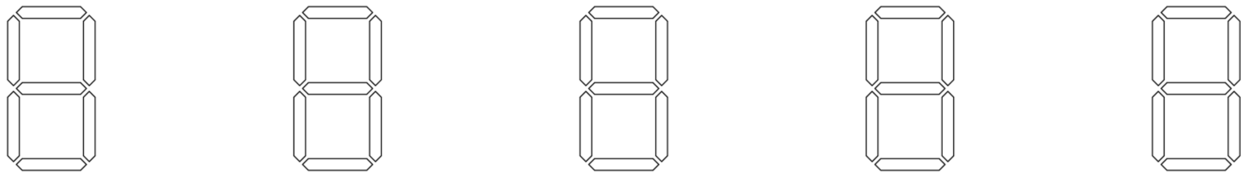
➡ Part 2 will retain the four registers, but the `display`, `enable`, `pattern` inputs will not be used.

Instead, you will introduce additional VHDL code in a *NEW* architecture to internally generate specified patterns for the different hex (seven-segment) displays.

ELEC 271 Lab 5 Worksheet *Name:* ____

<u>Part 1:</u> Prepare the bit vectors for the display patterns in the specifications that were given to you.

	`pattern`	`pattern`	`pattern`	`pattern`	`pattern`
3 bits	______	______	______	______	______



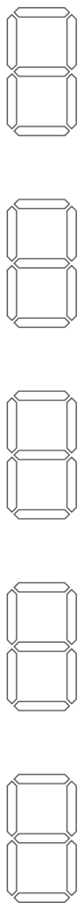
	`code_conv_out`	`code_conv_out`	`code_conv_out`	`code_conv_out`	`code_conv_out`
7 bits	____________	____________	____________	____________	____________

Reproduce the list of sample words from specifications here: ______________________________

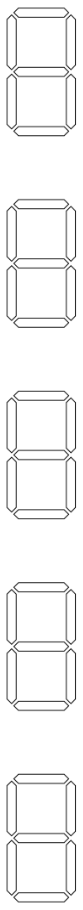
____

<u>Part 2:</u> For the two code converters, prepare the output code vectors from the given specifications.

`code_conv_out`



`other_code_conv_out`



From the above information, write <u>two</u> VHDL signal assignment statements in your *lab5.vhd* file using `when`…`else`… syntax, where the combinational logic output *vectors* depend on register output *vectors*.