

# ECSE-323

# Digital System Design

**Lab #2** – *Combinational Circuit Design with VHDL* Fall 2017

# Introduction

In this lab you will learn how to use the Altera Quartus II FPGA design software to implement combinational logic circuits described in VHDL.

# Learning Outcomes

*After completing this lab you should know how to:*

- Describe a circuit with VHDL, using component statements as well as conditional and selected signal assignment statements
- Include library design entities as components in your VHDL-based designs
- Use ROM modules to implement look-up tables

# Table of Contents

*This lab consists of the following stages:*

1. Design and functional simulation of a random number generator circuit.
2. Design and functional simulation of a ROM based enable signal decoder circuit
3. Design and functional simulation of a ASCII to 7-segment LED decoder circuit
4. Writeup of the lab reports

## **1. Design of a Random Number Generator.**

The lab project this term will be to create a card game, where one person can play against the machine. More details about the nature of the game will be given in future labs.

For now, you will create a circuit that will be the heart of a random number generator. The random numbers will be used to shuffle the deck of 52 cards used in our game.

## 1. Design of a Random Number Generator.

The approach to be used in generating the random numbers is the ***Linear Congruential Generator*** algorithm. This algorithm is based on the following formula:

$$R = \text{mod}(a * SEED + b, c)$$

where  $a$ ,  $b$ , and  $c$  are constants,  $SEED$  is some initial value and  $R$  is the generated random number. To generate more than one number you would replace  $SEED$  with  $R$  and re-compute a new value of  $R$ .

To take a simple example, suppose that  $a=7$ ,  $b=0$ ,  $c=13$ . Take  $SEED=5$ . Then, remembering the definition of the modulo operation from lab 1, the result would be:

$$R = \text{mod}(7 * 5 + 0, 13) = \text{mod}(35, 13) = 9$$

Now, take this value of  $R$  and use it as a new seed, and recompute:

$$R = \text{mod}(7 * 9 + 0, 13) = \text{mod}(63, 13) = 11$$

If we repeat this over and over we get the following sequence of values:

9, 11, 12, 6, 3, 8, 4, 2, 1, 7, 10, 5, 9, 11, 12...

The sequence repeats, with a period of 12 (it includes all numbers between 1 and 12).

The simple system defined in our example is not very useful as a random number generator, since its period is so small. It would be very easy to predict the next number in the sequence.

Thus, we need different values of  $a, b, c$ , which will yield a larger period for our random sequence. There are many possible parameters which will give useful sequences.

One of these parameter sets is  $a=65539, b=0, c=2^{31}$  (2 to the power 31). These parameters were used in the IBM *RANDU* function, as used in IBM computers in the '60s.

(System/360 Scientific Subroutine Package, Version III, Programmer's Manual. IBM, White Plains, New York, 1968, p. 77)



The main advantage of the RANDU version of the linear congruent generator is that it is *very easy to implement*. This is for two main reasons:

- Computing the modulo of a number  $a$  with respect to a number which is a power of 2, e.g.  $\text{mod}(a, 2^N)$ , can be done simply by setting all bits of  $a$  beyond the  $(N-1)$ th bit to zero.
- Multiplication by the number 65539 can be done efficiently using a shift-and-add approach.

Because of these two points, the operation  $\text{mod}(65539 * R, 2^{31})$  can be implemented with just two 32-bit adders (assuming that your value  $R$  has 32 bits) and some shifting of bits (which is just changing which bits are wired to the adder inputs).

Your job is to design the circuit to do this! Once you have it figured out, sketch a block diagram of your approach and show it to the TA, and explain how/why it works.



## VHDL Description of the random number generator circuit.

Once you have your design worked out, write a VHDL description of it.

Begin with an entity declaration having the following form (remember to replace the header with your own information)

```
-- this circuit implements the IBM RANDU version of a linear congruential generator
--
-- entity name: g00_RANDU
--
-- Copyright (C) 2013 Your Names go Here
-- Version 1.0
-- Author: Your Names; Your email addresses
-- Date: Today's Date

library ieee; -- allows use of the std_logic_vector type
use ieee.std_logic_1164.all;

entity g00_RANDU is
  port ( seed          : in std_logic_vector(31 downto 0);
        rand          : out std_logic_vector(31 downto 0) );
end g00_RANDU;
```



## TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *first* 2-hour lab period!

## CREATE THE VHDL DESIGN FILE

Create an empty VHDL file window (using the "New" command from the "File" menu in Quartus).

Save the file under the name *gNN\_RANDU.vhd*. Include it into your project, by checking the "***Add File to Current Project***" box.

First enter the entity declaration shown earlier. Then fill in the file with the VHDL architecture body for your *gNN\_RANDU* circuit.

Rather than making your own 32-bit adder you can use the Adder module from the Altera *lpm* library. To provide access to these modules in your design, you should include the following two lines at the beginning of your design entity:

**LIBRARY** lpm;

**USE** lpm.lpm\_components.all;

If you include these lines, you do not need to have component declarations for the lpm modules that you use in your design.

## CREATE THE VHDL DESIGN FILE (continued)

You should read over the on-line help pages in Quartus related to LPM Megafunctions. Each of the available modules in the library has a help page which describes the modules ports and functioning.

In particular, you should look at the help file for the help topic:

*‘ Example of a VHDL Design File with LPM Function Instantiation ’*

This gives an example of how to include an LPM file into your VHDL description.

**Make sure you have declared all of your internal (non-port) signals that you need to connect your components together!**

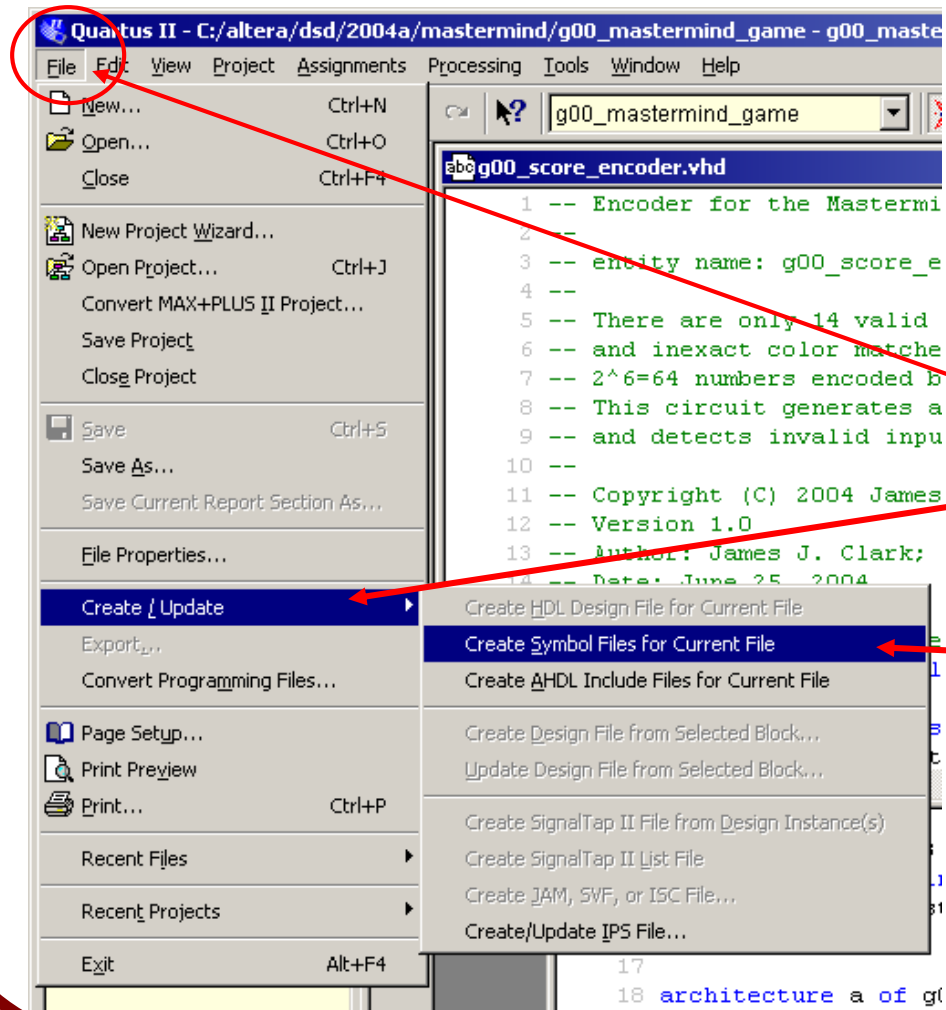
Once you have finished writing your VHDL description, save it and show it to the TA.



## CREATE THE CIRCUIT'S SYMBOL

In order to include your VHDL design in a schematic or in another design you must create a *symbol* for it.

This is done by selecting **Create/Update** from the **File** menu and selecting the **Create Symbol Files for Current File** command



“0000000000000000000000000000000001”

and run the simulation to determine the output. Write down the result and re-do the simulation with this result used as the new seed value.

Repeat for a total of 5 numbers generated.

Show the TA the results of your simulation.



It should be noted that the **RANDU** function is a notoriously bad random number generator. It should not be used for any serious applications. In fact, it can be shown that, if you have the values of two consecutive generated numbers, then the next number to be generated will have one of only 15 possible values! In particular, the relationship between the value of 3 successive values is:

$$\text{mod}(R_n - 6 * R_{n-1} + 9 * R_{n-2}, 2^{31}) = 0$$

You should check to see that this relationship holds with the numbers you obtained from your simulation (you can use Matlab or Wolfram Alpha for this).

But, even though the RANDU function is not very good as random number generators go, we will use it for the course project anyway!





## TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *second* 2-hour lab period!

## 2. Design of The Pop-Enable Circuit.

In this part of the lab you will create a circuit that will be used (in lab #3) as part of a circuit that removes cards from a players hand.

The circuit will take a 6-bit input value **N**, representing a the location of a card in the hand, numbered from 0 through 51 (since in this game a player can have no more than 52 cards in their hand), and will generate a 52-bit output, **P\_EN**.

The output bits should be set such that for a given value of **N**, **P\_EN(0)** through **P\_EN(N-1)** should be all zero and **P\_EN(N)** through **P\_EN(51)** should be all one. For values of **N** greater than 51, all output bits should be zero.

This circuit could be described in VHDL using a single selected signal assignment statement, but will we use a different approach to implementing this function – one using a ***ROM look-up table (LUT)***.

In the LUT approach we use a memory unit that has one entry for every possible input pattern. The input bits are therefore used as an address for the memory and the corresponding output values are stored in the memory.

Many modern FPGA devices, such as the CycloneII device that we will use throughout the lab experiments, contain embedded memory blocks. If these are large enough, they can be used for LUT implementations of boolean functions.

You can use one of the pre-defined Altera **LPM** (**L**ibrary of **P**arametrized **M**odules) modules to implement the LUT. You will write a VHDL description of the circuit, and instantiate the LUT using the *lpm\_rom* component.

To use the *lpm\_rom* module in your design, you must include the following two lines at the beginning of your design entity:

```
LIBRARY lpm;
USE lpm.lpm_components.all;
```

As an *example* of how to specify the generic parameters of a parametrized library module, look at the following *component instantiation statement* for an *lpm\_rom* module:

```
crc_table : lpm_rom -- use the altera rom library macrocell
GENERIC MAP (
    lpm_widthad => 7, -- sets the width of the ROM address bus
    lpm_numwords => 128, -- sets the number of words stored in the ROM
    lpm_outdata => "UNREGISTERED", -- no register on the output
    lpm_address_control => "REGISTERED", -- register on the input
    lpm_file => "crc_rom.mif", -- the ascii file containing the ROM data
    lpm_width => 8) -- the width of the word stored in each ROM location
PORT MAP (clock => clk, address => x, q => crc_of_x);
```

**Note that memory blocks on the Cyclone II chip must have registered inputs. Therefore you need to have a clock input as well.**

Note: For all of the Altera LPM modules, you do not have to include a COMPONENT statement in the architecture declarations area, as this is included when you invoke the lpm library. If you want to include one of your own modules as a component, you need to include the COMPONENT statement to declare it.

You will need to create an *.mif* (Memory Initialization File) file to specify the contents of the LUT. When your VHDL description is compiled by the Altera Quartus software, the .mif file will be read. This should have 64 lines, one for each memory location.

As usual, more information can be obtained from the Quartus help facility, an excerpt of which is shown on the next page.

### Memory Initialization File (taken from the Quartus help)

An ASCII text file (with the extension *.mif*) that specifies the initial content of a memory block (CAM, RAM, or ROM), that is, the initial values for each address. This file is used during project compilation and/or simulation. A MIF is used as an input file for memory initialization in the Compiler and Simulator. You can also use a Hexadecimal (Intel-Format) File (*.hex*) to provide memory initialization data. A MIF contains the initial values for each address in the memory. A separate file is required for each memory block. In a MIF, you are also required to specify the memory depth and width values. In addition, you can specify the radices used to display and interpret addresses and data values.

The following is an example (items between % symbols are comments)

```
DEPTH = 32;           % Memory depth and width are required %
WIDTH = 14;           % Enter a decimal number %

ADDRESS_RADIX = HEX; % Address and value radices are required %
DATA_RADIX = HEX;    % Enter BIN, DEC, HEX, OCT, or UNS; unless %
                    % otherwise specified, radices = HEX %

-- Specify values for addresses, which can be single address or range
CONTENT
    BEGIN
[0..F] : 3FFF;        % Range of addresses--Every address from 0 to F = 3FFF %
6      : F;           % Single address--Address 6 = F %
8      : F E 5;       % Range of three addresses starting from specific address -- %
END ;                % Addr[8] = F, Addr[9] = E, Addr[A] = 5 %
```

If multiple values are specified for the same address, only the last value is used.

Write up a VHDL description of the circuit, using the following entity declaration for your design:

```
entity gNN_pop_enable is
  port ( N           : in std_logic_vector(5 downto 0);
        P_EN        : out std_logic_vector(51 downto 0));
end gNN_pop_enable;
```

## CREATE THE DESIGN FILE AND SYMBOL

Once you have written the VHDL description, analyze it (using the *Processing/Analyze Current File* menu item, to check for errors.

When your design is error-free, create a symbol for it, and insert an instance into the top-level project file.

Show your completed VHDL description to your TA.



In preparation for simulation, compile the design for the pop\_enable circuit using the *Processing/Start Compilation* menu item.



## SIMULATE THE DESIGN

Once compilation is successful do a functional simulation. This simulation should test *all 64 possible* input patterns of the input code.

As before, in order to carry out the simulation, you should insert an instance of the *gNN\_pop\_enable* symbol into your top-level project schematic diagram, and hook up inputs and outputs as needed.

Show the TA the results of your simulation.



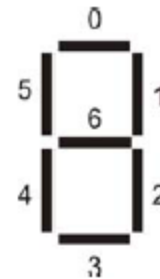


## TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *third* 2-hour lab period!

### 3. Design of the 7-Segment LED decoder/driver .

A 7-segment LED display has 7 individual light-emitting segments, as shown in the picture below. By turning on different segments at any one time we can obtain different characters or numbers. There are four of these attached to the Cyclone FPGA chip on the Altera board (which you will be introduced to in Lab 3), which you will use later in your full implementation of the time-stamp system.



numbering of the LED  
segments (from Altera DE1  
board manual)

In this part of the lab you will design a circuit that will be used to drive the 7-segment LEDs on the Altera board. It takes in a 4-bit code and generates the 7-segment display associated with the input code.





























*The outputs should be made active-low. This is convenient, as many LED displays, including the ones on the Altera board, turn on when their segment inputs are driven low.*

Your circuit should have 2 **modes**, selected by a 1-bit mode input:

**Mode = 0** indicates that the display should interpret the 4-bit input as a hexadecimal number (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

**Mode = 1** indicates that the display should interpret the 4-bit input as a card face value (A,2,3,4,5,6,7,8,9,10,J,Q,K,-,-,-)

Display the symbols on the 7-segment display as shown below:

Code =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Mode = 0																	
Mode = 1																	

Use the following entity declaration for your design:

```
entity gNN_7_segment_decoder is
  port ( code      : in std_logic_vector(3 downto 0);
         mode      : in std_logic;
         segments_out : out std_logic_vector(6 downto 0));
end gNN_7_segment_decoder;
```

To implement the 7-segment LED decoder, use a single selected signal assignment statement (with 32 cases).

To get you started, here are the first few lines of the architecture body. You fill in the rest!

```
SIGNAL xcode : std_logic_vector(4 downto 0);  
BEGIN  
xcode(4 downto 1) <= code; xcode(0) <= mode;  
WITH xcode SELECT  
segments_out <=  
    "1000000" WHEN "00000",  -- code = 0, mode = 0  
    "0000100" WHEN "00001",  -- code = 0, mode = 1  
    "1111100" WHEN "00001",  -- code = 1, mode = 0  
    ...
```

## CREATE THE DESIGN FILE AND SYMBOL

Once you have written the VHDL description, analyze it (using the *Processing/Analyze Current File* menu item, to check for errors.

When your design is error-free, create a symbol for it, and insert an instance into the top-level project file.

Show your completed VHDL description to your TA.



In preparation for simulation, compile the design for your 7-segment decoder circuit using the *Processing/Start Compilation* menu item.



## SIMULATE THE DESIGN

Once compilation has been successfully completed do a functional simulation. This simulation should test *all 32 possible* input patterns of the input code.

As before, in order to carry out the simulation, you should insert an instance of the *gNN\_7\_segment\_decoder* symbol into your top-level project schematic diagram, and hook up inputs and outputs as needed.

Show the TA the results of your simulation.





## TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *fourth* 2-hour lab period!

### 3. Writeup of the Lab Reports

Write up two (2) short reports, describing each of the *gNN\_RANDU* and *gNN\_7\_segment\_decoder* circuits. Leave the description of the pop\_enable circuit for lab 3's report.

The reports must include the following items:

- A header listing the group number (and company name if you gave it one), the names and student numbers of each group member.
- A title, giving the name (e.g. *gNN\_7\_segment\_decoder*) and function of the circuit.
- A description of the circuit's function, listing the inputs and outputs. Provide a pinout or symbol diagram.
- The VHDL description of the circuit (don't embed this in the text of the report, instead include it as a separate file in the assignment submission zip file).
- A complete discussion of how the circuit was tested, showing representative simulation plots, and detailing what test cases were used.

The lab report, and all associated design files must be submitted, as an assignment to the myCourses site. Only one submission need be made per group (both students will receive the same grade!).

**Combine all of the files that you are submitting into one *zip* file, and name the zip file gNN\_LAB\_2.zip (where NN is your group number).**

**The reports are due one week after the last day of the lab period.**



# Grade Sheet for Lab #2

Fall 2017.

Group Number:\_\_\_\_\_.

Group Member Name:\_\_\_\_\_.

Student Number:\_\_\_\_\_.

Group Member Name:\_\_\_\_\_.

Student Number:\_\_\_\_\_.

Marks

	1.	<u>VHDL code for the <i>RANDU</i> circuit</u>	_____.
	2.	<u>Block Diagram Sketch of the <i>RANDU</i> circuit</u>	_____.
	3.	<u>Simulation of the <i>RANDU</i> circuit</u>	_____.
	4.	<u>VHDL code for the <i>pop_enable</i> circuit</u>	_____.
	5.	<u>Simulation of the <i>pop_enable</i> circuit</u>	_____.
	6.	<u>VHDL code for the <i>7_segment_decoder</i> circuit</u>	_____.
	7.	<u>Simulation of the <i>7_segment_decoder</i> circuit</u>	_____.
			TA Signatures

Each part should be demonstrated to one of the TAs who will then give a grade and sign the grade sheet. Grades for each part will be either 0, 1, or 2. A mark of 2 will be given if everything is done correctly. A grade of 1 will be given if there are significant problems, but an attempt was made. A grade of 0 will be given for parts that were not done at all, or for which there is no TA signature.