ECSE-323
Digital System Design

Lab #2 – ***Combinational Circuit Design with VHDL*** Fall 2015

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 selected signal assignment statements
• Include library design entities as components in your VHDL-based designs

Table of Contents .

This lab consists of the following stages:

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.

You will design the circuits needed to compute the score for the Mastermind game.

Learning Outcomes .

After completing this lab you should know how to:

• Describe a circuit with VHDL, using component statements as well as selected signal assignment statements
• Include library design entities as components in your VHDL-based designs

Table of Contents .

This lab consists of the following stages:

1. Learning about computing the score in the Mastermind game
2. Functional simulation of the circuit to compute the number of exact matches
3. Design and functional simulation of the circuit to compute the number of color matches
4. Design and functional simulation of the score encoder circuit
5. Integration and simulation of the complete scoring circuit
6. Writeup of the lab report

1. Computing the score in the Mastermind game .

In this lab you will create the circuitry needed to compute the score in the Mastermind game. This lab is more challenging than the first one, and you should read over the lab a few times before beginning, and make sure you understand the ideas behind the design!

The point of the Mastermind game is for one of the players to guess, in as few attempts as possible, the colors of four hidden pegs. Each peg can have one of six different colors. The other player gives feedback to the guesser after each guess in the form of a score. The guesser then modifies his guess based on the information provided by the score.

There are two components to the score given to a guess in the Mastermind game:

• **score1 :** The number of pegs that were guessed exactly ***(right position and right color)***

• **score2 :** The number of pegs whose color was guessed correctly, but for which the position guess was incorrect ***(wrong position and right color)***. This part of the score is computed by considering only the pegs that are not exact matches.

Examples of Mastermind Scores

ACTUAL PATTERN	GUESSED PATTERN	SCORE (num_exact, num_color_matches)
🔴🔵🟡🔴	⚫⚪🔴🟡	(0, 2)
	🔴🟢🟡🔵	(2, 1)
	🔵🔵⚪🔴	(2, 0)
	🔴🔵🔵🔴	(3, 0)
	🔵🔵🔵🔵	(1, 0)
	🔴🔵🟡🔴	(4, 0)

2. Design of the circuit for computing the number of exact matches

The first part of the score (***the number of exact matches***) is easy to compute.

You need to compare the color codes for each of the four guess pegs and their corresponding true pattern pegs, and then count how many match.

Fortunately, you have already developed the circuits needed to do this - the ***gNN_comp6*** and ***gNN_num1s*** circuits. These just have to be included as components and connected up in the proper way.

The outline for complete VHDL description is shown on the next page (you need to fill in the architecture part). Use it as a general template for all of the descriptions that you write for the rest of the lab.

VHDL Description of the num_matches circuit.

```
-- exact match counting part of the scoring circuit for the Mastermind game
--
-- entity name: g00_num_matches
--
-- Copyright (C) 2015 James Clark
-- Version 1.0
-- Author: James J. Clark; clark@cim.mcgill.ca
-- Date: June 25, 2015

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

entity g00_num_matches is
 port ( P1, P2, P3, P4    : in std_logic_vector(2 downto 0);
        G1, G2, G3, G4    : in std_logic_vector(2 downto 0);
        N : out std_logic_vector(2 downto 0));
end g00_num_matches;

architecture a of g00_num_matches is

  --
  -- you fill in this part!
  --

end a;
```

SIMULATE THE DESIGN

Using the techniques learned in lab #1, do a ModelSim simulation of the *g00_num_matches* circuit. This simulation should test a small selection (say 8) of all the possible combinations of the inputs (which are 2^24 in number).

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 *first* 2-hour lab period!

3. Design of the circuit for computing the number of color matches

The computation of the number of color matches (the second part of the score) is trickier than computing the first part of the score. It can be computed with the following formula:

$$\text{score2} = \left(\sum_{\mathrm{i=1}}^{i=num\_colors} \min(NP_i, NG_i) \right) - N_E$$

where NP_i is the number of pegs in the true pattern with color i, NG_i is the number of pegs in the guess pattern with color i, and N_E is the number of exact matches (i.e. the first part of the score).

The implementation of this equation is straightforward, but involves many, many gates! Don't be discouraged by the size of the circuit. Be brave!

The vhdl code for the color_matches circuit can be broken up into 3 sub-parts:

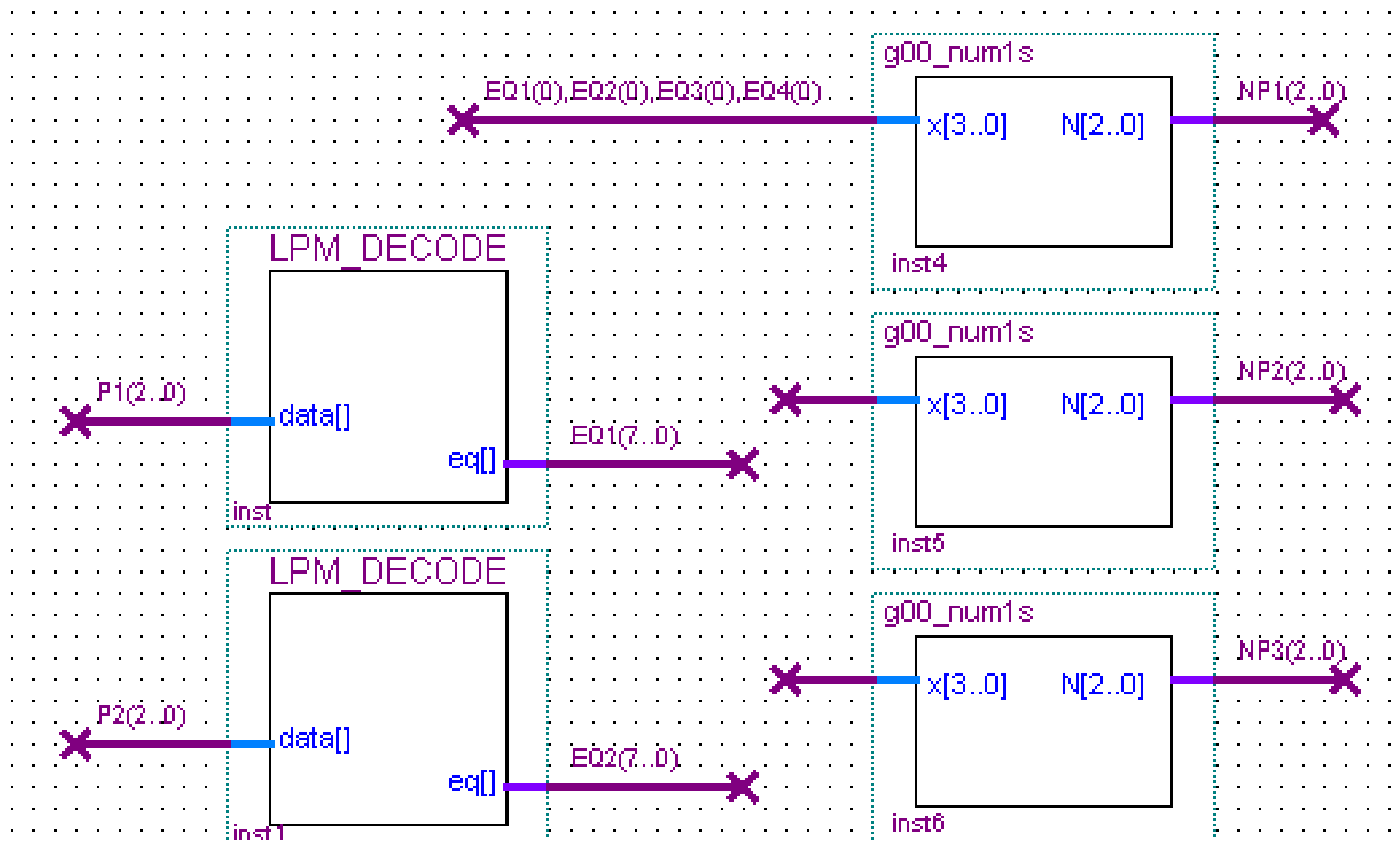
1. Computation of ***NPi*** and ***NGi*** for each color *i*
2. Implementation of the minimum operations
3. Summation of the minimum values (and subtraction of the number of exact matches)

For sub-part 1, you should use the approach shown on the ***next few pages***, using Altera ***lpm_decode*** library modules and the ***gNN_num1s*** circuit you designed in lab 1. (although the schematic diagram is shown, you will describe the circuit using VHDL)

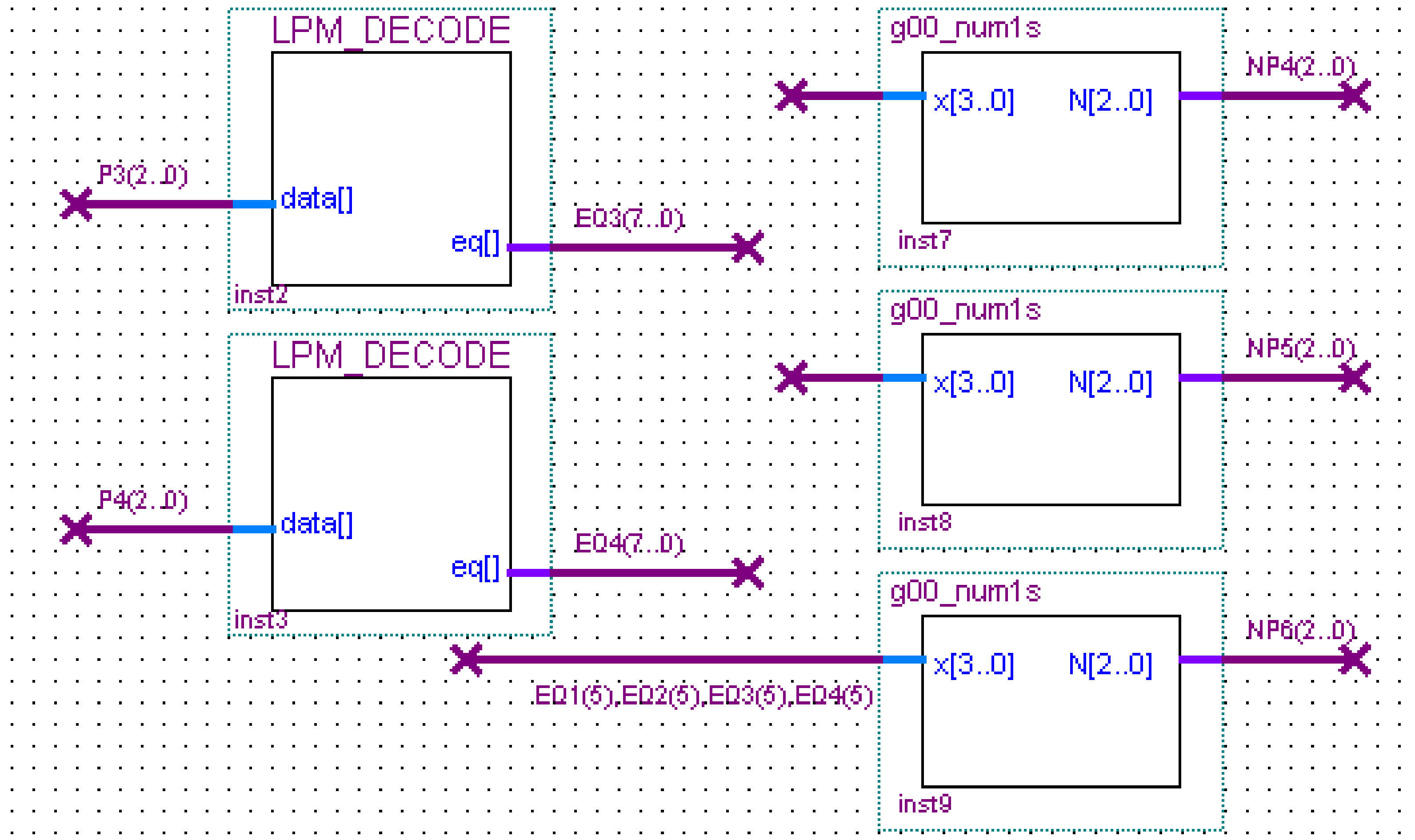
For sub-part 2 you need to design a new module to implement the ***minimum*** function (details later in this lab). This will then be included as a component in the color_matches vhdl description.

For sub-part 3 you will use Altera ***lpm_add_sub*** library components to implement the addition and subtraction operations.

Top part of the schematic diagram for the circuit to compute *NPi*



Bottom part of the schematic diagram of the circuit to compute *NPi*



The idea behind this circuit is that each peg has a color (one of six possible) that is represented by the 3-bit binary word *Pi* (where *i* indicates which of the four pegs we are looking at). The decoder module has an 8-bit output, *eq*. Only one of the bits of *eq* is high at a given time, while all the others are low.

Which decoder output bit is high is determined by the value of the decoder input. For example if *Pi=[0,1,1]* then *eq=[0,0,0,0,1,0,0,0]*. The 2 MSBs will always be zero, since there are only 6 colors.

Thus, we can see that the output of the decoder tells us the color of the peg.

To compute the number of pegs with a given color *i*, we need to input the *ith* outputs of each of the four decoders into the 4-bit ***num1s*** counter circuit.

Use an identical circuit to compute ***NGi***.

For now, just try to understand how this circuit is to work. We will get to its implementation in VHDL in a little while. But first we will look at the implementation of the minimum function.

The Boolean equation for a "greater-than" function is given on page 341 (p. 339 in the 2nd edition or p. 304 in the 1st edition) of the textbook. Use this to implement a "minimum" circuit, that takes in two 3-bit binary numbers and passes through to the output the smaller of these.

(note: the expression on p. 341 is for the 4-bit case - you can derive the 3-bit case simply by setting the fourth bits in the expression both to zero.)

Use VHDL to describe your circuit, using a single Concurrent Assignment statement to implement the comparator logic and a ***Selected Signal Assignment*** statement to pass the proper input to the output based on the output of the comparator.

Use the following *entity declaration* for your design (remembering to replace NN with your group number):

```
entity gNN_minimum3 is
 port ( N, M    : in std_logic_vector(2 downto 0);
        min : out std_logic_vector(2 downto 0));
end gNN_minimum3;
```

CREATE THE DESIGN FILE AND SYMBOL

Save the file as ***gNN_minimum3.vhd*** and create a symbol for it.

Once you have finished writing the ***complete*** VHDL description of your ***gNN_minimum3*** circuit, show it to your TA.

SIMULATE THE DESIGN

Using ModelSim, do a functional simulation of the *gNN_minimum3* circuit. This simulation should test all the possible combinations of the inputs (which are 2^6=64 in number).

Show the TA the VHDL testbed code and the results of your simulation.

TIME CHECK

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

Once you are satisfied with your minimum circuit, finish writing the VHDL description of the complete color_match score circuit.

Use the entity declaration shown below, and add the architecture section.

The architecture section will have a ***lot*** (more than 30!) of component instantiation statements, so you will have to check your description carefully for errors.

```
entity gNN_color_matches is
 port ( P1, P2, P3, P4    : in std_logic_vector(2 downto 0);
        G1, G2, G3, G4    : in std_logic_vector(2 downto 0);
        num_exact_matches : in std_logic_vector(2 downto 0);
        num_color_matches : out std_logic_vector(2 downto 0));
end gNN_color_matches;
```

As stated earlier, for the DECODE and ADD/SUBTRACT operations, it will be convenient to use the modules provided by the Altera Library of Parameterized Modules (***lpm_decode*** and ***lpm_add_sub***).

In order to use these library modules, you must include the following lines at the ***beginning*** of your VHDL description (i.e. before the entity declaration):

```
library lpm; -- include the Altera lpm library
use lpm.lpm_components.all;
```

For the lpm modules, you do not need to include any component declaration statements in the architecture block, as these declarations are included in the libraries. You just need to make the instantiation statements.

You should look at the Quartus ***HELP*** to get information to see the component declaration for the modules. You will need to know the name of the module PORT signals in order to do the instantiations.

CREATE THE DESIGN FILE AND SYMBOL

Write the complete vhdl description of the color_match circuit, and save it as ***gNN_color_matches.vhd***.

Create a symbol for the circuit.

Show the ***complete*** VHDL description of your ***gNN_color_matches*** circuit to your TA.

SIMULATE THE DESIGN

Do a functional simulation of the *gNN_ color_matches* circuit. There is a very large number of (2^32) possible combinations of the inputs and it will be too time-consuming to check all possible cases by hand in simulation. So just select 8 different cases to check in the simulation.

Write the VHDL testbed to generate the circuit inputs for the 8 selected cases, and run the simulation using ModelSim.

Show the TA the testbed code and 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!

4. Design of the Score Encoder Circuit .

The two parts of the score each take on values from 0 through 4. Thus 3 bits are needed for each part, for a total of 6 bits for the full score. This suggests that there are a total of 32 different possible scores.

It is easily seen, however, that not all scores are possible. For one thing, since we only check for color matches on the pegs which are not exact matches, the sum of the two parts of the score cannot exceed 4 (since we only have 4 pegs!). This means that only the following scores are allowed (since their sum of the two parts are not greater than 4):

(4,0) (3,0) (3,1) (2,0) (2,1) (2,2) (1,0) (1,1) (1,2) (1,3) (0,0) (0,1) (0,2) (0,3) (0,4)

Of these 15 allowed scores, the score (3,1) will never occur (*why is this???*).

Thus there are ***only 14 possible scores***, and we can therefore encode the two parts of the score with a single 4-bit number. Later in the system development, we will need to implement a table memory to hold score values, so it makes sense to compress the score to 4-bits so that the table memory size is minimized.

Write a complete VHDL description of the score_encoder circuit. Use the entity declaration shown below and add the architecture section. You should use a process block containing a single ***case statement*** with 32 cases to implement the encoding.

```
entity gNN_score_encoder is
 port ( score_code : out std_logic_vector(3 downto 0);
        num_exact_matches : in std_logic_vector(2 downto 0);
        num_color_matches : in std_logic_vector(2 downto 0));
end gNN_score_encoder;
```

CREATE THE DESIGN FILE AND SYMBOL

Save the file as ***gNN_score_encoder.vhd*** and create a symbol for it.

Show the ***complete*** VHDL description of your ***gNN_score_encoder*** circuit to your TA.

SIMULATE THE DESIGN

Using ModelSim, do a functional simulation of the *gNN_score_encoder* circuit. This simulation should test ***all*** of the possible combinations of the inputs (which are 2^6 in number).

Show the TA the VHDL testbed code and the results of your simulation.

5. Integration of the Complete Scoring Circuit

Now is the time to put all of the pieces together to make the complete scoring module.

This is not too complicated, as it is just a matter of instantiating as components the 3 modules that you have designed in this lab: *gNN_num_matches*, *gNN_color_matches*, *gNN_score_encoder*.

Use the following entity declaration:

```
entity g00_mastermind_score is
 port ( P1, P2, P3, P4     : in std_logic_vector(2 downto 0);
        G1, G2, G3, G4     : in std_logic_vector(2 downto 0);
       exact_match_score : out std_logic_vector(2 downto 0);
       color_match_score : out std_logic_vector(2 downto 0);
        score_code : out std_logic_vector(3 downto 0));
end g00_mastermind_score;
```

Show the TA your completed VHDL description of the *g00_mastermind_score* circuit.

SIMULATE THE DESIGN

Using ModelSim do a simulation of the *gNN_mastermind_score* circuit. There are 2^24 (>16 Million) possible combinations of the inputs, which is too many to check manually. So, just select 8 different input patterns to test.

The fact that there are more than 16 million possible input patterns but only 14 different score values is what makes this a challenging game.

Show the TA the VHDL testbed code and 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!

4. Writeup of the Lab Report .

Write up a report describing the ***gNN_mastermind_score*** circuit. You do not need to provide reports for the other modules (but you should include their vhdl descriptions).

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_mastermind_score***) 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 all circuits designed in this lab, including the testbenches (don't embed these in the text of the report, instead include them 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 which 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 at midnight, Friday October 23.

Grade Sheet for Lab #2 Fall 2015.

Group Number:_______________.

Group Member Name:_________________. Student Number:___________________.

Group Member Name:_________________. Student Number:___________________.

Marks			TA Signatures
	1.	Simulation of the *g00_num_matches* circuit	.
	2.	Show the VHDL code for the *gNN_minimum3* circuit	.
	3.	Simulation of the *gNN_minimum3* circuit	.
	4.	Show the VHDL code for the *gNN_color_matches* circuit	.
	5.	Simulation of the *gNN_color_matches* circuit	.
	6.	Show the VHDL code for the *gNN_score_encoder* circuit	.
	7.	Simulation of the *gNN_score_encoder* circuit	.
	8.	Show the VHDL code for the *gNN_mastermind_score* circuit	.
	9.	Simulation of the *gNN_mastermind_score* circuit	.

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.