

# ECSE-323

# Digital System Design

**Lab #2** – *Combinational Circuit Design with VHDL* Fall 2008

# 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
- Include library design entities as components in your VHDL-based designs
- Use ROM modules to implement look-up tables
- Assign circuit outputs and inputs to devices on the Altera DE1 board
- Configure the Cyclone II FPGA on the Altera DE1 board

# Table of Contents

*This lab consists of the following stages:*

- Design and timing simulation of a log base 2 function circuit
- Design, functional simulation, and testing on the Altera board of a Binary to 7-segment LED decoder circuit
- Writeup of the lab reports

## Course Project – Design of a Base-100 Financial Calculator

Your job for this term is to develop a financial calculator, for an avaricious race of aliens who are not very technically savvy, but have a lot of money and like to make investments. They also have a lot of fingers – 100 to be exact, so they do all of their arithmetic in base-100.

More details on what sort of functions this calculator is to have will be given in later labs. For this lab you will focus on developing a logarithm computation circuit (useful in some financial calculations) and a decoder used for displaying base-100 numbers on a 7-segment LED.

## 1. Design of a log Function circuit using a Lookup Table.

In this part of the lab you will create a circuit that computes the *log base 2* function. It will take as input a **12-bit** binary word representing a number between 1 and 2, and produce as output the logarithm, base-2, of this number.

The output should be a **16-bit** fractional binary word representing a number between 0 and 1 (represent 1 by all ones).

[ Note that if you want to compute the log base 2 of a number outside of this range you can merely figure out the power of 2 needed to divide this number to bring it into the range 1-2, and add this power to the result.

For example:  $4.7 = 2^2 * (1.175)$ ;  $\log_2(4.7) = 2 + \log_2(1.175)$  ]

## REPRESENTING FRACTIONAL NUMBERS

Binary numbers can be used to represent fractional values (e.g 42.71) just as easily as integer values. It is all a question of where you place the binary point.

Think of the bits in a binary number as being weighted by powers of 2 (just the digits in a decimal number are weighted by powers of 10). If you are given a fixed number of bits (say 7) then you have to specify the range of powers of 2 that are being represented. For example, we could use the 7 bits to represent powers of 2 from 2 down to -4:

$$V = b_6 2^2 + b_5 2^1 + b_4 2^0 + b_3 2^{-1} + b_2 2^{-2} + b_1 2^{-3} + b_0 2^{-4}$$

Using this representation a binary number 1101101 would represent the value  $2^2 + 2^1 + 2^{-1} + 2^{-2} + 2^{-4} = 6.8125$

For example, to represent numbers between 0 and 1, with 28 bits, you would use powers of 2 from -1 through -28. For the input you need to represent numbers between 1 and 2, so you would use powers of 2 from 0 through -27.

We could implement the log function using a numerical method such as evaluating the Taylor's series approximation. That approach would require a number of multiplication operations, which would use up both space (multipliers take a large number of gates) and time (long propagation delays). In this lab we will use a simpler approach - that of using a lookup table (LUT). In the LUT approach we use a memory unit that has an entry for every possible input pattern. The input bits are then used as an address for the memory.

Many modern FPGA devices, such as the Cyclone II 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* (Library of Parametrized Modules) modules to implement the LUT. You will write a VHDL description of the sine computation 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 => 8, -- sets the width of the ROM address bus  
    lpm_numwords => 256, -- sets the 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(inclock => clock, address => x, q => crc_of_x);
```

**Note that memory blocks on the Cyclone II chip must have registered inputs. Therefore we 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. You could create this file by hand, but since it will have  $2^{12}$  entries you should probably use a computer program (such as one written in C, or using Matlab) to create the file. *Prepare this file before coming into the lab!*

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.

## VHDL Description of the complete log2 function circuit.

The entity declaration should have the following form (remember to replace the header with your own information)

```
-- this circuit computes the log base 2 of the input
--
-- entity name: g00_log2
--
-- Copyright (C) 2008 James Clark
-- Version 1.0
-- Author: James J. Clark; clark@cim.mcgill.ca
-- Date: September 20, 2008

library ieee; -- allows use of the std_logic_vector type
use ieee.std_logic_1164.all;
library lpm; -- allows use of the Altera library modules
use lpm.lpm_components.all;

entity g00_log2 is
  port (
    clock      : in std_logic;
    input_value : in std_logic_vector(11 downto 0);
    log2       : out std_logic_vector(15 downto 0));
end g00_log2;
```

## VHDL Description of the log<sub>2</sub> function circuit.

The architecture body will contain the functionality of the circuit. You will instantiate the lookup table, and whatever other circuitry you deem to be required.

With a resolution in the input of 12 bits, the lookup table would need 4096 entries to hold results for all possible input cases. But note that the MSB of the input will always be “1”, since the input is a number between 1 and 2. So we only need for the lookup table to hold  $2^{11}$  or 2048 entries. The MSB can be ignored.

Create the .mif file using values computed by a hand calculator or computer program. Show the *mif* file and the *vhd* file to the TA.





## TIME CHECK

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

## SIMULATE THE DESIGN

First, using the techniques learned in lab #1, do a *functional* simulation of the *g00\_log2* circuit.

To generate the clock signal for the memory register, use the *overwrite clock* item on the Waveform Editor. Set the period to 50nsec. This will load the input register on each rising edge of the clock signal.

This simulation should test all 4096 input patterns. Compare the results to the entries in your .mif file. They should match.

Show the TA the results of your simulation.



## Viewing the Compilation Report

Look at the *Flow Summary* section of the Compilation Report and note the FPGA resource utilization (i.e. how many logic elements were used?). You will include this information in your report.

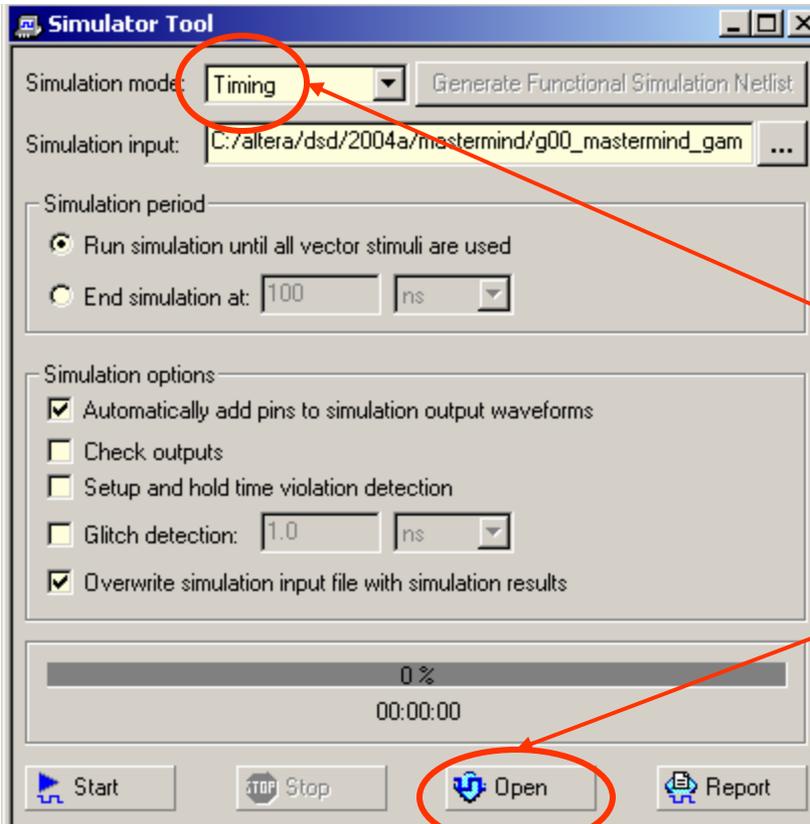
Read over the datasheet for the Cyclone II chip (available on the CD included in your lab kit) to understand the architecture of the device.

Finally, look at the *Timing Analyzer Summary* (under the *Timing Analyzer* section of the compilation report). Take note of the path with the largest propagation delay. Include this in your report. It is an important number, as it determines the maximum speed of any circuit that uses your design.

Show the Flow Summary and Timing Analyzer Summary to the TA.



Next, you will do a “Timing” simulation of the log2 circuit. This will take the various delays in your circuit into account. You should use timing simulations when you want to know the propagation delay for your circuits.



Select the Simulator Tool item from the Processing menu. A window like the one at the left will appear.

Select "Timing" as the simulation mode.

Click on "Open" to bring up the Waveform Editor.

In the waveform editor, enter a sequence of 4 different input values, where the transitions between different inputs are spaced 1000nsec apart. Use an end time of 5,000nsec (or 5usec). Use a clock signal with a period of 50nsec.

Use the following 4 input values:

**000000000000, 111111111111, 010101010101, 101010101010**

Check to see if the output values are correct, but also see how long it takes for the output to settle down to a stable value after the transition. Is this settling time the same for every transition? How does this settling time compare to the propagation delays reported by the Timing Analyzer?

Show the results of your simulation to the TA.



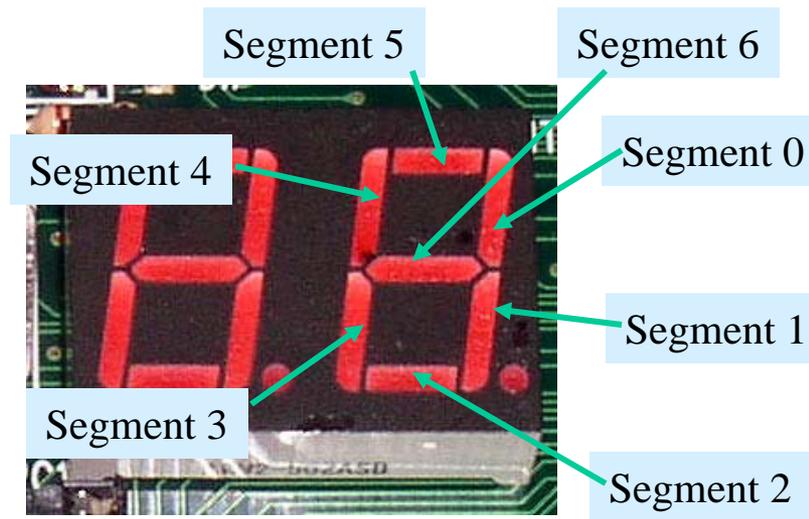


## TIME CHECK

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 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 on the Altera board, which you will use later in your full implementation of the base-100 calculator.



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 7-bit binary code representing the 100 digits between 00 and 99, and generates the 7-segment display associated with the input code, as shown on the following pages.

*Note: 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.*



**IMPORTANT**

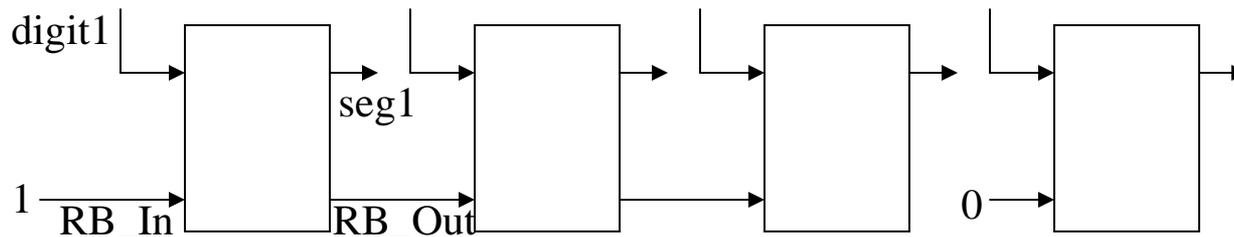
00-09	0	1	2	3	4	5	6	7	8	9
10-19	A	b	C	d	E	F	G	H	I	J
20-29	K	L	M	N	O	P	Q	R	S	T
30-39	U	V	W	X	Y	Z	_	-	=	
40-49	~	^	&	^	C		U	J	7	∩
	—	—	—							

50-59										
60-69										
70-79										
80-89										
90-99										

All other input codes should give a blank display (no segments on)

Your circuit should have *ripple-blanking* capability. Ripple blanking is the turning off of leading zeroes in a multi-digit display. For example, suppose we had a 4 digit display, with one decimal point. Thus we could display numbers such as **241.2** and **788.6**. But what about displaying numbers with value less than 100? If we didn't blank the leading zeroes our display would look like **009.5** or **083.4**. This looks ugly and unprofessional, so we would rather display **9.5** and **83.4** in these cases, where the leading zeroes have been suppressed.

The ripple blanking output should be connected to the ripple-blanking input of the next display decoder to the right. The ripple-blanking input of the left-most LED decoder should be connected to '1'. The rightmost LED decoder, and any decoders right of the decimal point should never be blanked, so their ripple blanking inputs should be connected to '0'. In the case described above, the display circuit would look like:



To implement the 7-segment LED decoder, write a VHDL description using *a single selected signal assignment statement*. (this will have 101 cases, so use cut-and-paste!)

Use the following *entity declaration*, replacing the gNN in gNN\_7\_segment\_decoder with your group's number (e.g. g08). You will have to supply the architecture body...

```
entity gNN_7_segment_decoder is
  port ( code          : in std_logic_vector(6 downto 0);
        RippleBlank_In : in std_logic;
        RippleBlank_Out : out std_logic;
        segments       : out std_logic_vector(6 downto 0));
end gNN_7_segment_decoder;
```

## 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.

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

Compile the *gNN\_7\_segment\_decoder* circuit and do a *functional* simulation. This simulation should test *all 128 possible* input patterns of the input value. You should also demonstrate the proper operation of the ripple-blanking function.

Show the TA the results of your simulation.





## TIME CHECK

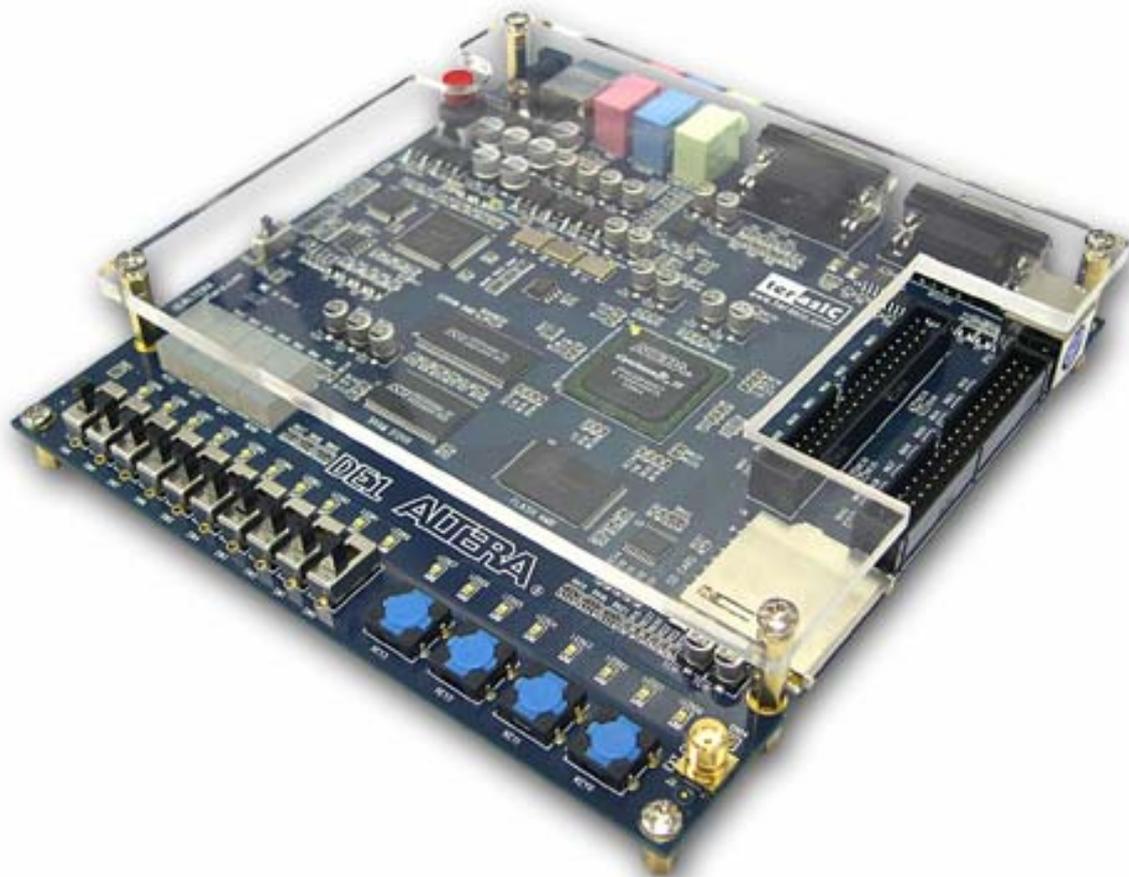
You should be this far (i.e. have completed the lab) at the end of your *third* 2-hour lab period!

### 3. Obtain the Altera Design Laboratory Kit

For the remainder of the lab experiments, groups will be using the Altera DE1 Development and Education Board. This package includes:

- Altera DE1 Board with a *Cyclone II EP2C20F484C7* FPGA
- Altera DE1 CD-ROM - Version 0.5 with documentation
- Altera Quartus II DVD - Version 7.0
- 1 Power Supply Adapter DC 7.5V/0.8A (US wall plug)
- 1 USB Cable
- 6 Silicon Footstands
- 2 Cables (black- and red-colored)
- 2 PIN Headers, 1P1N

# The Altera DE1 Development and Education Board



Each group will have their own package, which they can keep with them until the end of the course. To obtain the lab kit, *all* of the group members should go to the ECE department Technician's office, located in room 4140 of the Trottier building. Ask for *Mr. Charles Burtles*. He will have a list of the lab groups for this course, and will give you one of the Altera lab kits after you present appropriate identification (McGill student IDs).

*All group members must be present and display their ID card!*

*Print out and sign the waiver form (from the WebCT Experiments page) accepting responsibility for the kit. Bring this waiver with you when you go to pick up the lab kit.*

**All members of the group must be present in order to receive the kits.**

*Please note that you are responsible for any loss of or damage to the kits. The list price for the Altera DE1 kits is currently \$149 (price in US\$).*

## Testing the LED Decoder on the Altera Board

Once you compiled the LED decoder circuit, it is time to map it onto the target hardware, in this case the Cyclone II 2C20 chip on the Altera DE1 board. Please begin by reading over the DE1 user's manual, which can be found on the documentation CD provided as part of your lab kit.

Since you will now be working with an actual device, you have to be concerned with which device package pins the various inputs and outputs of the project are connected to.

In particular, you will want to connect the LED segment outputs from the instances of the *gNN\_7\_segment\_decoder* circuit to the corresponding segments of one of the four 7-segment LED displays on the Altera board.

**The mapping of the Altera Board's LED display segments to the pins on the Cyclone FPGA device is listed in *Table 4.4 on page 31* of the DE1 user's manual.**

You will also want to connect, for testing purposes, 7 of the *slide switches* on the DE1 board to the inputs of the *gNN\_7\_segment\_decoder* circuit.

The mapping of the slide switches to the FPGA pins is given in Table 4.1 on pages 28 and 29 of the DE1 user's manual.

You can tell the compiler of your choices for pin assignments for your inputs and outputs by opening the *Pin Planner*, which can be done by choosing the *Pins* item in the *Assignments* menu, as shown in the screenshot on the next page.

Top View - Wire Board  
Cyclone II - EP2C20F484C7

Groups:  
Named: [ ]

Node Name  
q[6..0]  
<<new node>>

This cell specifies the pin number, I/O bank, or edge location to which you want to assign the pin.

Named: [ ] Edit: [X] [✓] Filter: Pins: all

	Node Name	Direction	Location	I/O Bank
1	clock	Input		
2	q[6]	Output	PIN_H17	I/O Bank 5 Row I/O LVDS69p
3	q[5]	Output	PIN_H18	I/O Bank 5 Row I/O LVDS69n
4	q[4]	Output	PIN_H19	I/O Bank 5 Row I/O LVDS74p
5	q[3]	Output	PIN_J1	I/O Bank 2 Row I/O LVDS16p, DPCLKC
6	q[2]	Output	PIN_J2	I/O Bank 2 Row I/O LVDS16n
7	q[1]	Output	PIN_J4	I/O Bank 2 Row I/O
			PIN_J14	I/O Bank 4 Column I/O LVDS55p
			PIN_J15	I/O Bank 5 Row I/O

Enter the schematic pin names in the “Node Name” boxes (using the Edit box). Then enter the corresponding FPGA pin using the Location box (either by typing into the edit box or double-clicking to get the popup list of nodes as shown).

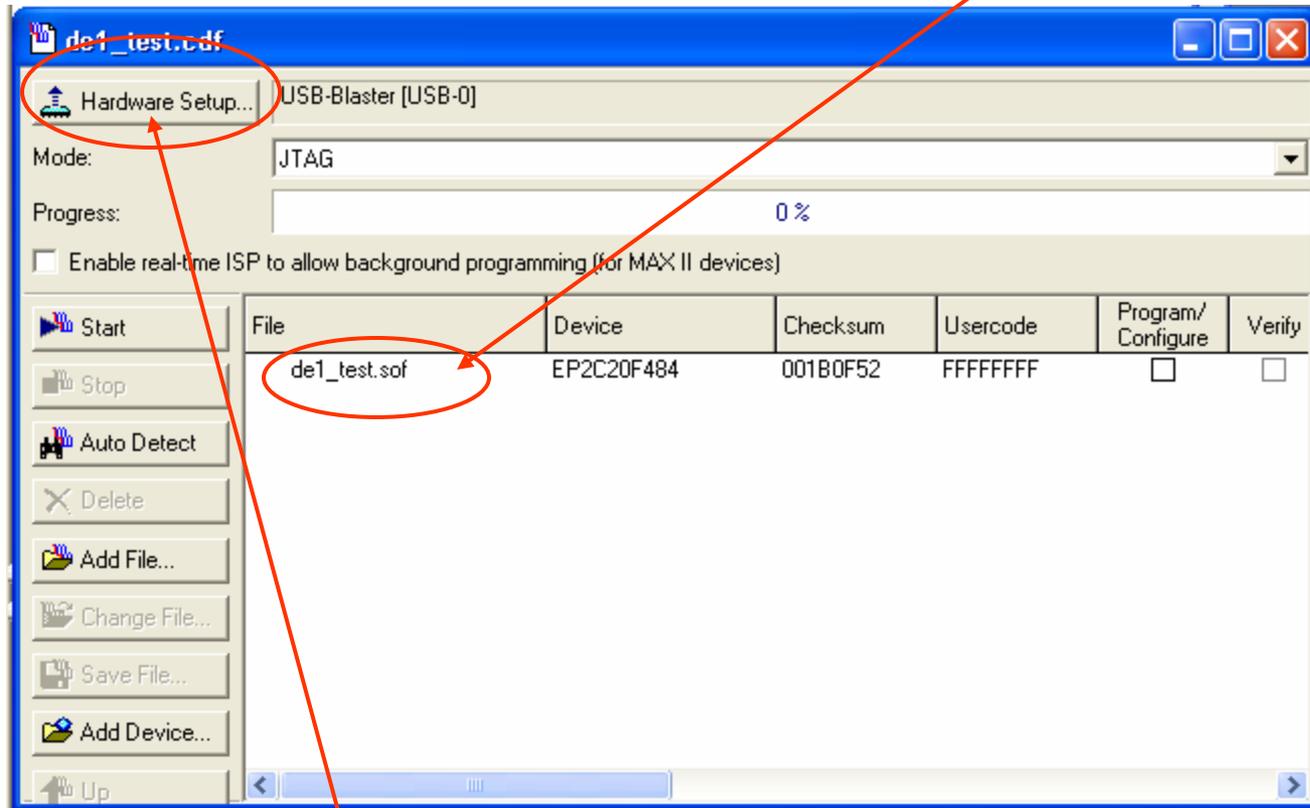
Once you have assigned all of the inputs and outputs of your circuit to appropriate device pins, *re-compile* your design.

**You can check that the pins have been assigned correctly by looking at the floorplan on the pin planner (zoom in), and verifying that the right pins have been used.**

Your design is now ready to be downloaded to the target hardware. Read section 4.1 of the DE1 user's manual for information on configuring (programming) the Cyclone II FPGA on the board. You will be using the *JTAG mode* to configure the device.

**Take the Altera board out of the kit box, and connect the USB cable to the computer's USB port and to the USB connector on the Altera board.**

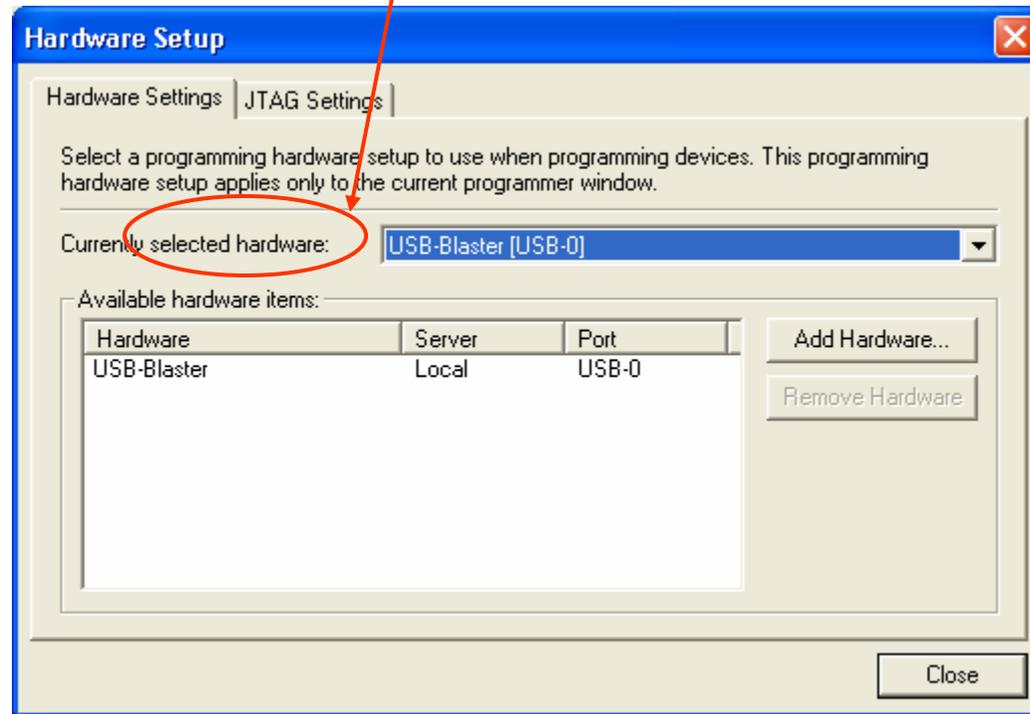
Next select the *Programmer* item from the *Tools* menu. You should see a window like the one shown below. There should be a file listed. If not, click “Add File”.



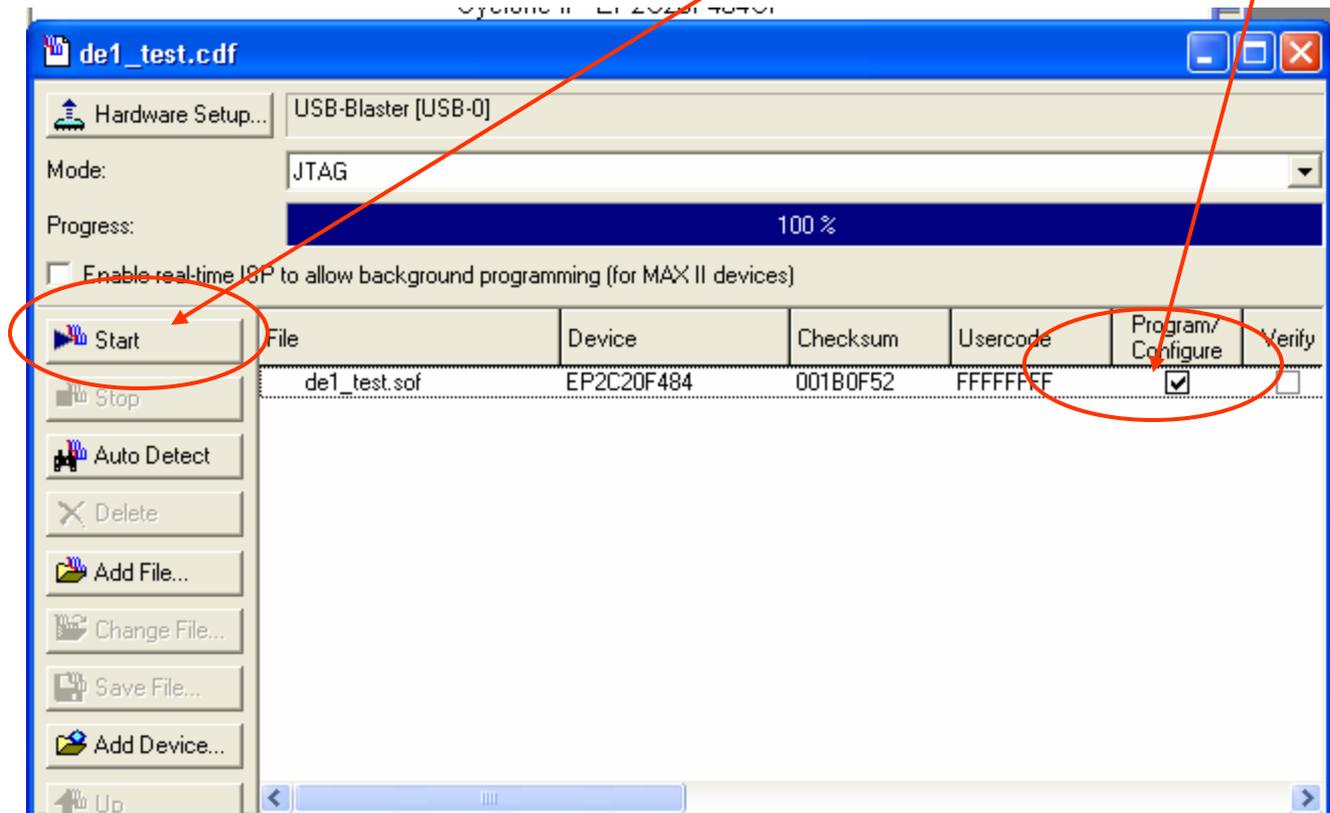
If there is no device visible, click on *Hardware Setup*

In the Hardware Setup window, select the correct communication hardware. Select “*USB-Blaster*”.

Click on Close to return to the Programmer window. If you still do not see a device, check the jumper settings on the board, and make sure that the USB cable is connected.



If everything seems in order (i.e. a file and a device are shown) you can carry out the FPGA programming. To do this, click on **Start**. Make sure that the Program/Configure checkbox is checked.



Demonstrate to the TA that your circuit is functioning properly by going through some of the 128 different switch settings.

Compare the outputs you see on the LEDs to the charts on page 22 and 23 of this lab description.

You don't have to demonstrate all 128 possibilities to the TA, just 4 or 5 will do. But you should test all 128 yourself, to make sure your circuit is working properly.





## 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 Reports

Write up two (2) short reports, describing each of the *gNN\_log2* and *gNN\_7\_segment\_decoder* circuits.

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 WebCT 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 17.**



# Grade Sheet for Lab #2

Fall 2008.

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

Group Member Name: \_\_\_\_\_ Student Number: \_\_\_\_\_.

Group Member Name: \_\_\_\_\_ Student Number: \_\_\_\_\_.

Marks


- |   |  |        |
|---|--|--------|
| • | <u>Mif file and VHDL code for the <i>log2</i> circuit</u>                  | _____. |
| • | <u>Functional Simulation of the <i>log2</i> circuit</u>                    | _____. |
| • | <u>Timing Simulation of the <i>log2</i> circuit</u>                        | _____. |
| • | <u>VHDL code for the <i>7_segment_decoder</i> circuit</u>                  | _____. |
| • | <u>Simulation of the <i>7 segment decoder</i> circuit</u>                  | _____. |
| • | <u>Testing of the <i>7 segment decoder</i> circuit on the Altera board</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.**