ECSE-323
Digital System Design

Lab #2 – *Combinational Circuit Design with VHDL* Winter 2014

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 simulation of a time conversion circuit
- Design and simulation of an binary-to-bcd circuit
- Design, simulation, and testing on the Altera board of a Binary to 7-segment LED decoder circuit
- Writeup of the lab reports

1. Design of an H:M:S to Seconds circuit using VHDL.

In this part of the lab you will create a circuit that takes in a time specified in hours, minutes, and seconds, and compute the time specified as the number of seconds since midnight.

The inputs to the circuit are all unsigned signals: *Hours* (5 bits), *Minutes* (6 bits) and *Seconds* (6 bits). The output is a 17-bit unsigned signal to be named *DaySeconds*.

You are to use the Altera LPM (Library of Parametrized Modules) modules ***lpm_multiply*** and ***lpm_add_sub*** as components to implement the required arithmetic operations:

DaySeconds = Seconds + (60 * Minutes) + (3600 * Hours) or
DaySeconds = Seconds + (60 * (Minutes + (60 * Hours))

(which formulation do you think would result in the smaller circuit?)

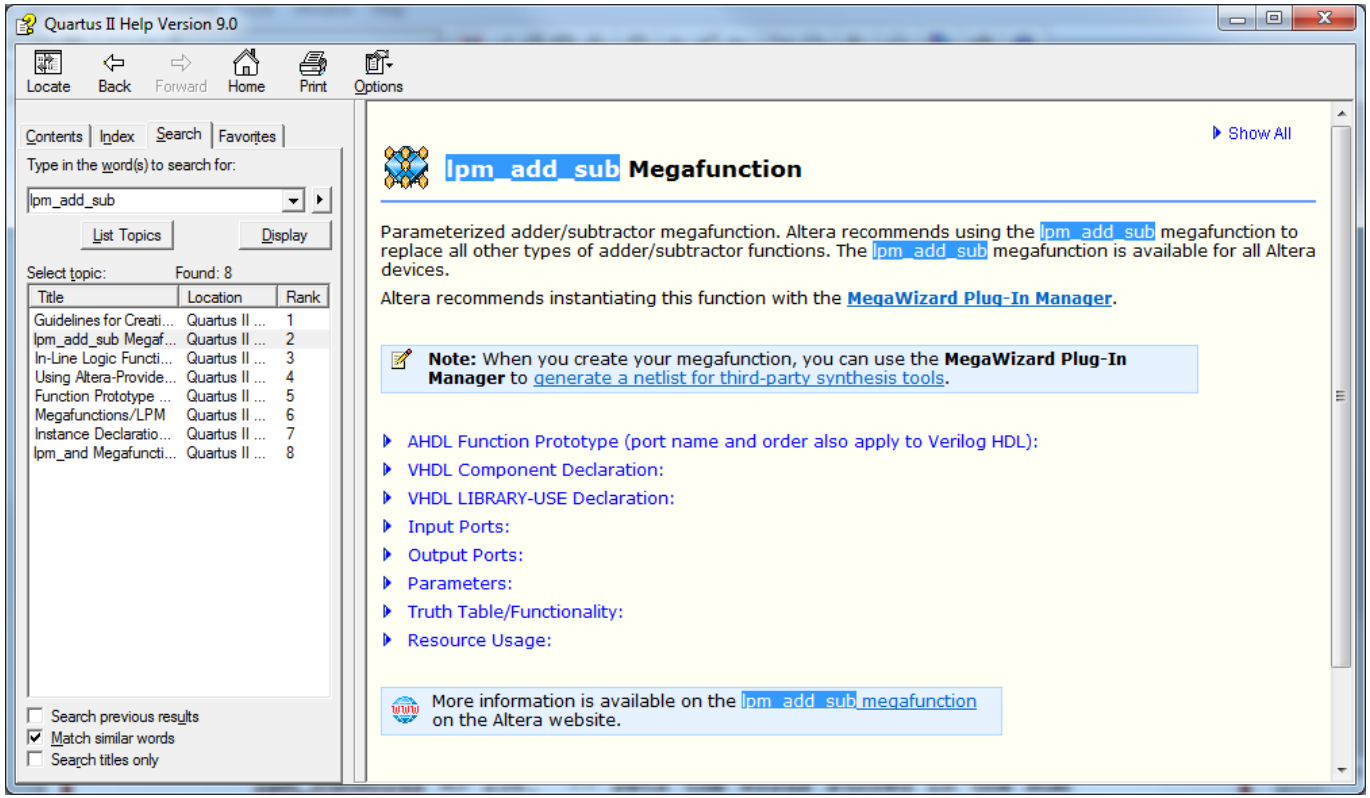
To use the *lpm* modules in your design, you must include the following two lines at the beginning of your design entity:

```
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(clock => clk, address => x, q => crc_of_x);
```

Search the Quartus Help pages for information on the port mappings and the generic parameters for the *lpm_mult* and *lpm_add_sub* modules.



Note: To use any 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 do need to include the COMPONENT statement in your design entity.

VHDL Description of the time converter circuit.

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

```
-- this circuit computes the number of seconds since midnight given
-- the current time in Hours (using a 24-hour notation), Minutes, and Seconds
--
-- entity name: g00_dayseconds
--
-- Copyright (C) 2014 James Clark
-- Version 1.0
-- Author: James J. Clark; james.j.clark@mcgill.ca
-- Date: January 19, 2014

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

library lpm;
use lpm.lpm_components.all; -- allows use of the Altera library modules

entity g00_dayseconds is
 port (   Hours            : in unsigned(4 downto 0);
          Minutes, Seconds : in unsigned(5 downto 0);
          DaySeconds       : out unsigned(16 downto 0));
end g00_dayseconds;
```

Your job is to fill in the rest of the design entity (i.e. the architecture)!
Show your complete VHDL description to the TA.

SIMULATE THE DESIGN

Once you have written the vhdl description compile the circuit.

Then, using the techniques learned in lab #1, do a ***functional*** simulation of the *gNN_dayseconds* circuit.

This simulation should test all 2^17 possible input patterns.

Compare the results to the expected values. You don't have to check every single value, but you should be able to see a pattern indicating that the results are correct.

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!

2. Design of a binary-to-BCD converter using a Lookup Table.

In this part of the lab you will create a circuit that converts between a 6-bit binary value and its 2-digit Binary-Coded-Decimal (BCD) representation.

For example, the BCD representation of the binary value **110101** is **0101 0011**.

The input ***BIN*** will be a 6 bit unsigned signal and the output ***BCD*** will be an 8 bit std_logic_vector.

You should get it clear in your mind why we treat the BCD output as an *std_logic_vector* instead of as an *unsigned* signal!

You could implement the conversion function using a 6-input 8-output Boolean logic function. This would be tedious to design and would use many logic 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, in this case the ***lpm_rom*** component.

As seen earlier, the ***component instantiation statement*** for the ***lpm_rom*** is

```
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(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.

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.

As usual, information on the format of the .mif file can be obtained from the Quartus help facility.

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 radixes used to display and interpret addresses and data values.

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

```
DEPTH = 64;                  % Memory depth and width are required %
WIDTH = 8;                   % Enter a decimal number %

ADDRESS_RADIX = HEX;  % Address and value radixes are required %
DATA_RADIX = HEX;       % Enter BIN, DEC, HEX, OCT, or UNS; unless %
                             % otherwise specified, radixes = HEX %
-- Specify values for addresses, which can be single address or range
CONTENT
          BEGIN
[0..F] : 3F;                 % Range of addresses--Every address from 0 to F = 3FFF %
6      : F3;                    % Single address--Address 6 = F 3%
8      : CF 7E 35;                  % Range of three addresses starting from specific address -- %
END ;                        % Addr[8] = CF, Addr[9] = 7E, Addr[A] = 35 %
```

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

VHDL Description of the complete binary-to-BCD converter.

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

```
-- this circuit converts a 6-bit binary number to a 2-digit BCD representation
--
-- entity name: g00_binary_to_BCD
--
-- Copyright (C) 2014 James Clark
-- Version 1.0
-- Author: James J. Clark; clark@cim.mcgill.ca
-- Date: January 19, 2014

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

library lpm; -- allows use of the Altera library modules
use lpm.lpm_components.all;

entity g00_binary_to_BCD is
 port (   clock : in std_logic; -- to clock the lpm_rom register
          bin   : in unsigned(5 downto 0);
          BCD   : out std_logic_vector(7 downto 0)
      );
end binary_to_BCD;
```

VHDL Description of the *binary_to_BCD* 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 6 bits, the lookup table needs 64 entries to hold results for all possible input cases.

Note that the lpm_rom *address* signal is defined in its component declaration to be of type *std_logic_vector*, whereas the input to the converter circuit is of type *unsigned*. Thus you will have to do a type conversion, as mentioned in the second vhdl lecture.

Show both the ***mif*** file and the ***vhd*** file to the TA.

SIMULATE THE DESIGN

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

To generate the clock signal for the memory register, use the *overwrite clock* item on the Waveform Editor. Set the period to 20nsec. This will load the input register on each rising edge of the clock signal.

This simulation should test all 64 input patterns. Compare the results to the entries in your .mif file. They should match.

Show the TA the results of your simulation.

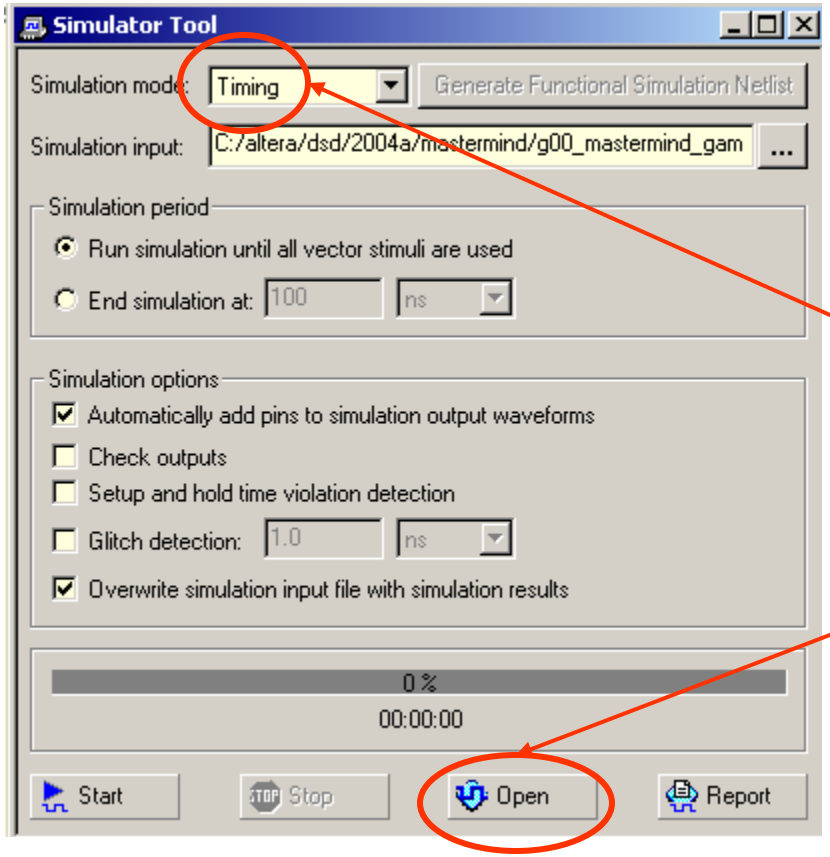
Viewing the Compilation Report

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.

Next, you will do a “Timing” simulation of the *binary_to_BCD* 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 500nsec apart. Use an end time of 2,000nsec (or 2usec). Use a clock signal with a period of 20nsec.

Try the following 4 input values in sequence:

000000, 111111, 101010, 010101

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

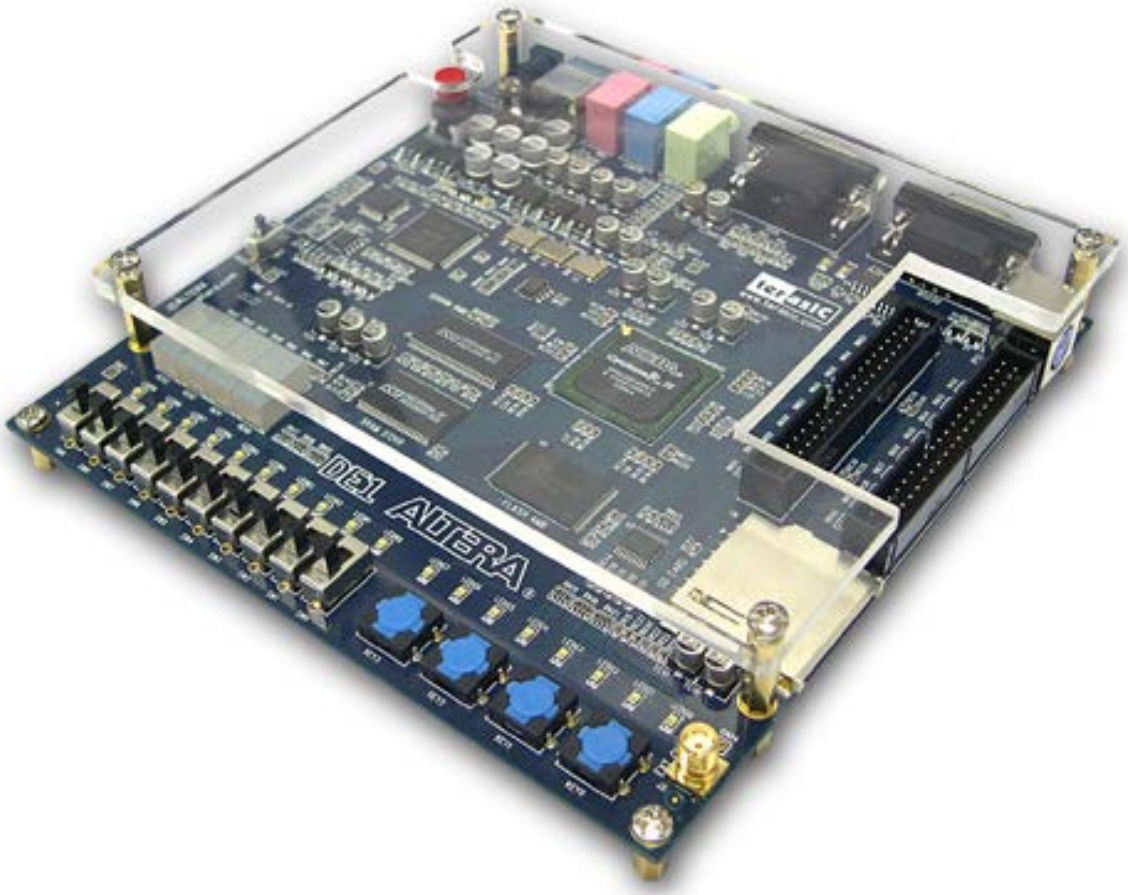
You should be this far (i.e. have completed the lab) at the end of your *second* 2-hour lab period!

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, ***both*** of the group members should go to the ECE department Technician's office, located in room 4140 of the Trottier building**.** Come between 14:00 and 16:00 in the afternoon.

The technicians 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).

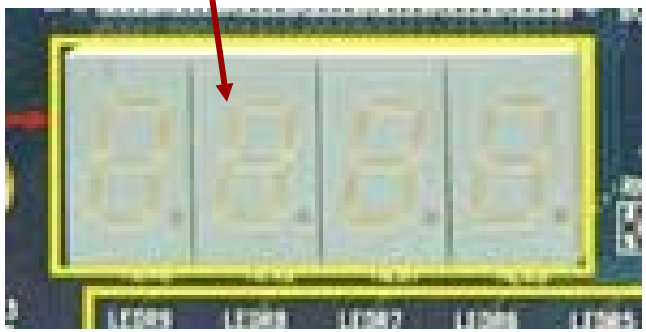
Print out and sign the waiver form (from the WebCT Experiments page) accepting responsibility for the kit. Bring this waiver with you when you go to pick up the lab kit.

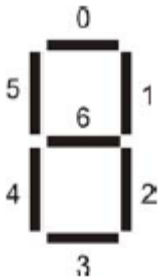
***All* members of the group must be present in order to receive the kits.**

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 $125.

4. Design of the BCD-to-7-Segment decoder .

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 course project.



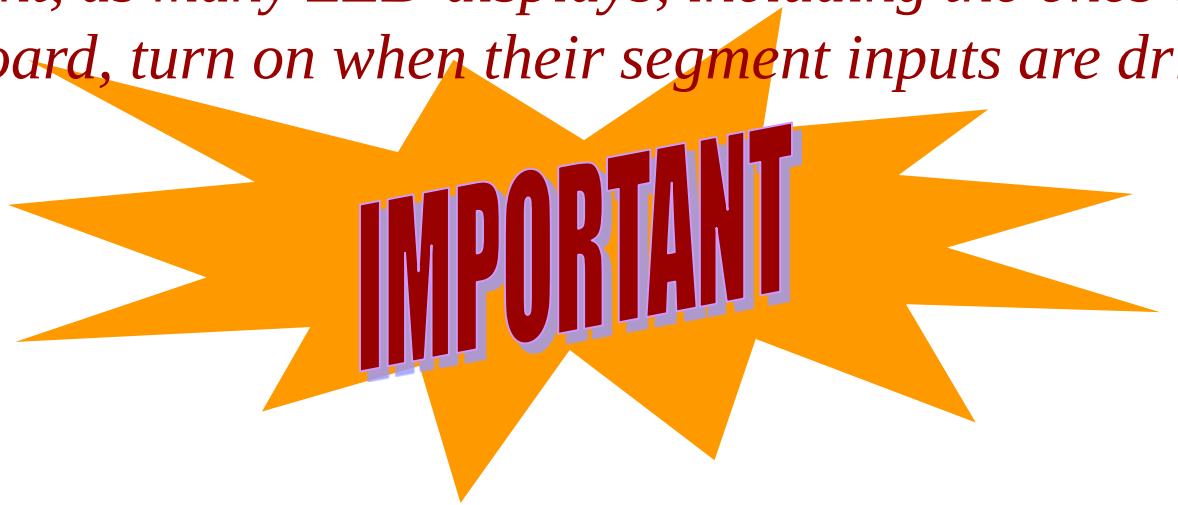


numbering of the LED segments (from the Altera DE1 board manual)

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 binary code representing the 16 hexadecimal digits between 0 and F, and generates the appropriate 7-segment display associated with the input code.

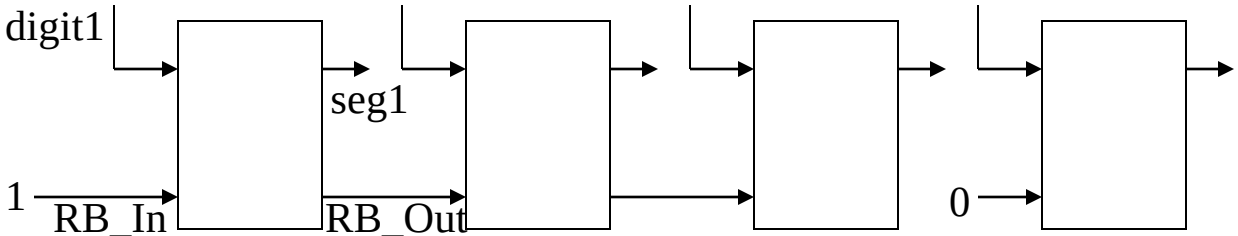
(0 1 2 3 4 5 6 7 8 9 A B C D E F)

*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.*



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. Thus we could display numbers such as **2412** and **7886**. But what about displaying numbers with value less than 1000? If we didn't blank the leading zeroes our display would look like **0095** or **0834**. This looks ugly and unprofessional, so we would rather display **95** and **834** 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, 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.

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(3 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 16 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!

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. Start by reading over the DE1 user's manual, which can be found on the documentation CD provided as part of your lab kit (and also on mycourses).

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.

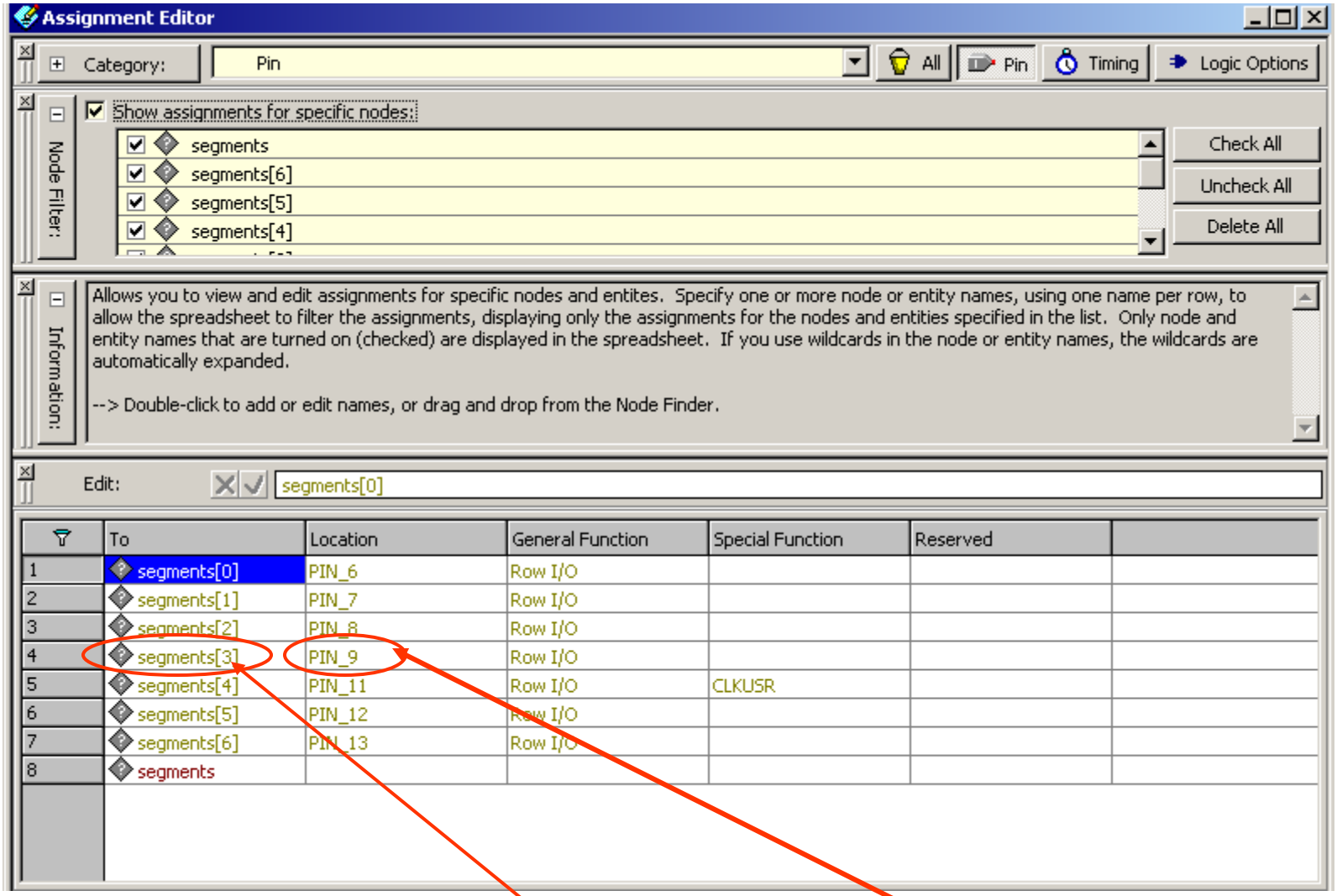
In particular, you will want to connect the LED segment outputs from the instances of the ***gNN_7_segment_decoder*** circuit to the corresponding segments of one of the four 7-segment LED displays on the Altera board.

The mapping of the Altera Board's 7-segment LEDs' segments to the pins on the Cyclone FPGA device is listed in Table 4.4 on page 31 of the DE1 Development and Education Board Users Manual. The pin mappings for the other board components can also be found in section 4 of the manual.

You will also want to connect, for testing purposes, 4 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 ***Assignment Editor***, which can be done by choosing the ***Pins*** item in the ***Assignments*** menu, as shown in the screenshot on the next page.



Enter the pin number for a given circuit node into the Location boxes

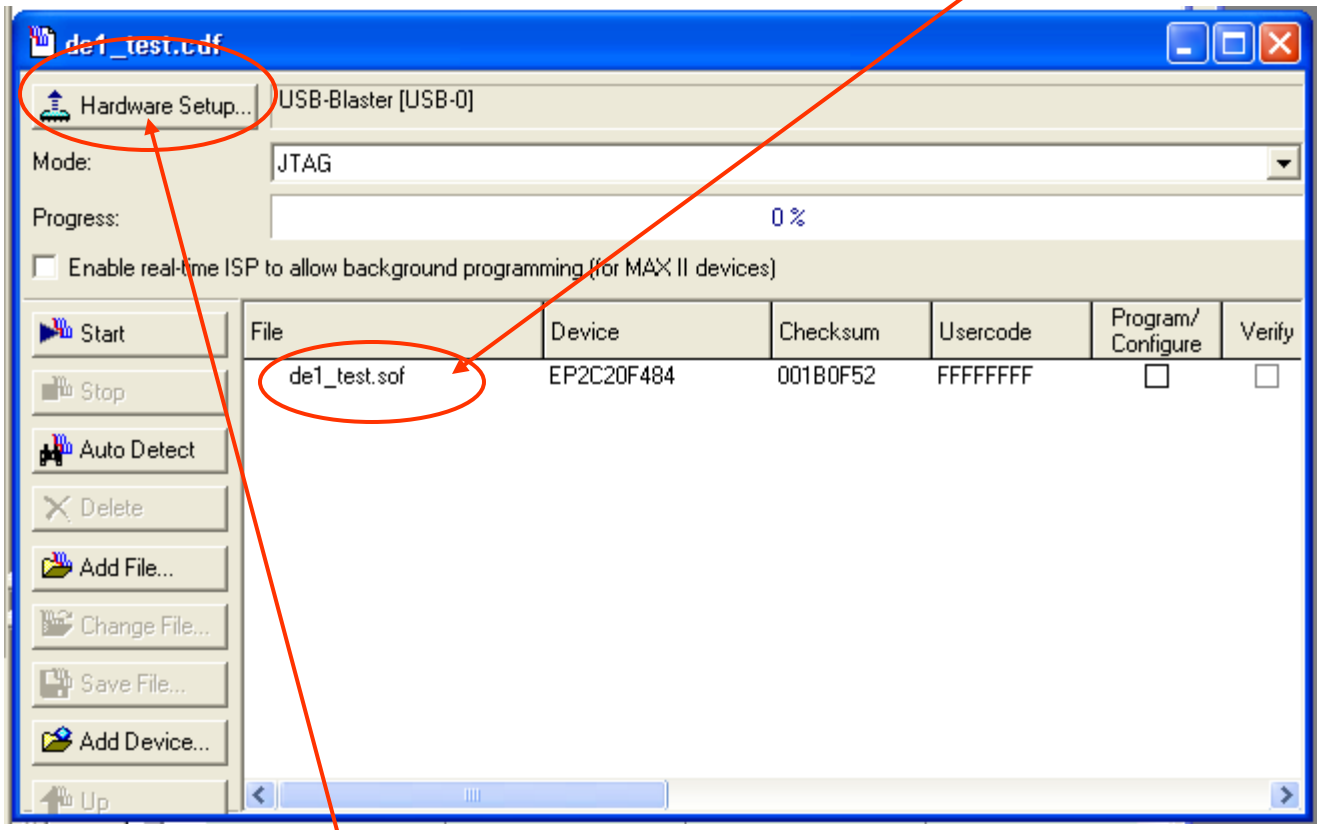
Once you have assigned all of the inputs and outputs of your circuit to appropriate device pins, ***re-compile*** your design.

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.

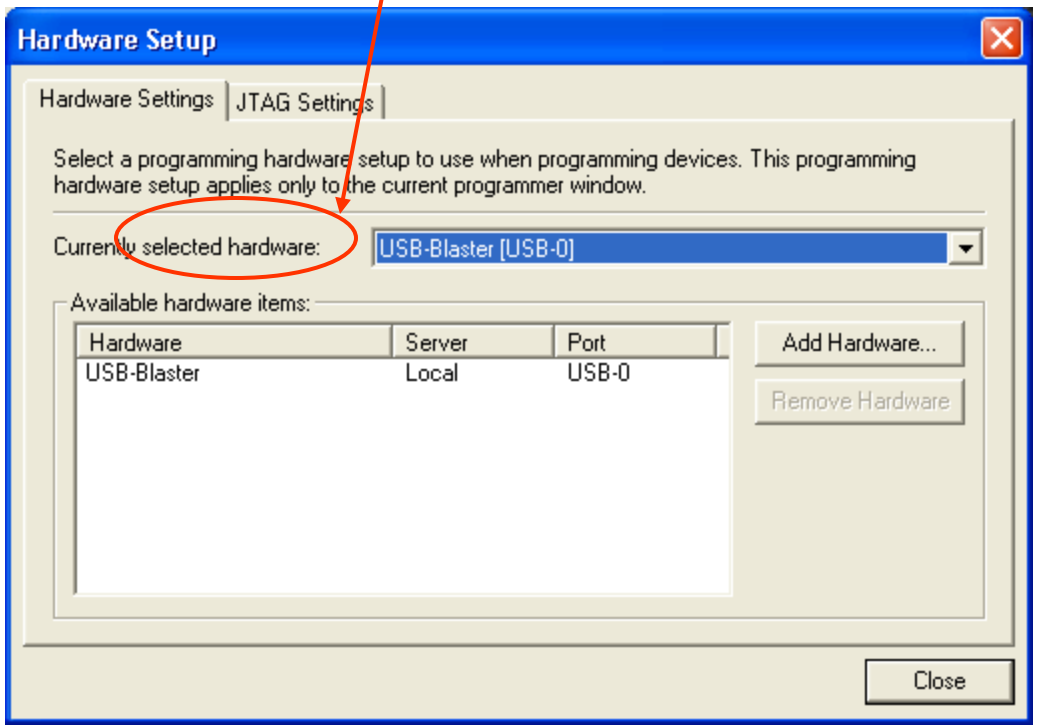
Next select the ***Programmer*** item from the ***Tools*** menu. You should see a window like the one shown below. There should be a .sof (SRAM Output File) file listed. If not, click "Add File".



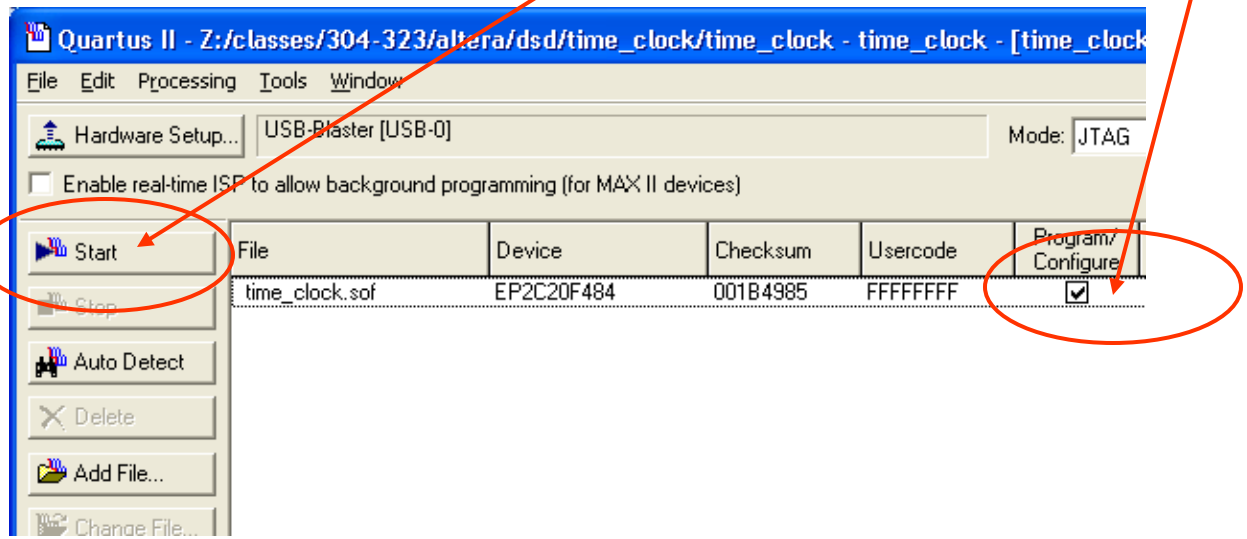
If there is no device visible, click on ***Hardware Setup***

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 all of the 16 different switch settings.

Once you have verified the proper functioning of the LED decoder circuit, write a new VHDL module that connects one instance of it to the output of the *binary_to_BCD* circuit.

Connect the 50Mhz clock on the DE1 board (read the DE1 manual to find the pin assignment for this clock signal) to the clock input of the *binary_to_BCD* circuit. Connect 6 of the switches on the DE1 board to the input of the *binary_to_BCD* circuit. Recompile the circuit and download to the DE1 board. By selecting various inputs through the switch settings, demonstrate to the TA the operation of the *binary_to_BCD* circuit.

TIME CHECK

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

5. Writeup of the Lab Reports .

Write up two (2) short reports, describing each of the ***gNN_binary_to_BCD*** and ***gNN_dayseconds*** circuits. You do not need to provide a report for the LED decoder circuit.

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_binary_to_BCD***) 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 at midnight, Friday February 21.

Grade Sheet for Lab #2

Winter 2014.

Group Number: _____.

Group Member Name: _____. Student Number: _____.

Group Member Name: _____. Student Number: _____.

Marks		
	VHDL code for the *dayseconds* circuit	.
	Functional Simulation of the *dayseconds* circuit	.
	mif file and VHDL code for the *binary_to_BCD* circuit	.
	Functional Simulation of the *binary_to_BCD* circuit	.
	Timing Simulation of the *binary_to_BCD* circuit	.
	VHDL code for the *7_segment_decoder* circuit	.
	Simulation of the *7_segment_decoder* circuit	.
	Testing of the *7_segment_decoder* circuit on the Altera board	.
	Testing of the *binary_to_BCD* circuit on the Altera board	.
		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.