ECSE-323 Digital System Design

Lab #2 – ***Combinational Circuit Design with VHDL*** Fall 2016

Prof. J. Clark

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 a number of combinational circuits needed for implementing the course project.

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

Table of Contents .

This lab consists of the following stages:

1. Design of a 64:6 Encoder circuit.

Use the gNN_16_4_Encoder circuit designed in lab #1 to create a larger 64:6 encoder. The 64:6 encoder should have 64 single bit inputs, with a 6-bit output indicating which of the inputs is high. If more than one of the inputs in high, then the output should indicate the input with the lowest index. There should be a single bit error output which goes high when none of the inputs are high.

Describe the circuit using VHDL, and use four instances of the 16:4 encoder as *components* in the new circuit, plus whatever additional simple assignment statements, selected assignment statements, or conditional assignment statements you think are needed.

VHDL Description of the 64:6 encoder circuit.

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

```
--
-- entity name: g00_64_6_encoder
--
-- Copyright (C) 2016 your name
-- Version 1.0
-- Author: designers' names
-- Date: October ??, 2016

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

entity gNN_64_6_encoder is
 port ( inputs     : in std_logic_vector(63 downto 0);
        CODE       : out std_logic_vector(5 downto 0);
        ERROR      : out std_logic);
end gNN_64_6_encoder;
```

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_64_6_encoder.vhd*. Include it into your project, by checking the "***Add File to Current Project***" box.

First enter the entity declaration from the previous slide. Then fill in the rest of the file with the VHDL architecture body for your *gNN_64_6_encoder* circuit.

Save the file and show your VHDL description to the TA.

SIMULATE THE DESIGN

Using the techniques learned in lab #1, do a ***functional*** simulation of the *gNN_64_6_encoder* circuit. This simulation should test all 64 single bit high cases, as well as a few other (multi-bit high or no-bit high cases) input patterns.

Write the VHDL testbench to generate the circuit inputs and run the simulation using ModelSim.

Show the TA the results of your simulation.

TIME CHECK

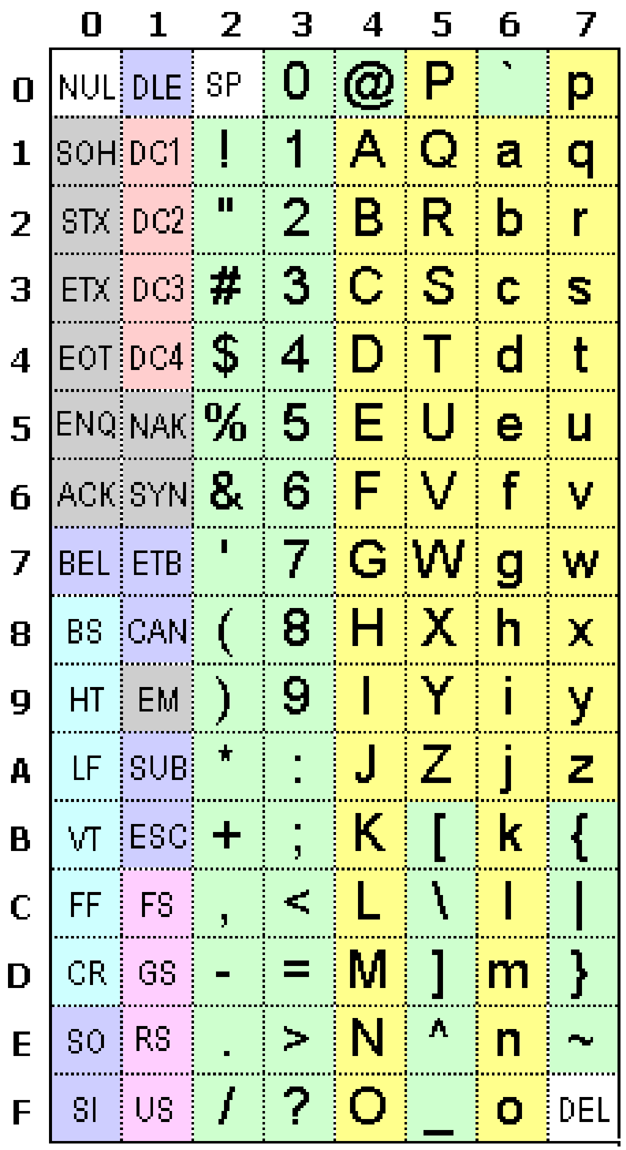
You should be this far (i.e. have completed the lab) at the end of your *first* 2-hour lab period!

2. Design of an ASCII Keyboard Encoder.

One application of 2^N:N encoders of the type you have designed is the implementation of fast keyboard encoders. For example, suppose you have a typewriter-style keyboard with an array of 64 keys, each with a switch that is closed when the key is pressed. Rather than sending all 64 lines to a processor, we can encode the data into 6 lines (or bits), assuming that only one key at a time is pressed.

In this part of the lab you will modify the 64:6 encoder slightly, to provide a different sort of code – the ASCII code. This code is a way of encoding commonly used symbols on a keyboard (letters, numbers, special characters such as ?!# etc.) into a 7-bit code. The coding is shown on the next page.

(Note: typically keyboard encoders are not implemented in this way, instead a matrix arrangement is used to reduce the number of wires needed. But matrix methods are slow(er) and have various issues that need to be dealt with).



	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

The 7-bit ASCII table.

The index along the top row is the 3 most-significant bits of the code, while the index along the left column is the 4 least significant bits of the code (in hexadecimal).

For example, the asci code 7A corresponds to the character “z”.

The two leftmost columns correspond to non-printing codes used to control display or printer operation.

Note that the ascii code for the digits 0 through 9 are equal to 30 hex + digit. For example, the ascii code for the digit 7 is 37 hex. So, to convert a binary digit to its ascii counterpart you would add 30 hex to the number.

Let us assume that the 64 keys on our keyboard include the symbols represented by the 2,3,4 and 5 columns of the ASCII table (e.g. space, !, “, #, … \,],^,_)

Using VHDL describe a circuit that takes in the 64 keyswitch lines and generates the corresponding 7-bit ASCII code. Use the 64:6 encoder circuit you designed in the previous part of the lab as a component in your VHDL description. Note that if more than one key is pressed at a time, you will generate the code for the key with the smallest index.

You can associate the symbols on the keys (e.g. A, B, etc) to the keyswitch line number as you wish. I suggest doing this in a way that makes the ASCII encoding circuit as simple as possible (e.g. just adding 20 hex to the code output by the 64:6 encoder). Do not use any arithmetic operations or modules. Instead, use a large selected signal assignment statement to do the mapping to ASCII.

VHDL Description of the keyboard encoder circuit.

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

```
--
-- entity name: gNN_keyboard_encoder
--
-- Copyright (C) 2016 your name
-- Version 1.0
-- Author: designers' names
-- Date: October ??, 2016

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

entity gNN_keyboard_encoder is
 port ( keys      : in std_logic_vector(63 downto 0);
        ASCII_CODE       : out std_logic_vector(6 downto 0)
          );
end gNN_keyboard_encoder;
```

CREATE THE DESIGN FILE

Once you have written the VHDL description, show your completed VHDL description to your TA.

SIMULATE THE DESIGN

Compile the *gNN_keyboard_encoder* circuit and do a functional simulation. This simulation should test ***a representative set*** of input patterns of the key inputs (at least 16 different key presses).

Write the VHDL testbench to generate the circuit inputs and run the simulation using ModelSim.

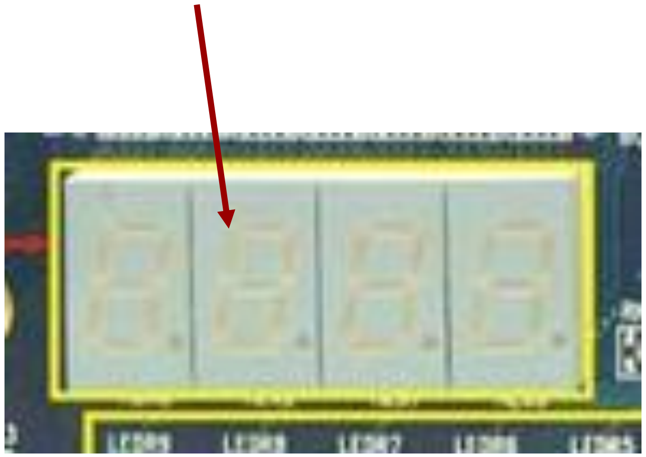
Show the TA the results of your simulation.

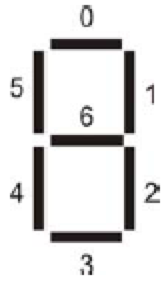
TIME CHECK

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

3. Design of a 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 DE1 board.



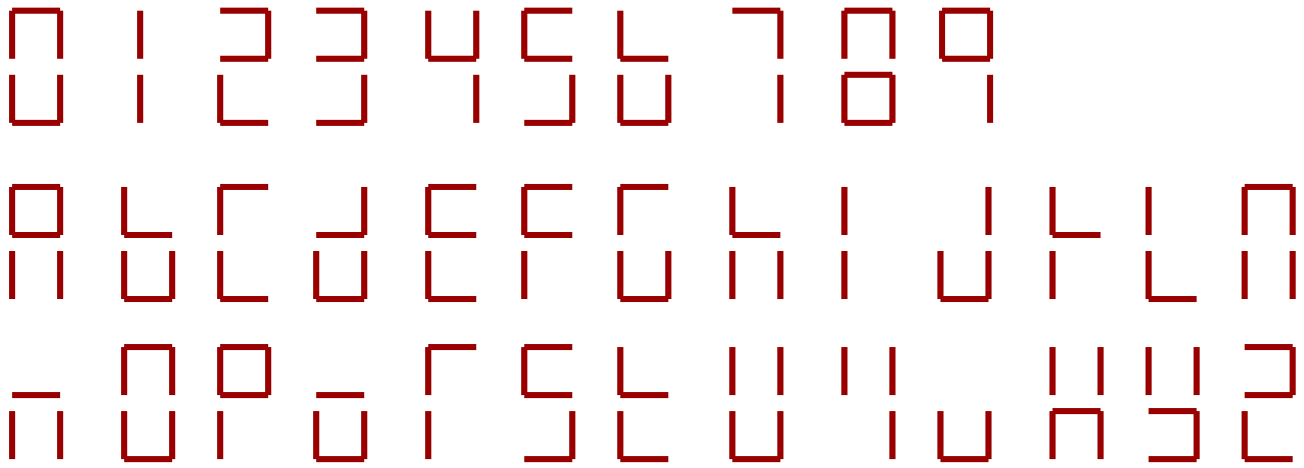


numbering of the LED segments (from the Altera DE1 board manual)

In this part of the lab you will design a circuit that will be used to drive the 7-segment LEDs on the Altera DE1 board. It takes in an 7-bit ASCII code and generates the associated 7-segment display.

Shown below are suggested display patterns for the numbers 0-9 and letters A-Z (use the same pattern for both capital and small letters). The display for all other ASCII characters should be left blank.

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.*



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 ( asci_code : in std_logic_vector(6 downto 0);
        segments : out std_logic_vector(6 downto 0));
end gNN_7_segment_decoder;
```

CREATE THE DESIGN FILE

Once you have written the VHDL description, show the completed VHDL description to your TA.

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 ascii code.

Write the VHDL testbench to generate the circuit inputs and run the simulation using ModelSim.

Show the TA the results of your simulation.

TIME CHECK

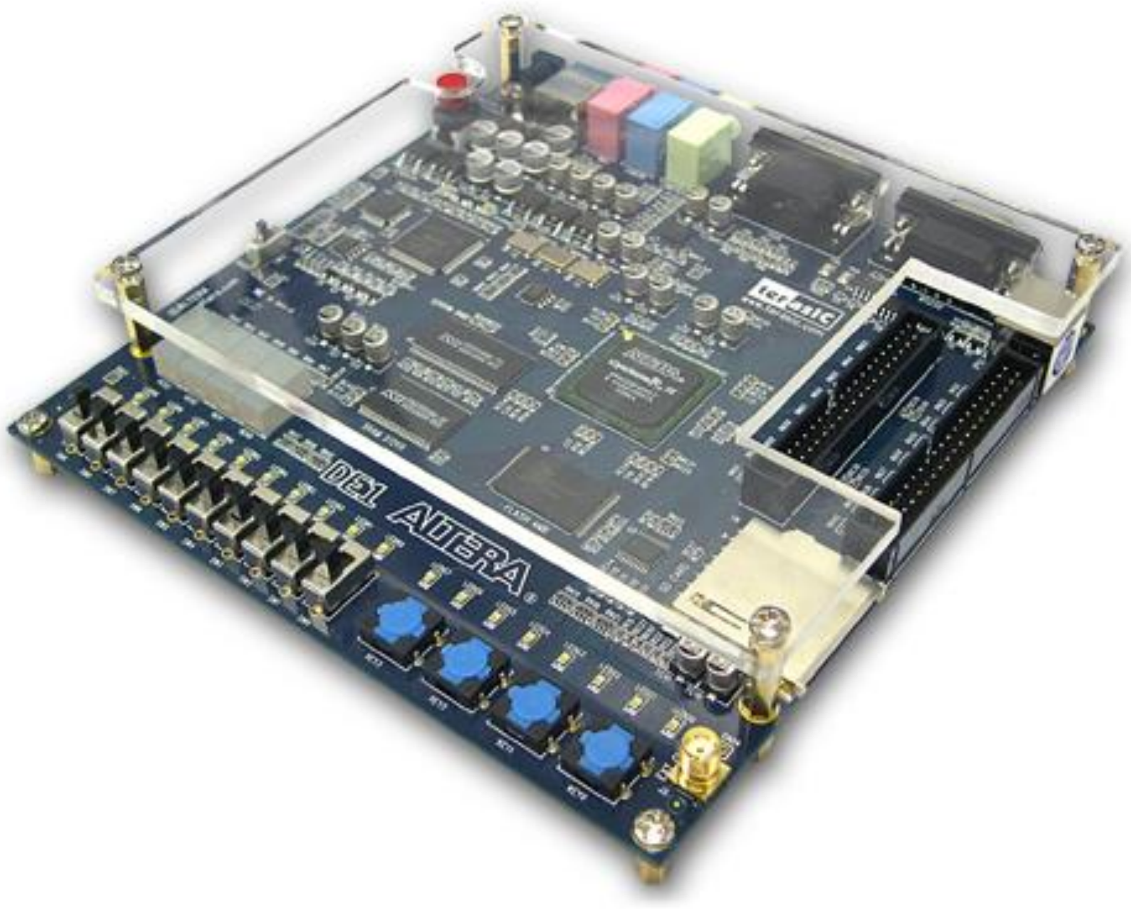
You should be this far (i.e. have completed the lab) at the end of your *third* 2-hour lab period!

2. 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:

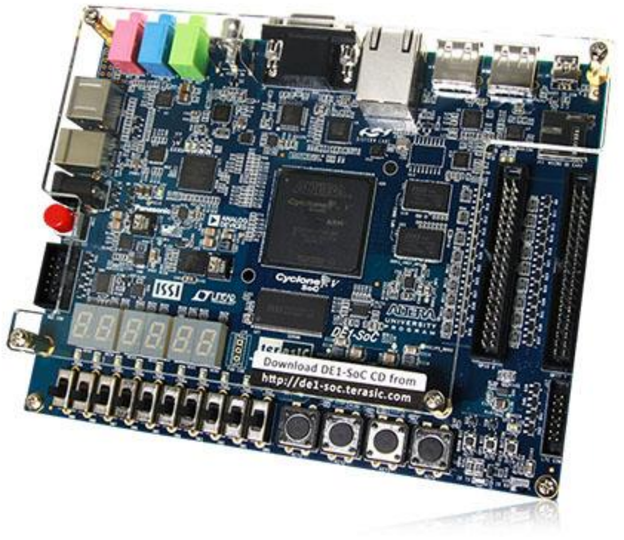
- Altera DE1 Board with a *Cyclone II EP2C20F484C7* FPGA
- Altera DE1 CD-ROM - Version 0.5 with documentation
- Altera Quartus II DVD - Version 7.0
- 1 Power Supply Adapter DC 7.5V/0.8A (US wall plug)
- 1 USB Cable
- 6 Silicon Footstands
- 2 Cables (black- and red-colored)
- 2 PIN Headers, 1P1N

The Altera DE1 Development and Education Board



Some of the lab groups will be given a newer version of the Altera DE1 board, called the DE1-SoC board. This board is essentially the same as the DE1 board, but instead of a Cyclone II FPGA, it has a newer Cyclone V FPGA.

The lab instructions will refer to the DE1 board only, but if you have a DE1-SoC board then there are some differences you will have to take into account.



1. The FPGA device on the DE1-SoC board is the Cyclone V 5CSEMA5F31C6N
2. You must use the Altera Quartus Prime V15.0 (or V15.1) software, which also comes with a newer version of Modelsim
3. Some of the details (such as pin numbers, programming procedure) are different, but you can find the correct values and procedures in the DE1-SoC users manual.

You MUST use the correct Quartus version for your board, i.e. V13.0sp1 for DE1 and V15.x for DE1-SoC

Each group will have their own kit, 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. They 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 accepting responsibility for the kit (to be found on the mycourses lab experiments page). 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 academic price for the Altera DE1 kits is currently USD$127. The academic price for the Altera DE1-SoC kits is currently USD$175.

3. Testing the 7-Segment LED Decoder on the Altera Board .

You will now test the 7-segment decoder circuit you designed earlier. Create, using VHDL, a circuit that connects the LED decoder circuit to the keyboard encoder circuit you designed earlier in the lab.

You will then test this combined circuit on the DE1 board, using the 10 slide switches on the board as some of the keyswitch lines to be input to the keyboard encoder (so you will only be using 10 of the 64 lines in this test).

Compile the test circuit in the Quartus software.

Once you have compiled the LED decoder circuit, it is time to map it onto the target hardware, in this case the Cyclone II chip on the Altera DE1 board. Please read over the DE1 user's manual, which can be found on the myCourses lab experiments page.

Since you will now be working with an actual device, you have to be concerned with which FPGA 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 one instance of the ***gNN_7_segment_decoder*** circuit to the corresponding segments of one of the four 7-segment LED displays on the Altera board.

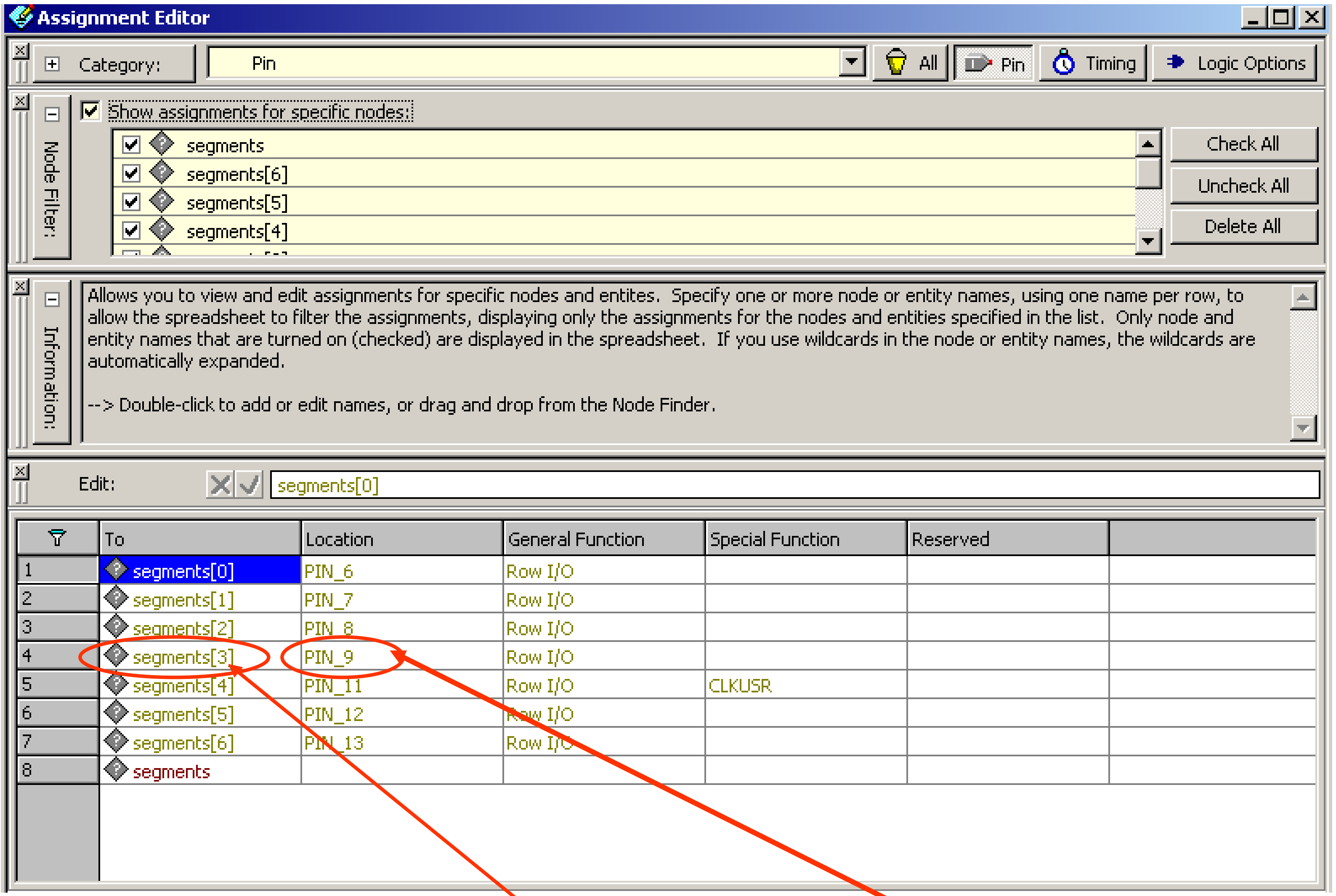
The mapping of the Altera Board's 7-segment LEDs' segments to the pins on the Cyclone FPGA device is listed in Table 4.4 on page 31 of the DE1 Development and Education Board Users Manual. The pin mappings for the other board components can also be found in section 4 of the manual.

You will also want to connect, for testing purposes, the 10 ***slide switches*** on the DE1 board to 10 of the 64 key inputs of the ***gNN_keyboard_encoder*** circuit. It is best for purposes of testing the LEDs to connect the 10 switch lines to the keyboard encoder lines corresponding to letter or number symbols, as otherwise the LED display will be blank.

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.

If you are using the newer DE1-SoC board, remember to check for the correct pin assignments, which will be different than for the DE1 board.



Allows you to view and edit assignments for specific nodes and entites. Specify one or more node or entity names, using one name per row, to allow the spreadsheet to filter the assignments, displaying only the assignments for the nodes and entities specified in the list. Only node and entity names that are turned on (checked) are displayed in the spreadsheet. If you use wildcards in the node or entity names, the wildcards are automatically expanded.

--> Double-click to add or edit names, or drag and drop from the Node Finder.

Edit: segments[0]

	To	Location	General Function	Special Function	Reserved
1	segments[0]	PIN_6	Row I/O		
2	segments[1]	PIN_7	Row I/O		
3	segments[2]	PIN_8	Row I/O		
4	segments[3]	PIN_9	Row I/O		
5	segments[4]	PIN_11	Row I/O	CLKUSR	
6	segments[5]	PIN_12	Row I/O		
7	segments[6]	PIN_13	Row I/O		
8	segments				

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.

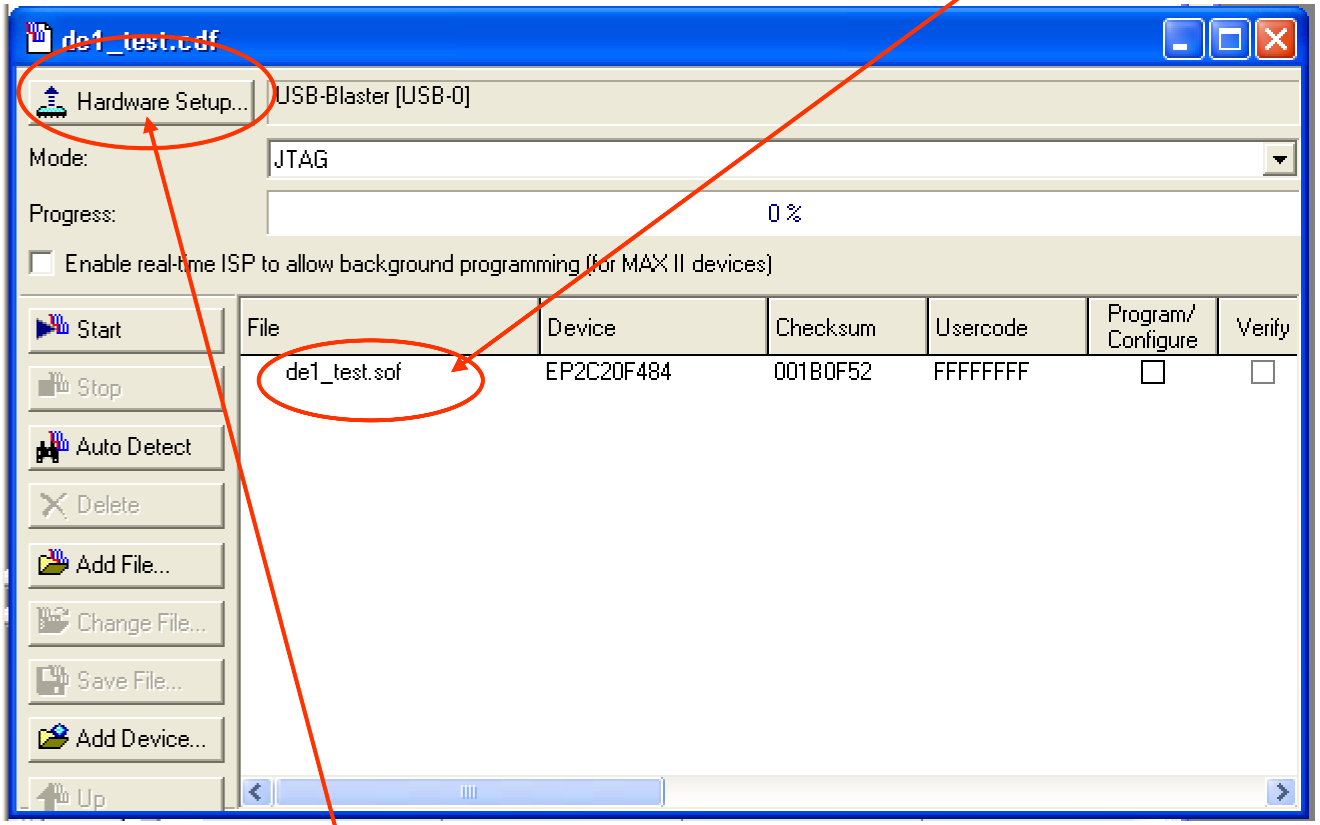
Show your floorplan to the TA.

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.

The programming procedure is a little bit different than what is pictured in these slides for the DE1-SoC board, but you can find the procedure in the DE1-SoC user's manual.

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".



USB-Blaster [USB-0]

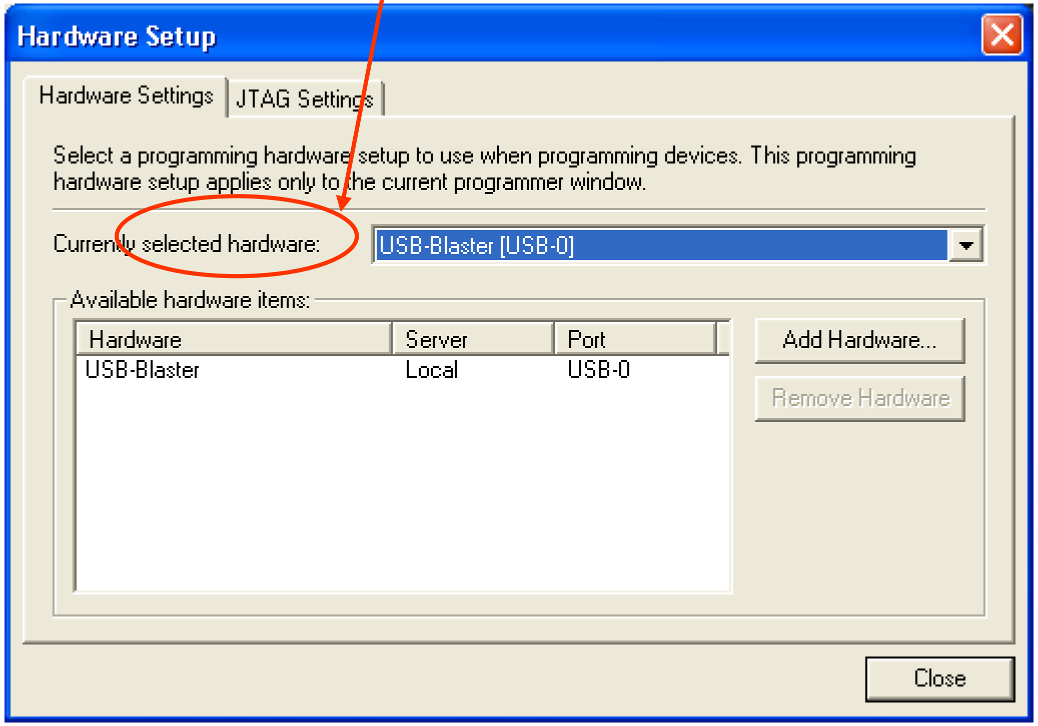
Mode: JTAG

File	Device	Checksum	Usercode	Program/ Configure	Verify
de1_test.sof	EP2C20F484	001B0F52	FFFFFFFF	☐	☐

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.



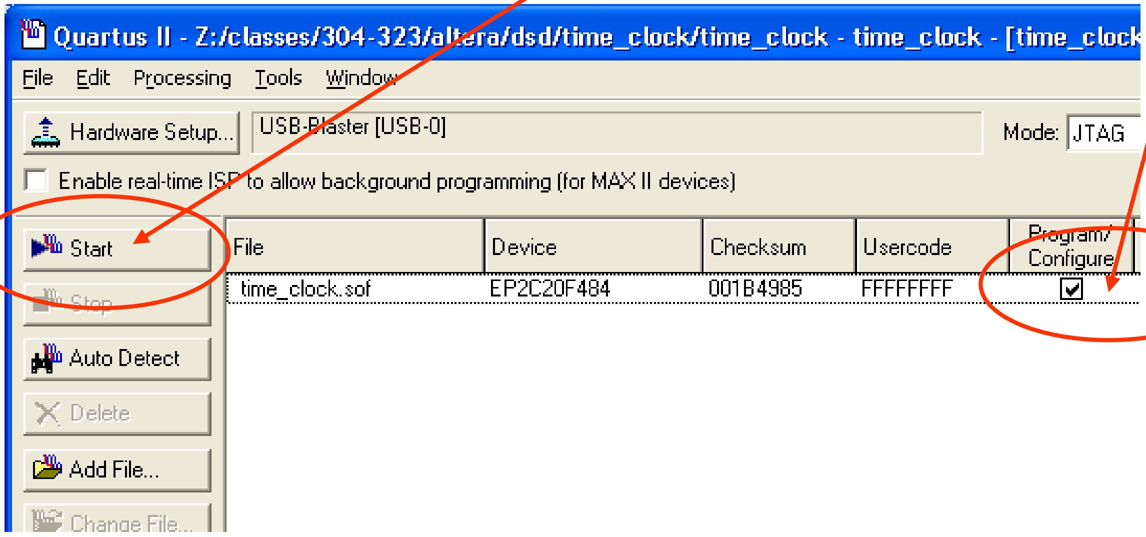
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

Available hardware items:

Hardware	Server	Port
USB-Blaster	Local	USB-0

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.


Quartus II - Z:/classes/304-323/altera/dsd/time_clock/time_clock - time_clock - [time_clock
File Edit Processing Tools Window
Hardware Setup...
Mode:
Start
Stop
Auto Detect
Delete
Add File...
Change File...

USB-Blaster [USB-0]

Mode: JTAG

Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure
time_clock.sof	EP2C20F484	001B4985	FFFFFFFF	✓

Make sure that the displayed symbols correspond correctly to the selected input code.

Demonstrate to the TA that your circuit is functioning properly by going through all 10 of the different switch settings.

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 a short reports, describing the ***gNN_keyboard_encoder*** circuit.

The report must include the following items:

• A header listing the group number, the names and student numbers of each group member.
• A title, giving the name (e.g. ***gNN_keyboard_encoder***) 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 including the testbench (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 (including the testing on the DE1 board using the LED display), showing representative simulation plots, and detailing what test cases were used.
• It is recommended that a scan or photograph of the signed gradesheet be attached to the report document.

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 11:59 PM, Friday, October 21.

Grade Sheet for Lab #2 Fall 2016.

Group Number:________________.

Group Member Name:________________. Student Number:________________.

Group Member Name:________________. Student Number:________________.

Marks			TA Signatures
	1.	VHDL code for the *64:6 encoder* circuit	.
	2.	Simulation of the *64:6 encoder* circuit	.
	3.	VHDL code for the *keyboard encoder* circuit	.
	4.	Simulation of the *keyboard encoder* circuit	.
	5.	VHDL code for the *7_segment_decoder* circuit	.
	6.	Simulation of the *7_segment_decoder* circuit	.
	7.	Testing of the *7_segment_decoder* circuit on the DE1 board	.

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.