Experiment 28 - Digital Electronics with Altera
Part II: The Practical Part

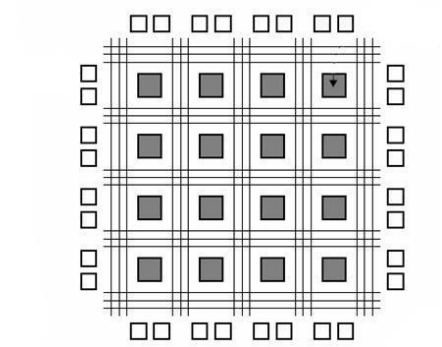
Department of Electrical Engineering & Electronics

April 2022, Ver. 8.2

Experiment Specifications

Module(s)	ELEC211
Experiment code	28
Semester	2
Level	5
Lab location	Allocated lab, or if working remotely, home PC or laptop. Ideally Windows – see specific instructions for MAC OSX.
Work	**Individual.**
Timetabled time	7 hours (1 lab day). Allocate further time to write your report. Study the PreLab material and attempt the test questions first (even if you do not need to submit the test yet).
Subject(s) of relevance	Digital Logic Design, FPGAs and Verilog
Assessment method	Online submission via Canvas using the provided MS Word submission template. See the **Instructions** section. ***Advice: take screenshots so you can write the report as you go.*** Standard UoL penalties on plagiarism (inc close paraphrasing) and collusion apply.
Submission deadline	See Canvas for details; usually 1 week after your assigned lab day.

Important: Marking of all coursework is **anonymous**. Do not include your name, student ID number, group number, email or any other personal information in your report or in the name of the file submitted via Canvas. A penalty will be applied to submissions that do not meet this requirement.



Instructions:

Before you begin

- Read **this script** carefully through before attempting the experiment.
- Other supporting material is on Canvas to guide you, both in ELEC211 and in your labs module (ELEC222/224/273). From the latter, see **Exp 28 PreLab** and **Exp 28 Supporting Material** as well as the **Exp 28 Submission Template**.
- **Before starting, you should read all of the above documents carefully, and also make sure you are up to date on ELEC211 Canvas Lectures 1-15.**
- **Attempt the Pre-Lab Questions on Canvas, worth 30%, before this experiment. (However, you may not necessarily need to submit the Test before the lab.)**
- To submit your Report, download the Exp 28 Submission Template from Canvas. Follow the instructions there, fill in the required information and upload to Canvas. See detailed guidance on the Submission Template about keeping it anonymous.
- Results should be submitted as screenshots, photos and other information. Therefore, keep a copy of all the required information, results, screenshots, photos etc. while carrying out the experiment, to use for your submission on Canvas. Don't leave it 'until later' to get the evidence you need – do it as you go. **See the Submission Template to find out in advance which pieces of evidence you will need.**
- Make sure all screenshots are of sufficient resolution to be clear/readable. Marks will not be awarded if the screenshots are unclear. Follow the detailed guidance below – two images are required for each screenshot. On the full-screen screenshots in your Appendix, your personalised filenames should be readable on 100% zoom or similar.

Marking scheme highlights (see Submission template for full details):

1.	The pre-lab test	[30 Marks]
2.	The schematic, including the compilation result, of Section 2.3	[3 Marks]
3.	The output simulation waveform of Section 2.4	[3 Marks]
4.	The schematic ***YI_7SegmentDecoder.bdf*** of Section 2.5	[4 Marks]
5.	The simulation waveform of Section 2.6	[4 Marks]
6.	The schematics (inc. ***YI_dec_counter.bdf***) of Section 3.1	[4 Marks]
7.	The simulation waveforms of Section 3.1	[4 Marks]
8.	The initial schematic of Section 3.2	[4 Marks]
9.	The simulation waveform and explanation of Section 3.2	[4 Marks]
10.	The schematics and simulations of the divide-by-12, and of the cascading of two divide-by-12 counters, of Section 3.2	[10 Marks]
11.	The code and annotated simulation of Section 4.1	[10 Marks]
12.	The schematic and annotated simulation of Section 4.2	[10 Marks]
13.	The schematic and annotated simulation of Section 4.3	[10 Marks]

Important

- All **code** should be included as **text** and **not** screenshots.
- **Throughout this assignment, please insert your last-letter initials (see p.9) at the beginning of all saved filenames. This should be readable from the title of the relevant window. This instruction is implied below with "YI" for "your initials".** If this format is not followed, corresponding sections may receive a mark of zero.
- For **every snapshot** from the screen (whether of schematics or simulations):
  - **FIRSTLY,** use **PrintScreen** under MS Windows, or CMD+SHIFT+3 or **Screenshot** under macOS, to show the entire desktop including time and date. Dump these full-screen screenshots at the end of your Report in an Appendix, as evidence that the work done is uniquely your own.
  - **SECONDLY,** use the **Snipping tool** (MS Windows) or **Screenshot tool**(macOS) to show the relevant part of your design or simulation. This 'focused' screenshot is the one you should include in the appropriate section of your Report.
  - If the entire desktop, including taskbar + time + date + initialled filename is not visible in a screenshot in the Appendix, the associated 'focused' screenshot in your main Report may be ignored, with that section receiving a mark of zero.
- Including screenshots/photos of other people's work is academic malpractice and will be penalised in accordance with the University Codes of Practice on Assessment.

Preamble: Two ways to do *extra* Quartus II work outside the lab

This section is for students who may wish to undertake further work with Quartus II either before or after the lab. The main work still needs to be done in the lab, where DE1 boards are available.

To work on this experiment outside the lab, you are encouraged to download and install **Quartus II 13.0sp1 Web Edition** on your machine. This is not a requirement but you may find it helpful. Nevertheless, you should aim to complete all the main work in the lab as you will not have access to the DE1 development board outside the lab.

If you want to work at home before the lab, it would be a good idea to save your project folders and files onto removable storage and then transfer them to your M:/ drive in the lab, or vice-versa if you want to continue some software work at home after the lab. Don't forget, too, that your M:/ drive is accessible in library PCs, etc. Finally, see note at the end of this section about the Remote Teaching Centre Service.

1. Preparing for download / installation

Students running mac OS / Mac OSX

Quartus II is software that runs under Microsoft Windows OS. If you run a macOS /Mac OS X, you will need to first install a Microsoft Windows OS emulator calledCrossOver. This emulator is provided by the CodeWeavers company; a trial versionlasting 14 days is available and it can be downloaded from: https://www.codeweavers.com/products/crossover-mac/download .

For downloading CrossOver, you are required to insert your name and email address.This will enable the "Download Trial Now" button. After completing the form and clicking on the button, the browser should automatically ask you to download the "crossover-19.0.1.zip" file. Save the zip file to a location of your choice, then extract itand copy the "CrossOver" program into the "Applications" folder. Launch CrossOver, an alert message should open as shown in Figure B1. Make sure that CrossOver wasdownloaded from "www.codeweavers.com".

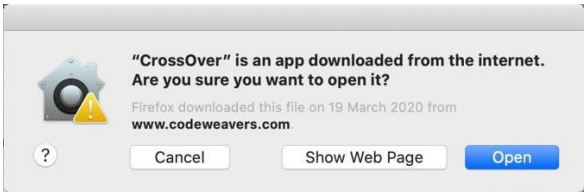


Figure B1. macOS security feature notifying the user that CrossOver has been downloaded from the Internet

Click "Open" to continue and another message should open to give you the option tolaunch CrossOver in different versions (Figure B2); click "Try now" to run the trial version. This will launch CrossOver in trial mode and another dialog window should open asking you if the CodeWeavers company can log statistics about the usage of the application, click on the corresponding button depending on your personal preference. You now have CrossOver correctly installed and ready for use.

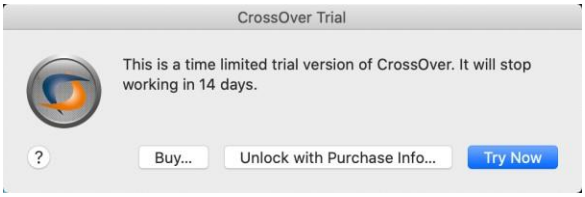


Figure B2. CrossOver trial

***All students** (including those running mac OS / Mac OSX)*

Visit https://www.intel.com/content/www/us/en/software-kit/711791/intel-quartus-ii-web-edition-design-software-version-13-0sp1-for-windows.html and ensure that the selection at the top of the screen is as shown in Figure B3. If you are running Linux, please instead visit: https://www.intel.com/content/www/us/en/software-kit/711790/intel-quartus-ii-web-edition-design-software-version-13-0sp1-for-linux.html and you will see a similar screen.

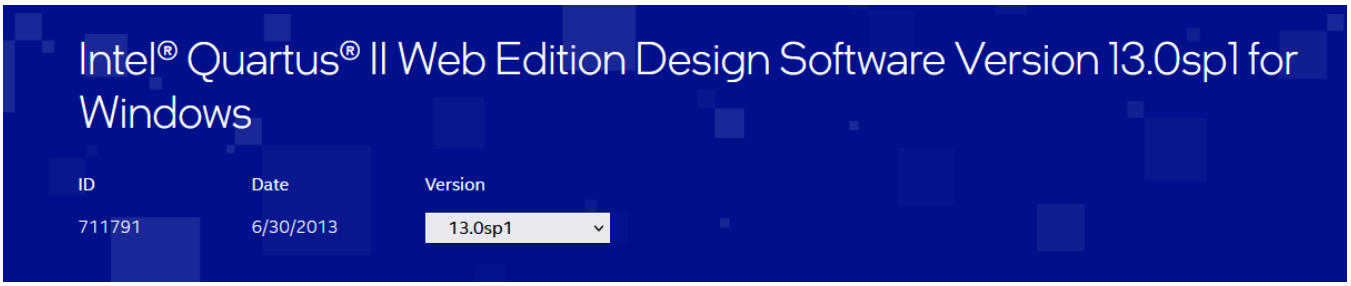


Figure B3. Correct selection for Quartus II Web Edition download, for those using a Windows PC or emulating Windows via CrossOver on a MAC.

Make sure to select "Individual Files" as shown in Figure B4, and click to download the highlighted files ***but please read the next page of this document before proceeding further***. Note: MAC users: in CrossOver, you may find that the files will only install on a Win7 bottle withinthe emulator – this is OK. Linux users – obviously the file extensions will be different, but note also that confusingly, on the Linux version of the page the QuartusSetupWeb executable appears **below** the ModelSimSetup executable (not above it as in Fig. B4).

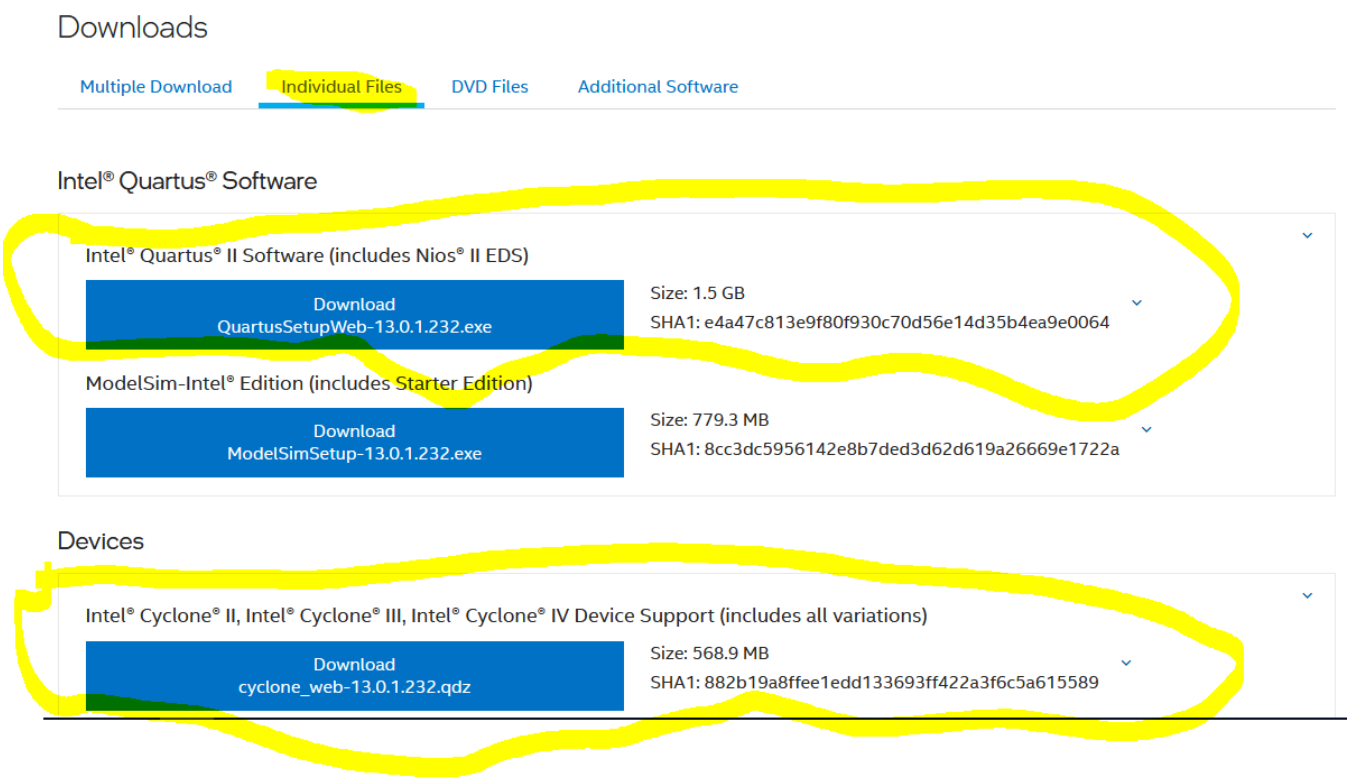


Figure B4. Minimum files for a functioning download of Quartus II Web Edition.

2. Download/installation of Quartus II including Cyclone device support

Important – Intel Software License Agreement / Legal Disclaimer / Security issue

You probably noticed a yellow cautionary banner at the top of the previous webpage, and furthermore, when you try to download the files highlighted above, Intel will ask you to agree to a software license agreement / legal disclaimer. **It is your choice whether or not to do so.** We signpost this opportunity to download a local version of Quartus II Web Edition 13.0 sp1 so as to provide you with a bonus resource to use outside the lab for extra preparation and any work you choose to continue with afterwards. **We have not encountered problems with this software, but we make no guarantees – download and install is at your own risk.**

Importantly, on the previous webpage (have a look) Intel mention a specific security vulnerability as follows: "You may be exposed to a vulnerability issue if you have installed or plan to install Intel® Quartus® II Design Software from version 11.0 to version 18.0 to a location with space(s) in the path. See this KDB solution for more details." Unsurprisingly, one solution to this particular vulnerability is to ensure that you install to a location without any spaces in the path – see the solution link they provide for more details.

Intel have provided some technical recommendations (follow link) for those who choose to go ahead. One of them is to only download the software components you need – one good reason to only download the two components mentioned on the previous page in this document.

You are also welcome to click the link in the yellow banner at the top of the previous webpage, to download version 13.1 instead – let us know how you get on – but at the time of writing we cannot confirm whether or how this improves the situation; again, it's your choice given the information provided by Intel, and they will still ask you to sign the agreement / disclaimer.

If you are at all concerned, we suggest, instead, trying to use Quartus II installed on a University PC, via remote connection with the Remote Teaching Centre Service (see next page). Contact Dr McIntosh (dmc@liverpool.ac.uk) if you have any queries.

Having read the above and visited the links, if you are personally happy to sign the disclaimer, continue to download the free Web edition of Quartus II, and of Cyclone. Please be aware that these installation files together total 2.1 GB and depending on your internet connection speed, could take anything up to a few hours to download! Please be patient.

Installing Quartus II Web Edition – do not ignore this step!

Now, install the software by opening the first file (QuartusSetupWeb-13.0.1.232 or similar) and following the instructions. **Don't forget to follow the important instruction above about the installation location path (make sure there are no spaces in the path).** [Note that the fully installed program, with device support, will take around 5.9 GB of disk space which you should ensure is available.] When installation of Quartus II is complete, **you may be prompted to 'install devicesupport' or a similar message**, if this has not been automatically done. This leads into installation of the Cyclone device support which is necessary and important.

Continuing to install Cyclone device support (if not auto-installed)

Take the above option and follow the wizard. For example, on a Windows 10 PC you will probably see a dialogue box asking if your .qdz files (for the Cyclone) are to be found in C:\altera\13.0sp1\quartus\bin. The file (**cyclone_web- 13.0.1.232.qdz**) was downloaded earlier.

Whichever location is listed in the dialogue box prompt, cut and paste the .qdz file from where it was downloaded / unzipped, to the prompted location, beforecontinuing. Then, continue to follow the instructions in the wizard. Make sure to check'Devices' and 'Cyclone II/III/IV'.

If at some point the options are greyed out and you see an error message, this can mean that the Cyclone device support has already been installed.

3. Remote Teaching Centre Service – 2nd way to access Quartus II off-campus

As noted above, you can access your M:/ drive from most University MWS PCs e.g. library PCs. A number of UoL PCs also have Quartus II installed. If you struggle with download/installation to your own machine, you can if you wish try the Remote Teaching Centre Service (RTCS); after login, select 'work in my own time' and under 'Choose an application', find 'Quartus'. Follow the RTCS instructions (mostly on the main RTCS pages but there is also a guide here). Note: if you're using a Linux system, try these instructions for the final step of getting the remote desktop connection (instead of opening the linked Remote Desktop Connection file).

Please bear in mind that (a) a suitable PC with Quartus may not be available if too many students are trying to use one at a given time; (b) when compiling, etc., Quartus II may operate more slowly on some University PCs depending on their capabilities etc. than a local lightweight installation to your own computer might; (c) save your work frequently (to your M:/ drive, NOT to the default which will probably be the C:/ drive of the remote computer – you can't guarantee access to this again!) in case of sudden unexpected disconnection from the remote PC and (d) except a little lag occasionally over the remote connection.

For support, questions or general queries with any of the above, please contact Dr McIntosh in the first instance (dmc@liverpool.ac.uk) and I will be happy to try and support you and/or respond to your questions/queries.

The Experiment

1. Introduction

This experiment will take you through a tutorial on how to use the *Altera Quartus II* FPGA (Field Programmable Gate Array) development package and (on campus) application on the **DE1** development board (Figure 1).

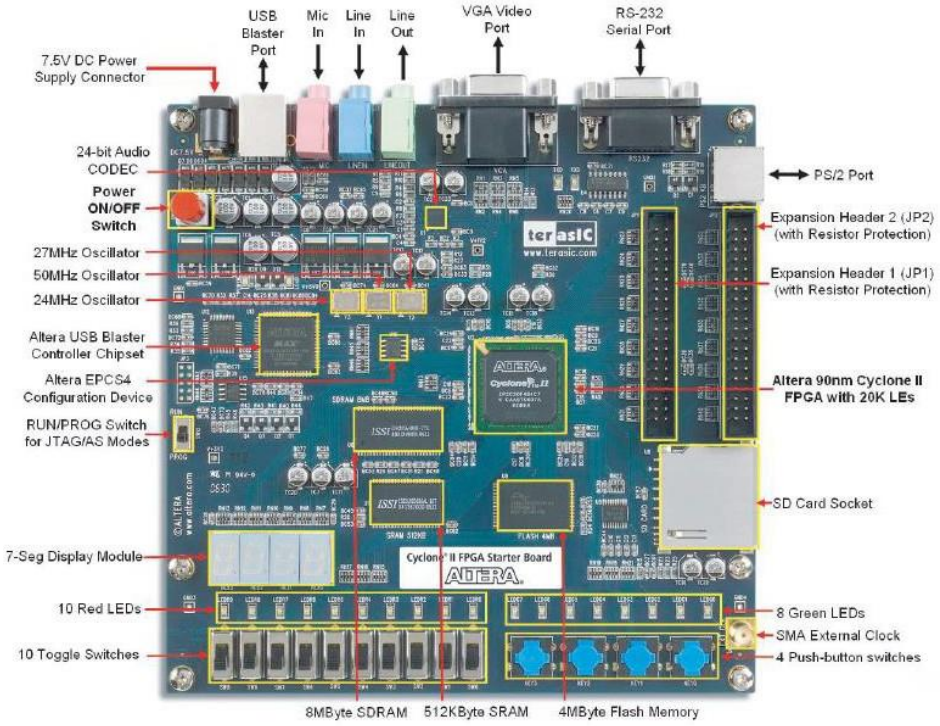


Figure 1. Altera DE1 Board

Structure

You must complete the **PreLab** on Canvas, & read this script and the **supporting documents** to familiarise you with each stage of the experiment, before you begin your work. Also, ensure you are up-to-date with all relevant ELEC211 lectures / problem classes on Canvas.

After inputting your design, **three steps** are required following any project:

- **Compile** the project to see whether there is/are any error(s).
- **Simulate** the project based on some known input, to check output is correct.
- **Test** your project by programming the DE1 board.

This experiment is built around learning how to design, and then following the above three steps within Quartus II. Here is an approximate overview of the experiment (not including pre-lab test):

- Part 2: Project Design based on Schematic Capture
  - Project setup / Schematic Capture
  - Compilation
  - Simulation
  - Programming the board
  - Decoder design (based on Verilog) [combinational design]
- Part 3: Sequential design
  - Counter design (based on Verilog)
- Part 4: Design Assignments
  1. Single Pulser
  2. Pushbutton-Enabled Counter
  3. Adder/Subtractor logic

At some stages during the experiment you will need to refer to the '**Supporting Material**' and **PreLab** documents on Canvas for further information and guidance. Use your initiative!

2. Project Design based on Schematic Capture [14 Marks]

Schematic capture is one of the most common methods for entering a design. It involves using a design tool to draw a schematic of the desired circuit. This is probably the easiest method of design to understand, but it is not necessarily the most efficient.

Quartus II software includes schematic symbols and models for the most common 74-series TTL IC's. This means that a designer can easily take a schematic or existing design of a circuit implemented using TTL parts and enter that schematic into the software. The entire circuit, which may contain dozens of TTL chips on a printed circuit board, can then be compiled into a single FPGA package.

Although very good for updating old projects, schematic capture is not usually the most efficient way of capturing the behaviour of new designs. For that purpose, a Hardware Description Language (HDL) is probably the more efficient. We will cover a popular HDL called Verilog in a later section of this experiment.

2.1. Start the Altera software

While in the lab, you must work on the lab PC. This will give you the best overall functionality, ensure you can program the DE1 board without problems, and allow us to support you most effectively. On the lab PC, select **Start → All Programs → Altera 13.0.1.232 → Quartus II 13.0sp1**[1]. The Quartus II Manager window will open (Figure 2). This window allows you to access all of the different tools that you will be using from one place. (If the "**Getting Started with Quartus II**" window appears, close it by clicking the "x" at the top right).

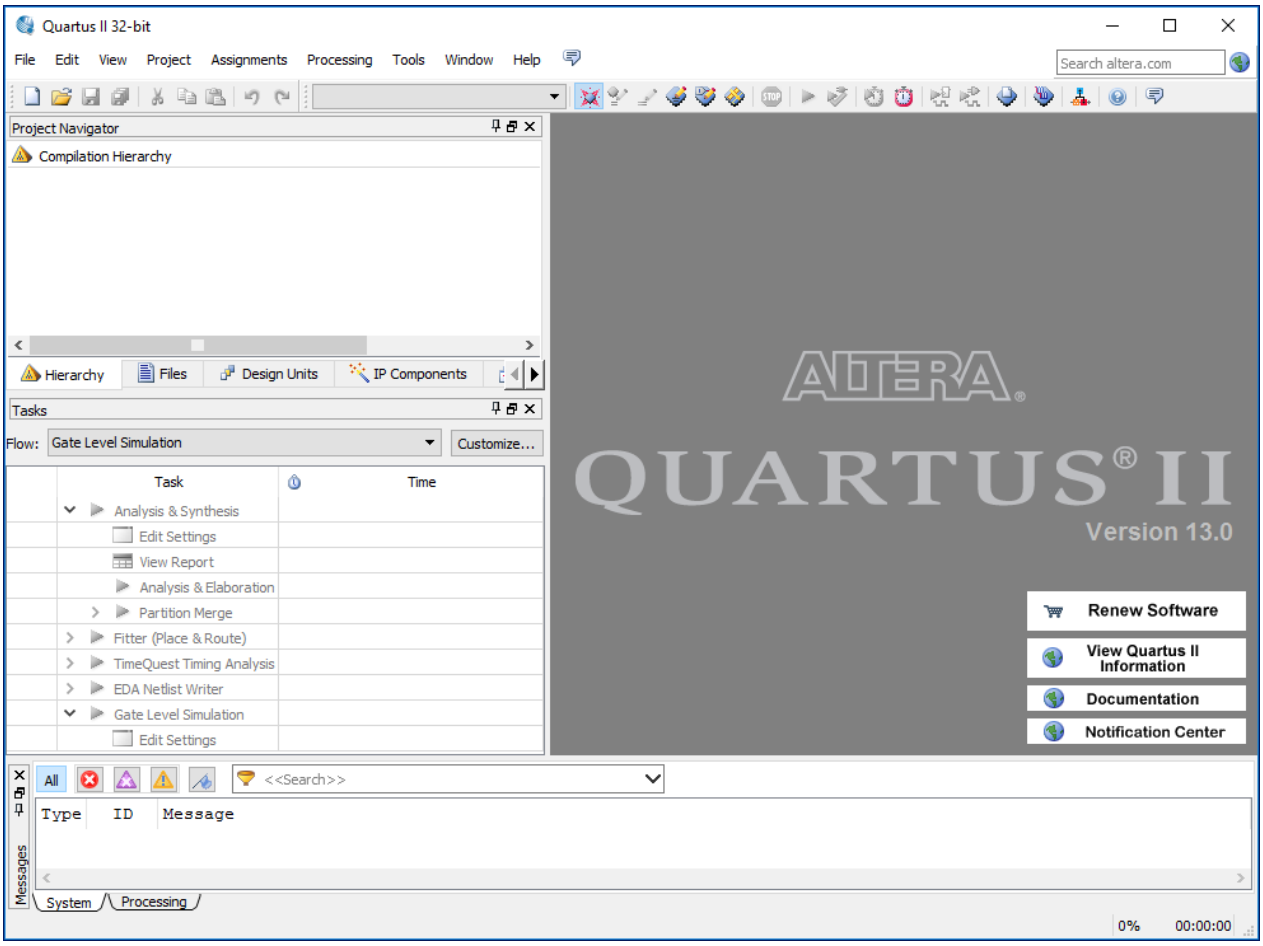


Figure 2. Quartus II Manager Window

[1] (If doing additional work at home, this may be 32-bit or 64-bit depending on your appropriate download above.)

The tool bar along the top is basically the same for all applications. Some icons will be greyed out, depending on which windows are in the foreground. Some applications will have their own tool bars, but these will be along the left side of the window.

Select **File → New...** Select **Block Diagram/Schematic File** (Figure 3)**.** Click **OK.**

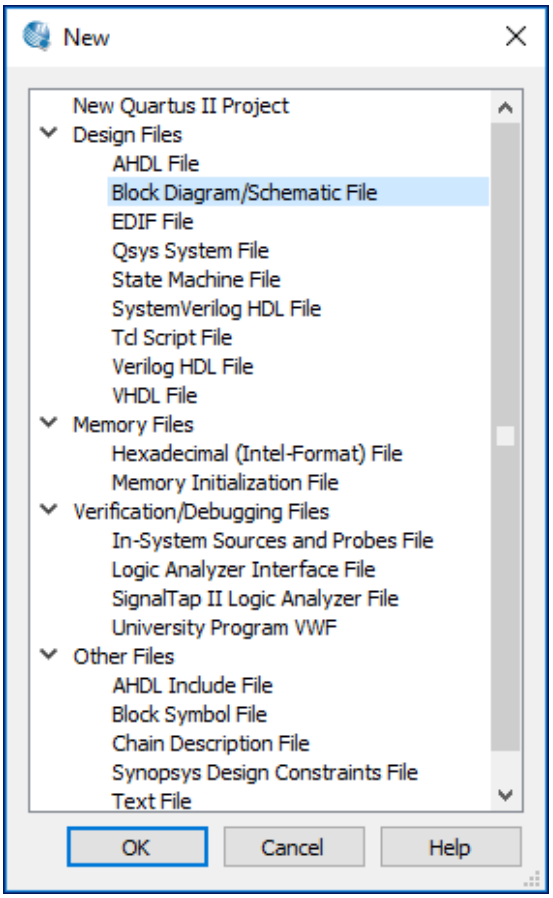


Figure 3. New item selection window

The Schematic Editor window will open. Select **File->Save As but do not save the file yet!** Select your 'M' drive in the 'Drives:' dialog. **IMPORTANT: SELECT YOUR M:/ DRIVE ITSELF AS THE PARENT FOLDER, NOT M:/SOME_FOLDER_NAME/SOME_FOLDER_NAME, FOR EXAMPLE, AND NOT SOME OTHER DEFAULT LOCATION**[2] (not, for example, 'C:/users/…'; the latter is just a local profile for your username on the specific PC you are using, whereas you can access your M:/ drive from most MWS computers on campus.)

On your M:/ drive, while still within the Save As dialogue, create a folder with the name "Exp28", and in that folder, create another folder called "**YI_dec7448**". From this folder, enter **YI_dec7448.bdf** in the *File Name* field, where in both cases **YI**stands for **your last-letter initials**. E.g. Joseph Bloggs would save HS_dec7448.bdf, in a project folder called HS_dec7448, in a parent folder called Exp28 on the M:/ drive. In the *Save as type* field choose Block Diagram/Schematic Files (*.bdf). Click ***Save*** **(**Figure 4**). Ensure that 'Create new project based on this file' is ticked.**

Please follow the precise file naming instructions carefully in this Experiment; they are important for a number of reasons, as explained elsewhere.

[2] (Of course, if/when doing additional work from home on a local version of Quartus II on own machine, you won't be selecting your M: drive, but follow the other instructions as closely as possible to satisfy project requirements and ensure the most painless transfer between your own computer and a lab PC or vice versa)

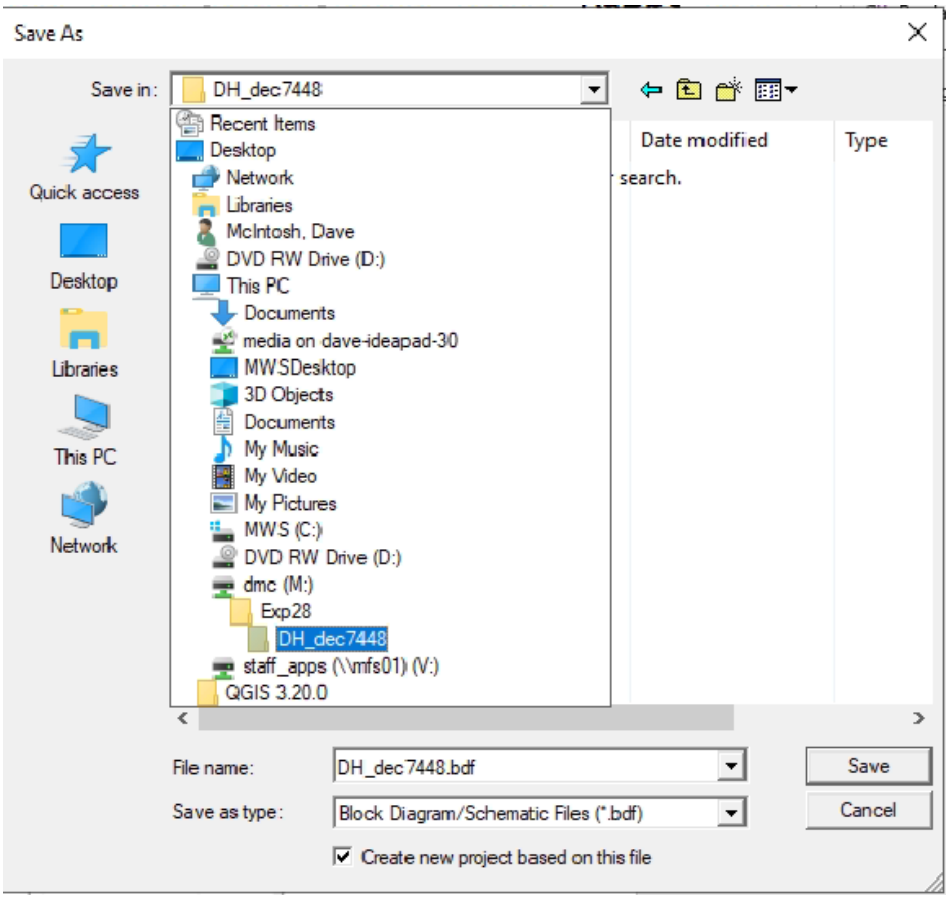


Figure 4. Save As Dialog

Click ***Yes*** to creating a new project in the New Project Dialog (Figure 5).

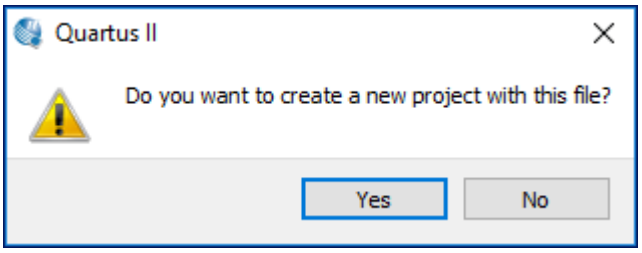


Figure 5. New Project Dialog

This will start the **New Project Wizard (**Figure 6**)**.

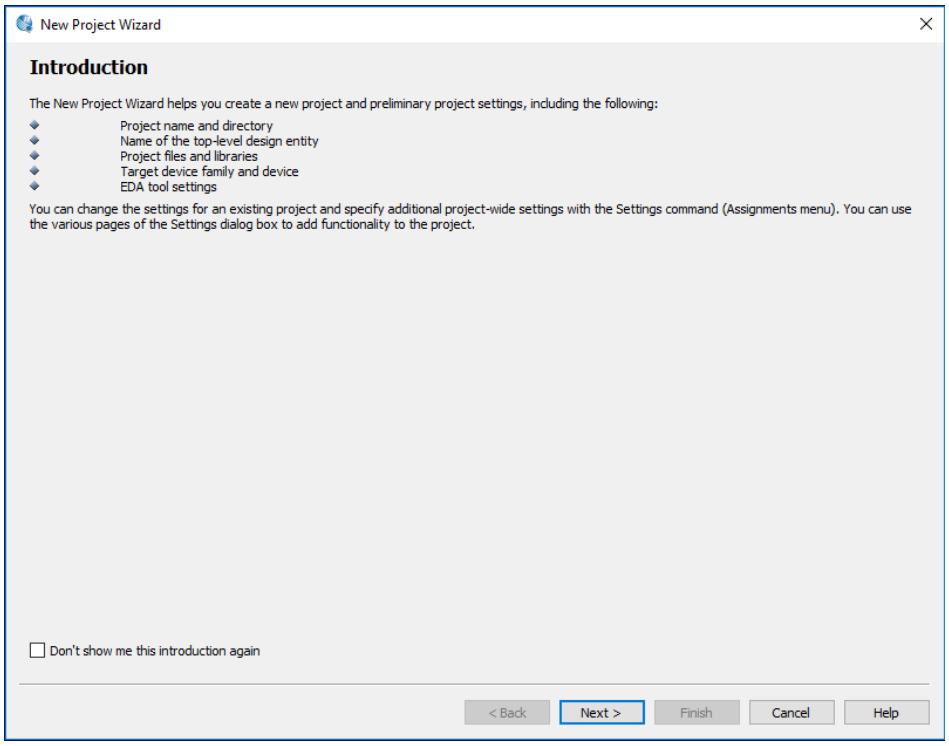


Figure 6. New Project Wizard

Click ***Next*** to move to page 1 of the New Project Wizard (Figure 7). The working directory and name of the project should have been entered automatically in this case, because you have built the project by first saving the .bdf file and opting to create a project based on this file.

[However, if and when creating projects directly from the New Project Wizard in future (instead of from a file) then you will find that you need to enter the new project folder name yourself (the working directory folder name, within Exp28 parent folder) and the project name too. For a smooth experience in this Experiment, always name these two the same as one another. Quartus will auto-fill the 'top-level design entity' name for you, th4e same as your project name.]

Click ***Next*** to move to page 2 (Figure 8).

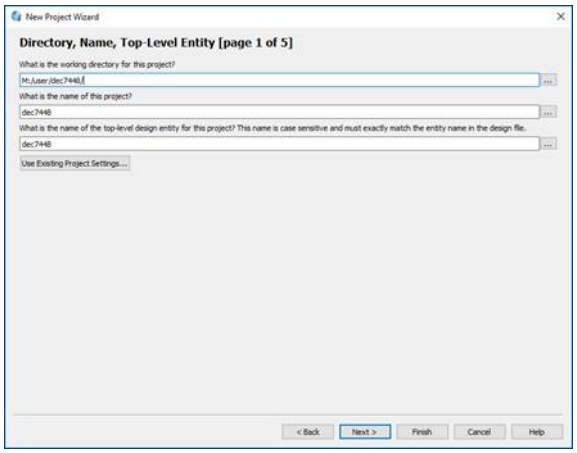


Figure 7. New Project Wizard - page 1

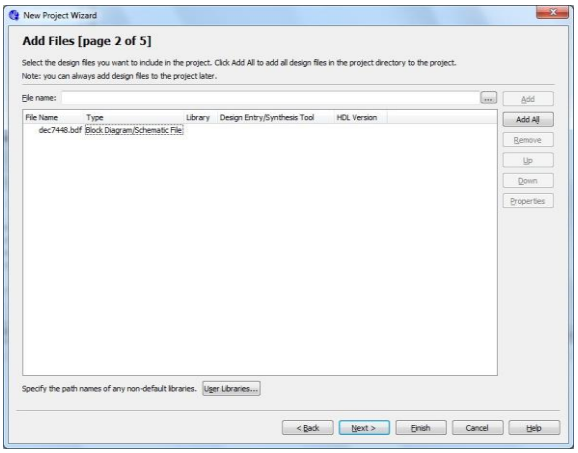


Figure 8. New Project Wizard - page 2

Click ***Next*** to move to page 3 of the New Project Wizard (Figure 9). Select **Cyclone II** family with the correct package (FBGA), pin count (484), and speed grade (7). Select the **EP2C20F484C7** device and click ***Next*** to page 4 of the New Project Wizard (Figure 10).

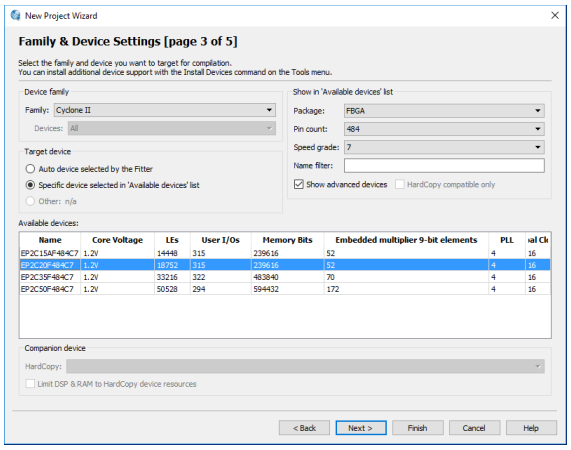


Figure 9. New Project Wizard - page 3

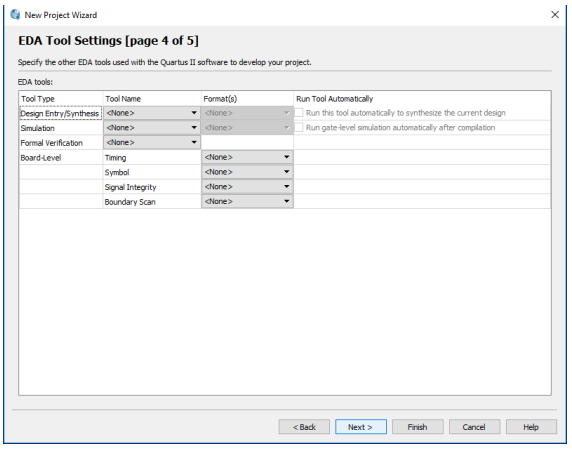


Figure 10. New Project Wizard - page 4

Click ***Next*** to move to page 5 of the New Project Wizard (Figure 11). Click ***Finish*** to exit the New Project Wizard. You are now ready to begin designing!

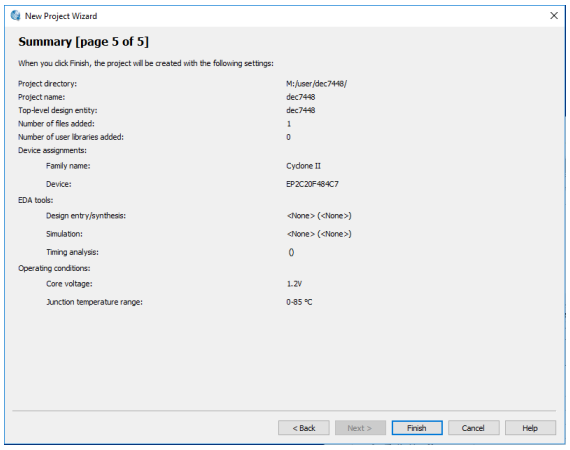


Figure 11. New Project Wizard - page 5

2.2. Using the Graphics Editor

Double click anywhere in the empty Graphics Editor window. The *Symbol* dialogue box will open (Figure 12). Expand ***others\ maxplus2*** library and then select **7448** in the ***Libraries*** box and click OK.

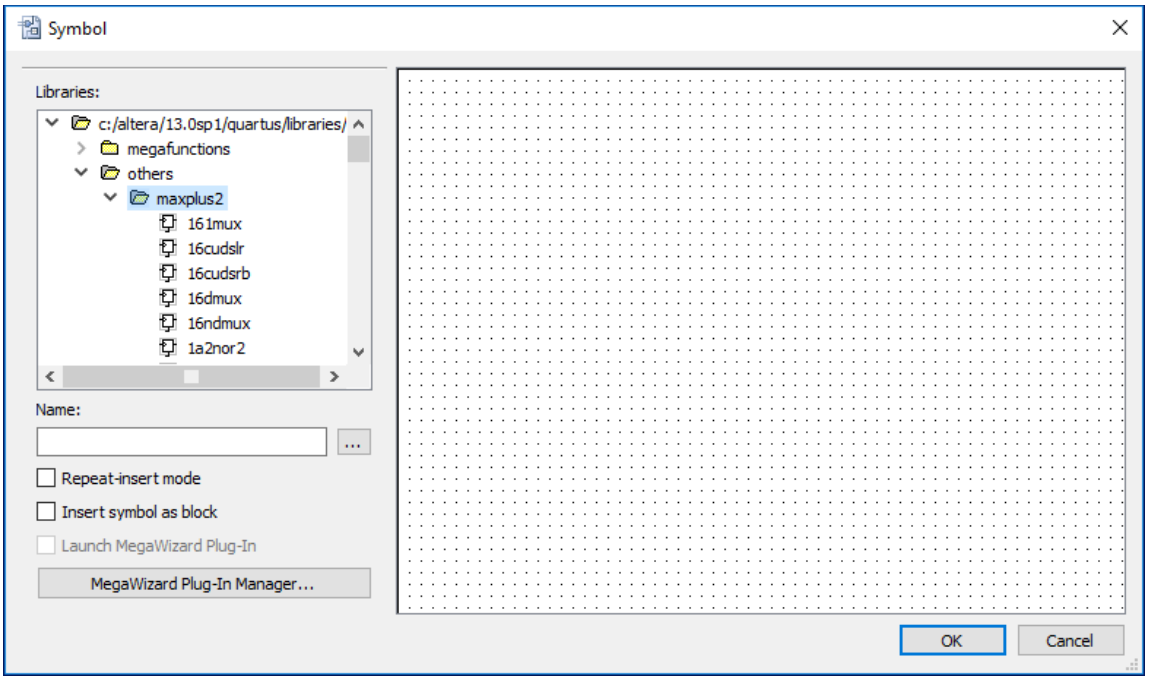


Figure 12. Symbol Dialog box

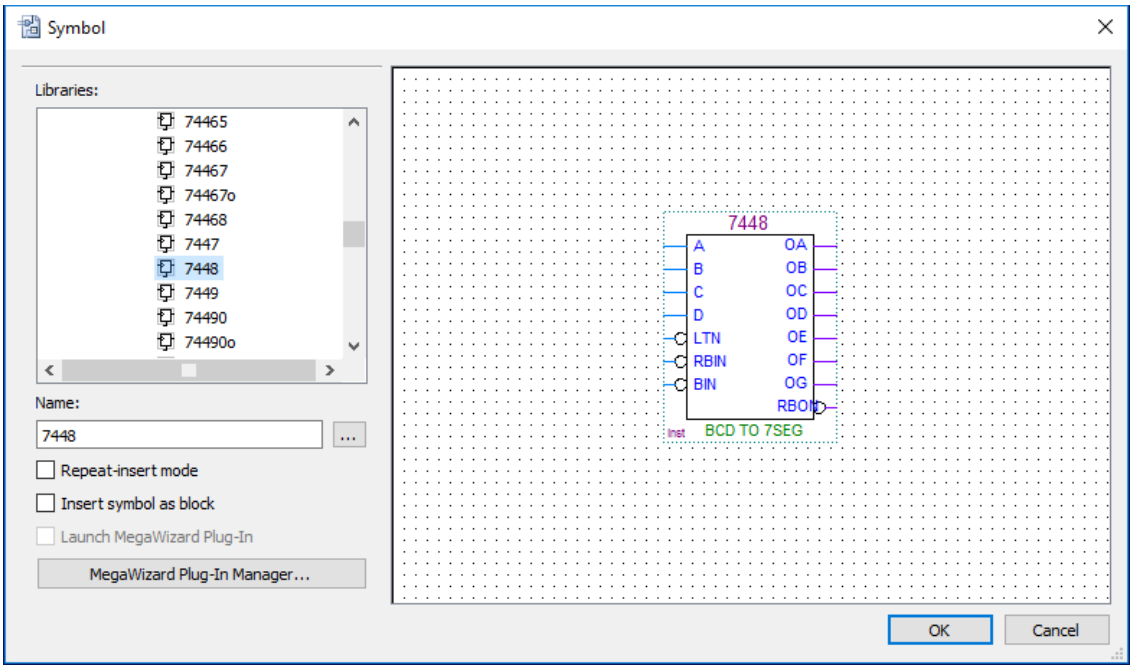


Figure 13. 7448 Symbol Selection

A 7448 symbol will appear in the Graphics editor screen (Figure 14). Place the 7448 symbol by left clicking.

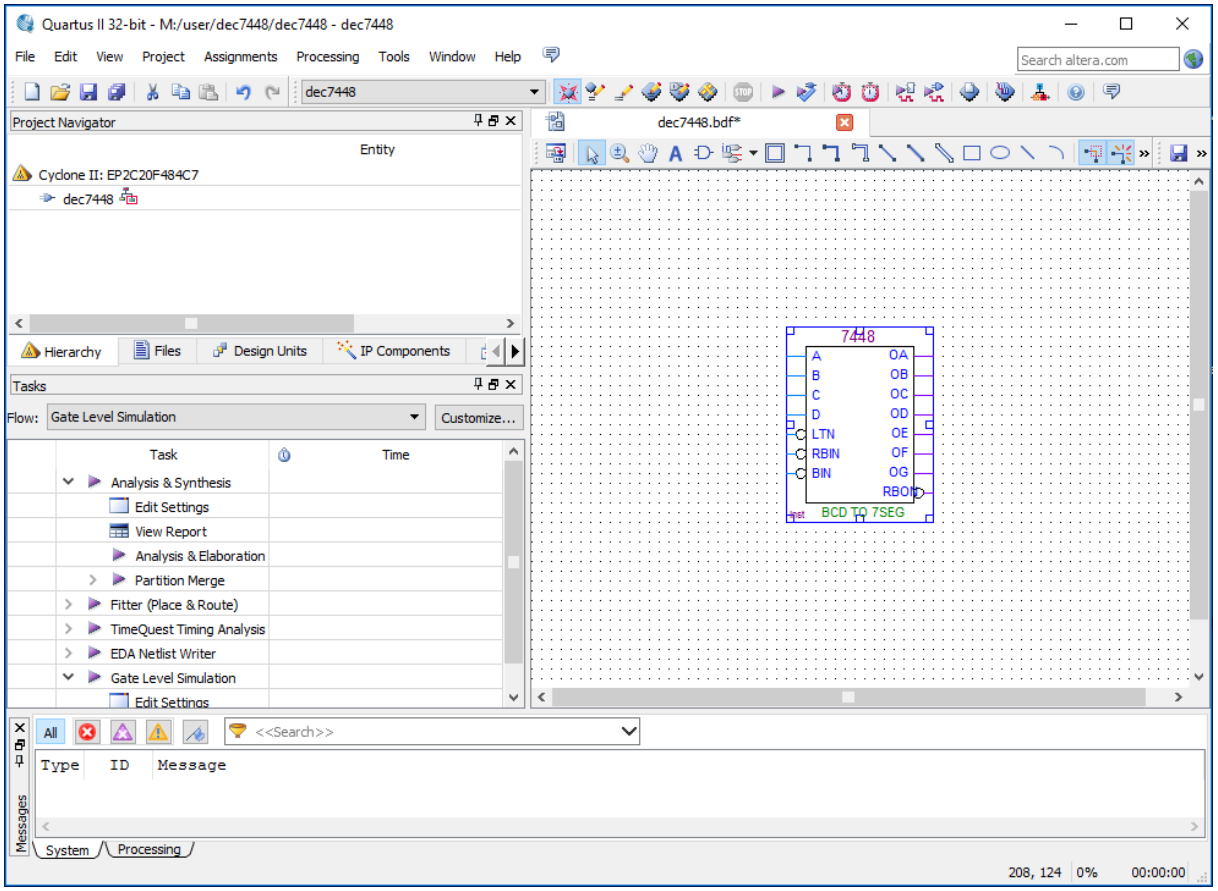


Figure 14. Graphics Editor

The Graphics Editor has many built in symbols. It has primitives such as AND gates, NOR, gates, etc. It also has *megafunctions*, which are combinations of gates that form higher level objects such as multipliers and dividers. Double click on any empty part of the Graphics Editor window to bring up the *Symbol* dialog box again. In the *Symbol Libraries* window expand the ***Primitives*** folder to see the different symbols available in the primitives library. Don't select any right now; you can look at the symbols in more detail later, if you want to. For now, just click on Cancel and continue with the tutorial.

Since we just want to emulate a 7448, most of our work has already been done. All we need to do is make input and output connections and connect any control lines that need to be set. To see what is inside the 7448, double-click on the 7448 symbol. You should see a schematic like that shown in Figure 15.

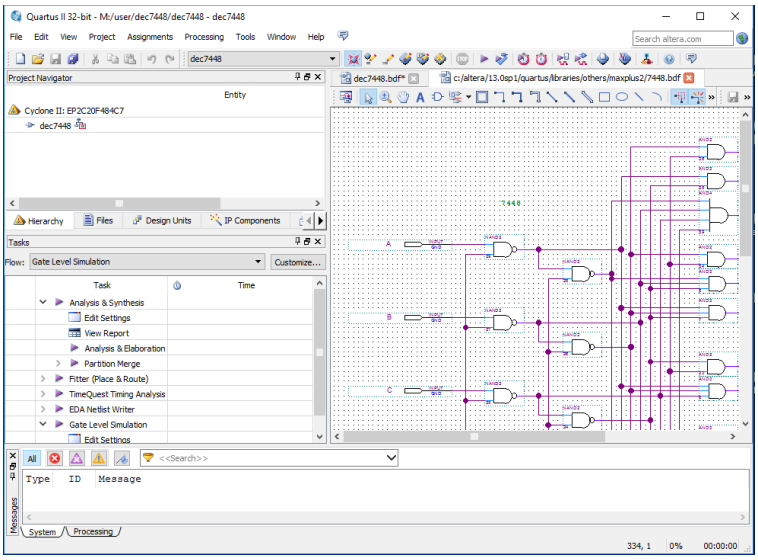


Figure 15. 7448 Schematic

Aren't you glad you didn't have to draw the whole thing? By double-clicking on the symbol, we have gone down a level and are now looking at the inside of the symbol. This is an example of what is meant by a design *hierarchy*. We have a top-level schematic, which contains the 7448 symbol. If we go down one level into the 7448, we can see the schematic for the 7448 itself, which is made up of primitives such as logic gates and I/O pins. Close the window showing the inside of the 7448 (it will have the title *7448.bdf*) by clicking on the orange cross.

Now we need to add inputs and outputs to the drawing. Double click inside the Graphic Editor window, anywhere to the left of the 7448 symbol. You should get the *Symbol* dialog box like before. Enter ***input*** for the symbol name to select an input (Figure 16).

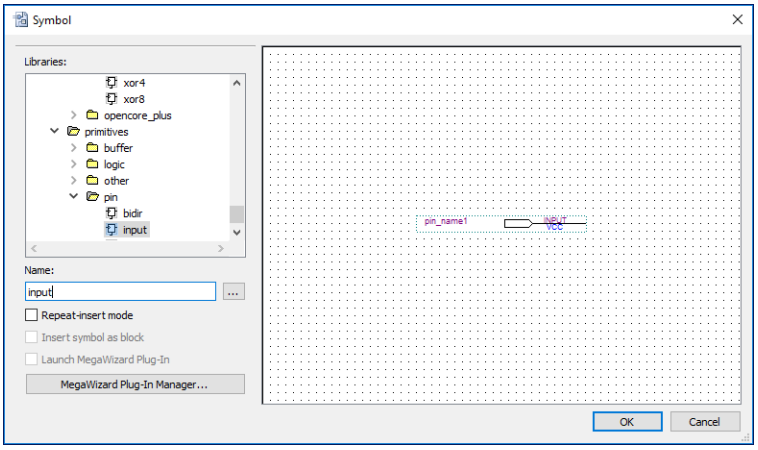


Figure 16. Input Symbol

Click OK and place the input symbol in your design by moving it to the correct location and left clicking (Figure 17). The input symbol represents an input to that layer of the design. If the design is the top layer then it represents a physical input pin on the PLD.

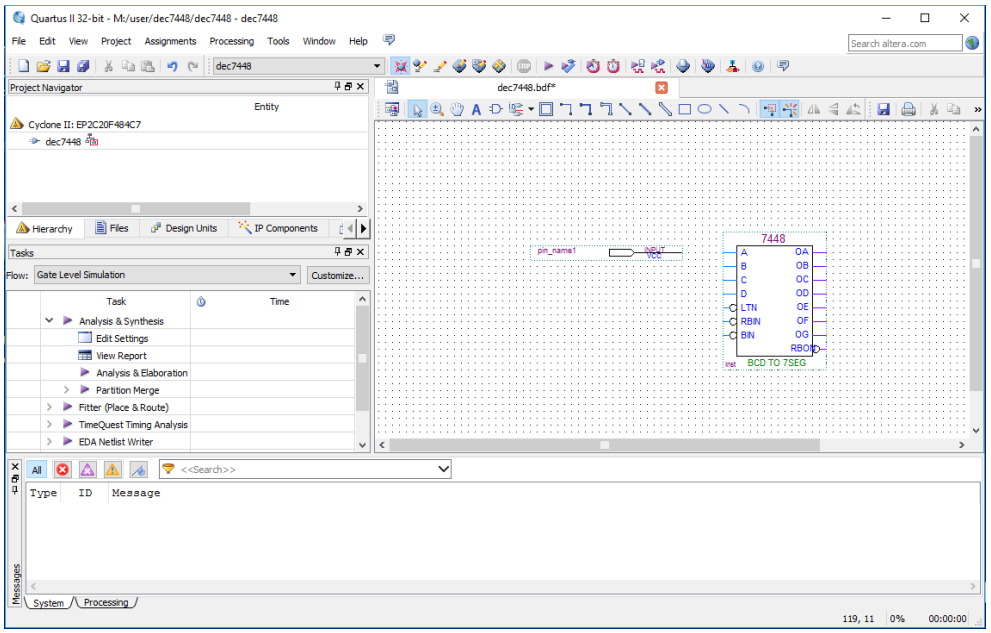


Figure 17. Design with one input

2.3. Drawing Your Schematic [3 Marks]

Now we need to connect the input pin of the PLD to one of the inputs of the 7448. Move the cursor to the right end of the horizontal line in the input pin symbol. The cursor will turn into a cross, indicating that you can attach a wire to that point.

Now click on the right end of the input signal and hold the mouse button. Drag the cursor to the left end of the pin labelled A on the 7448 and release the button. You have now connected the PLD input pin to the input of the 7448. It should look something like Figure 18.

Click on the input pin symbol and drag it so that the wire between it and the pin on the 7448 is straight. The input pin should be selected. Select **Edit->Copy** to copy the pin. Click just below the first pin to select an insertion point and then select **Edit->Paste** to paste in the copyof the pin. (Alternatively, you can hold down the Ctrl key on the keyboard and click on and drag the pin. A copy will be created wherever you dragged to). Move it until it is just below thefirst pin and aligned with it horizontally. Connect the second pin to the B input. Repeat this to create a third and fourth pin. Connect these to inputs C and D. Your schematic should now look like Figure 19.

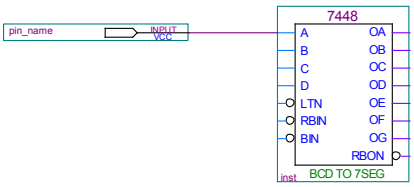


Figure 18. One input connected

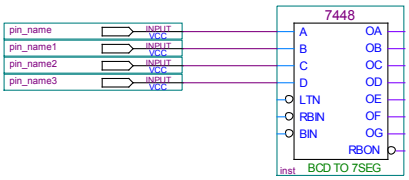


Figure 19. Four inputs connected

Although the pins on the 7448 are labelled correctly already, the design inputs are all labelled PIN_NAME. Double click on the first pin and change the Pin Name to **INPUT_A**. Depending on where you click the Pin Properties Dialog (Figure 20) may appear. This also allows you to change the pin name.

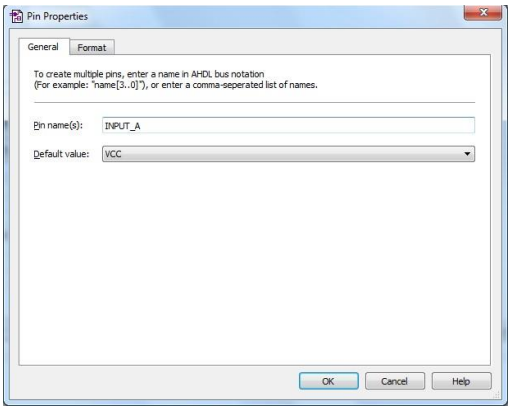


Figure 20. Pin Properties Dialog

Repeat this for the other three pins, naming them **INPUT_B, INPUT_C,** and **INPUT_D**, as shown in Figure 21.

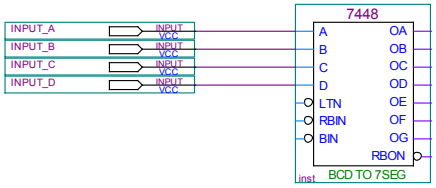


Figure 21. All input pins named

Now we need to add output pins. Repeat the procedure you used to get the first input pin, except use the symbol name ***output***. Copy the pin to get a total of seven pins. Connect them to the seven outputs labelled OA through OG. Label the output pins as OUTPUT_A through OUTPUT_G. Your schematic should now look like Figure 22.

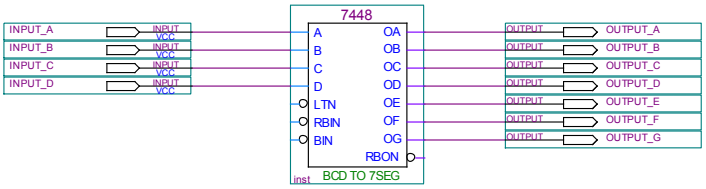


Figure 22. All pins named

We have three input pins that are not connected. Connect them to positive power supply voltage using the following procedure. Add a new symbol to the schematic above and to the left of the 7448 symbol, using the same procedure you did for the input and output pins. This symbol name should be vcc. This symbol represents the positive supply voltage connected to the FPGA. Connect the vcc symbol to the LTN, RBIN, and BIN pins on the 7448. Your schematic should look like Figure 23. Note that when wires cross, they are NOT usually connected unless there is a connection blob where the wires cross. So the Vcc symbol is NOT connected to any of the input pins A, B, C, and D. The schematic is now complete. Select **File->Save** to save your work.

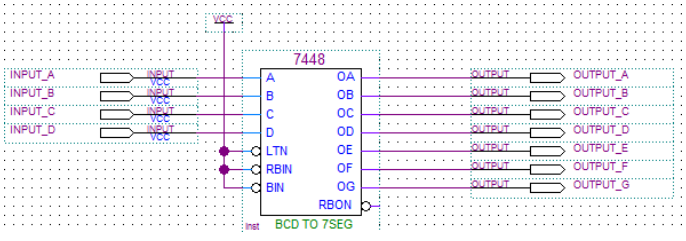


Figure 23. All pins connected

Double click (or right click and select **Properties**) on the vcc and change its instance name to **inst1 (**Figure 24**)**.

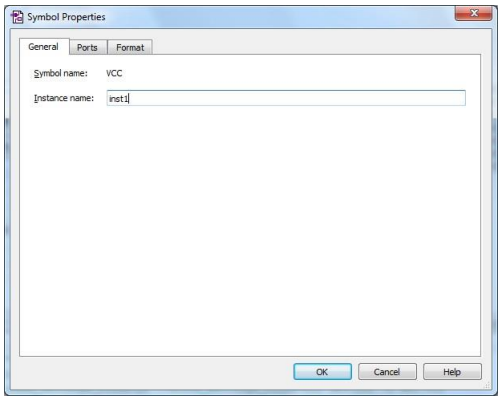


Figure 24. Symbol Properties Dialog

Now **compile** the design: under the Processing dialogue click on Start Compilation, Figure 25. You may be prompted to save your design changes (because you changed the vcc instance name) – click Yes.

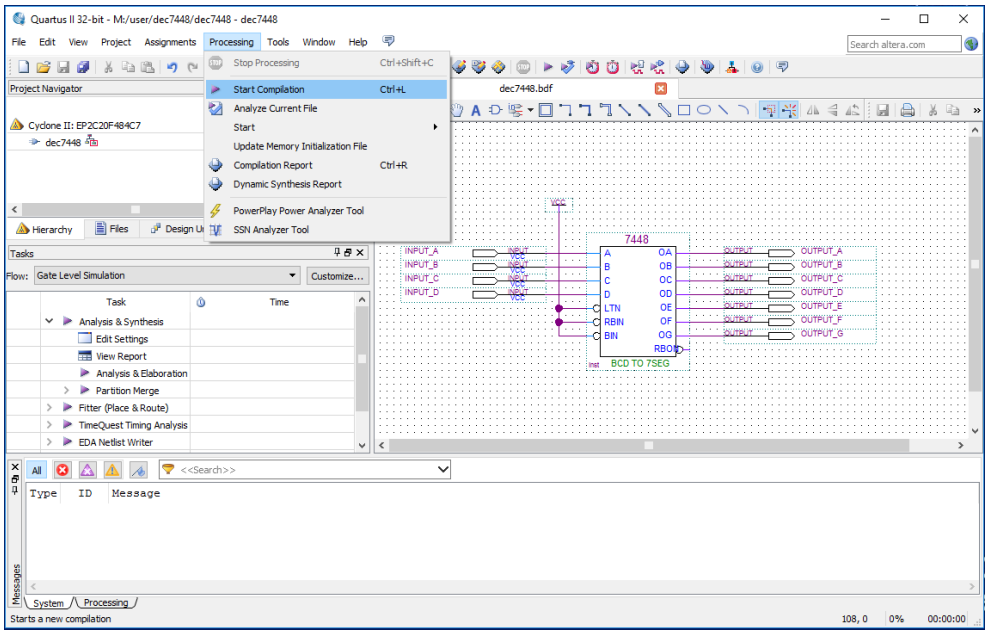


Figure 25. Start compilation dialog

Finally, you should get a dialog box like the one in Figure 26, stating that the compilation was successful, with some warnings.

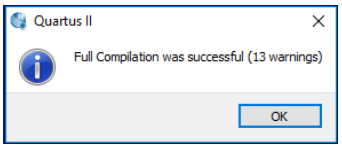


Figure 26. Full Compilation Dialog

If you did everything correctly, you won't see any errors but you'll often see some warnings. In developing more complex projects, you will probably see some errors, and lots of warnings. An **error** is a problem so serious that the compilation can't continue. You would have to fix the error before you can compile. A **warning** won't usually stop the compilation, but you should look at the warning message in case it's something you need to fix before proceeding. Some warnings can be ignored, while others will need to be fixed. Look at the help messages and use your own judgment and experience to determine which warnings you need to heed.

The error messages are generally quite detailed and clear, and you can usually determine what they mean just by reading them. You can also use **Help → Message List** to get more information on the error. (If using the Web Edition from home, this will open the Intel Quartus Prime Pro Edition Help version 21.1. Click on 'List of Messages' in the Content menu – bottom-left – and find your message ID.)

The Compilation Report (Figure 26) shows how many pins your design used, and also how many Logic Cells (LCs) your design used.

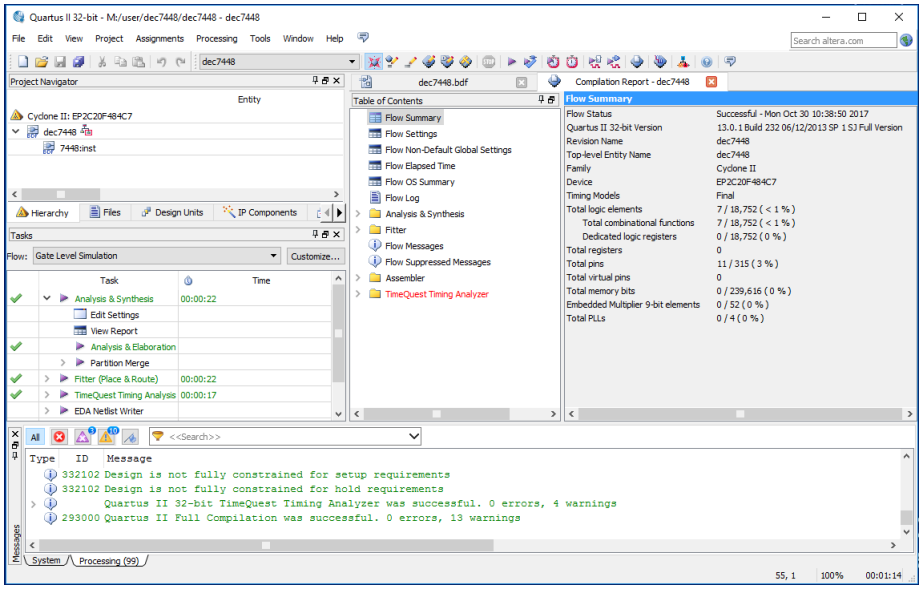


Figure 27. Compilation Report

2.4. Waveform Entry for Simulation [3 Marks]

We have entered our design and it has compiled without errors. The next step is to run the program and see if it works as we expected. Use inputs similar to the waveform given in Figure 28 and find the output. **(See Section 4 of the 'Supporting Material' document).**

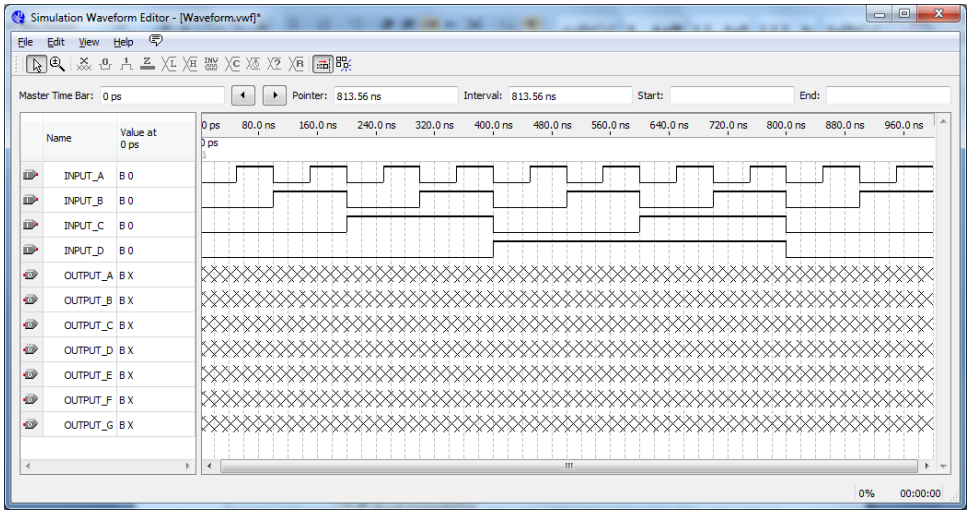


Figure 28. Completed Input Waveform

Following simulation, you must now plan the pins and program the DE1 board, and take further screenshots and photos to provide evidence of this – see Section 3 (and Section 2 as well) of the Supporting Material document. You will notice strange behaviour on the 7 segment displays – that's because the segment LEDs are active LOW. Solve this by inserting NOT gates before the outputs on your schematic (then recompile and try again.)

2.5. Design Project based on Verilog Code: Decoder [4 Marks]

Save any changes and close the simulation and the previous project (File ➔ Close for the simulation, then File → Close Project for the project). *It's very important to close the project, to avoid your new schematic in the next instruction becoming associated with the previous project!*

Create a new schematic file named **YI_7SegmentDecoder.bdf**, as part of a new project named **YI_7SegmentDecoder**, in a new folder (within your 'Exp28' folder on the M:/ drive) also named **YI_7SegmentDecoder**, following the process outlined in Section 2.1 above.

If you prefer, an alternative way and perhaps safer way to achieve this is, and a good habit to get into, is to create the new project **before** the new .bdf file. To do it this way, select File ➔ New Project Wizard. On page 1 of 5, change the working directory to Exp28/**YI**_7SegmentDecoder (you will be prompted to 'approve' the creation of this new folder) and in the second field, enter the project name **YI**_7SegmentDecoder. Then, follow the wizard steps as described in Section 2.1 above. Having created the new project, now create your new schematic file within this project, and save it (using File ➔ Save As), again using the name **YI**_7SegmentDecoder.bdf (same name as project). Using this name will, by default, tend to ensure that this becomes the 'top-level design entity' (see PreLab document Sect. 5).

> To avoid problems, unless instructed otherwise, it's usually safest to name the file you expect to become the 'top-level design entity' (overarching design file containing any relevant sub-designs) – and only this file – **with the same as the project title.** In this case, your top-level design entity is a Block Diagram/Schematic (.bdf) file. The same point would apply if you wanted the top-level design entity to be a Verilog HDL (.v) file, only in that case you would name the module within the .v file by the project name too. If you get into problems, there is a way to reassign the top-level design entity (see Prelab document) and a demonstrator can help you if necessary. For now, name your .bdf file the same as the project name (and the project folder name); save other files such as Verilog files by some slightly different name, unless they are going to be the top-level design entity. Make the module name inside a Verilog file the same as its filename, for simplicity. The instructions in the script usually tell you exactly what names to use anyway.

Now, having first created and saved an empty block diagram / schematic file as above, you can create a Verilog file and add it as a symbol to the blank schematic (which is going to be your top-level design entity in this project). Select 'Verilog HDL File' from the **File->New** menu item. Then enter the text code for the decoder as shown in Figure 29, but replacing the identifier 'seven_segment' after the 'module' keyword with '**YI_seven_segment**' using your last-letter initials. The lines beginning with // are comment fields which are used to document designs. (If a multiple-line comment is necessary, begin this with /* and end it with */.)

Hint: to save yourself some typing at this point, see the Prelab material, but be careful in pasting text from a PDF to your .v file, and don't forget to replace the module name as indicated above.

Note: in Figure 29, notice how text such as the case statement is indented. This is not compulsory in Verilog, however indentation of related lines of code (as shown in the Prelab) is good practice as it greatly improves the readability of your code. Please follow this practice.

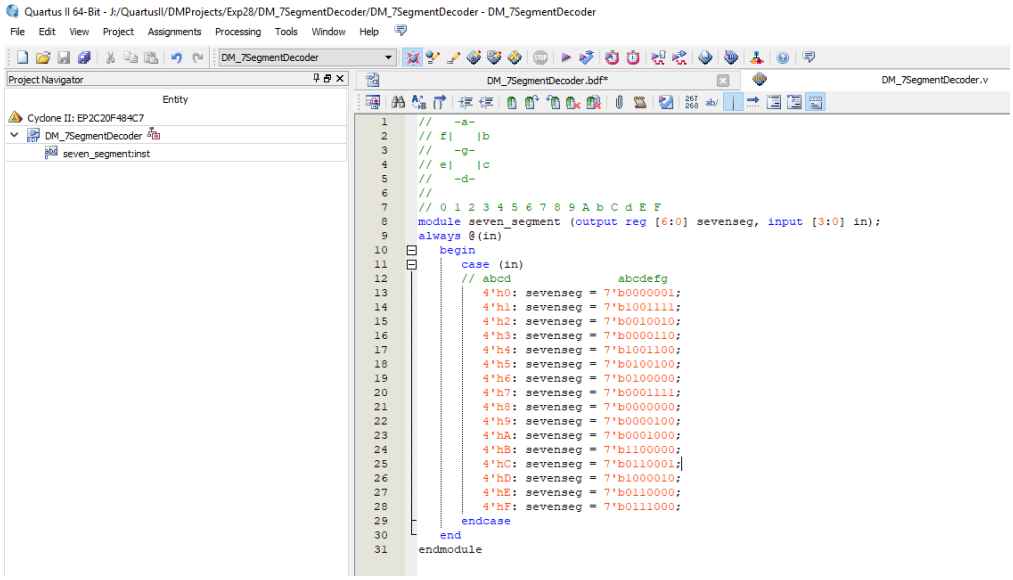


Figure 29. Verilog 7-Segment Decoder file (.v)

Save the Verilog file using **File->Save** in your project's directory with the filename "**YI**_seven_segment.v" (.v indicates a Verilog Design File) and make sure that the check box for ***Add file to current project*** is selected (Figure 31). Don't forget to use your last-letter initials, even though they are not shown in the screenshot below; this is important for documenting your work, and it's also useful for the Verilog filename and the module name within it to be the same as one another.

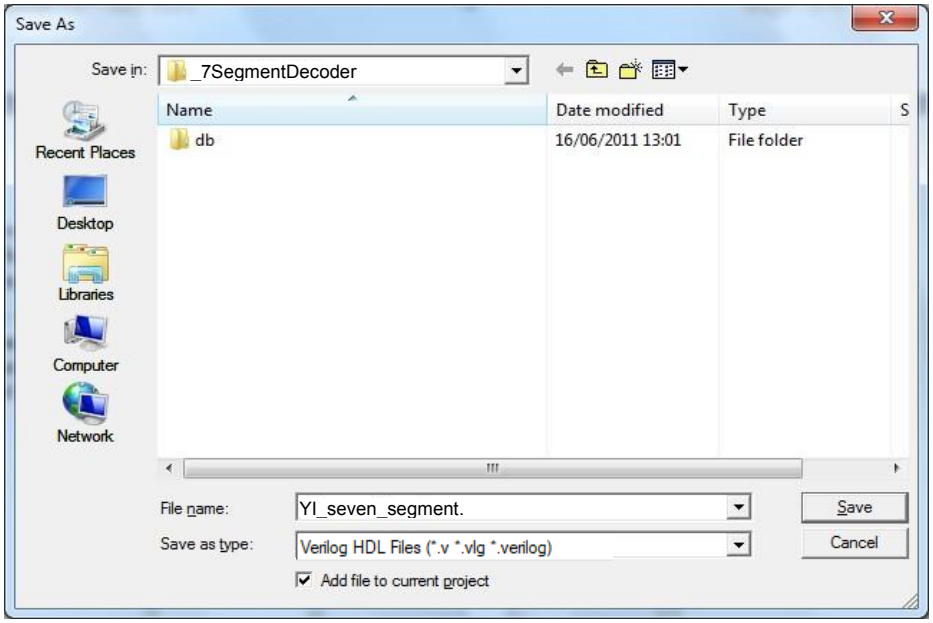


Figure 31. 'Save As' Dialog

To create a graphical symbol from this file to include in the schematic select **File** → Create/Update → Create Symbol Files For Current File**.**

Next, we need to insert the created symbol into the top-level schematic file. Right-click anywhere on the window of the top-level .**bdf** file and select **Insert->Symbol (**Figure 32**)**. Expand the Project Library and select your newly created **YI_seven_segment** module. Select OK to place the symbol in your design.

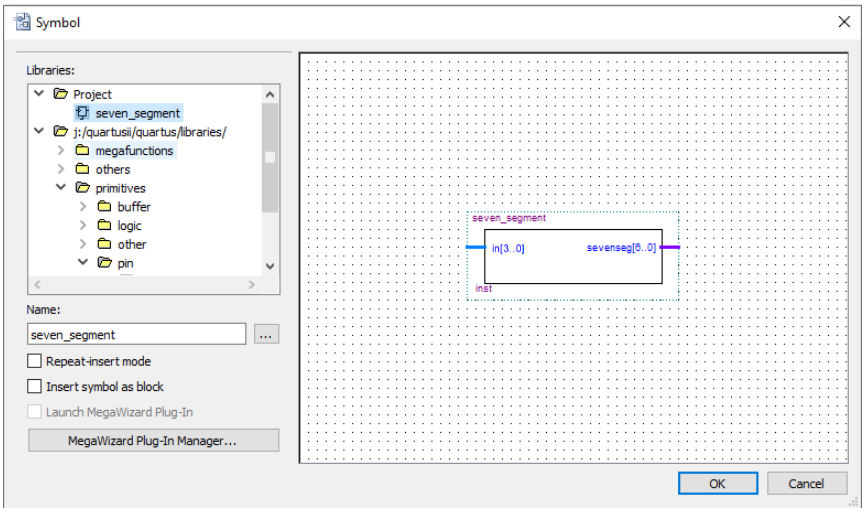


Figure 32. Select Symbol Dialog

Notice how the inputs to the seven_segment module are shown as a single thick line and labelled in[3..0]. This represents a "bus" of four connections. Similarly, your outputs are a bus of 7 connections. As we don't want the compiler to think we are directly connecting single wires to buses, we need to label the input/output pins as buses too, just as the bus inputs and outputs themselves have already been labelled in our Verilog design. What you should now do is insert one input and one output pin (similar to your earlier method for the first project), but following

this, right-click the input pin and in the Properties dialogue, give the name in[3..0] to the input pin (Figure 33). Of course, in[3..0] indicates a group of four signals labelled in[3] down to in[0], which will travel on the device's input bus. For convenience, we're keeping the naming identical to the naming used in the device itself ("in" for the inputs, and "sevenseg" for the outputs.)

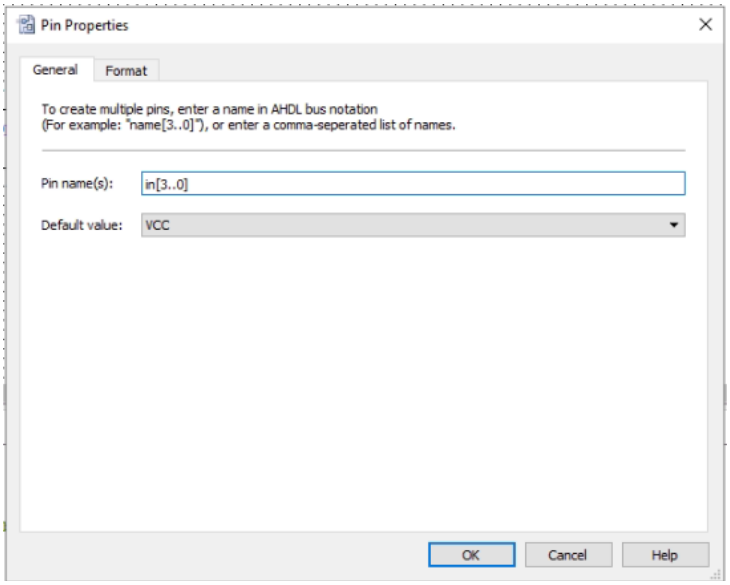


Figure 33. Pin Properties Dialog

Do the same to the output pin, typing in "sevenseg[6..0]" to match the device's output bus. Figure 34 shows what your eventual design should look like.

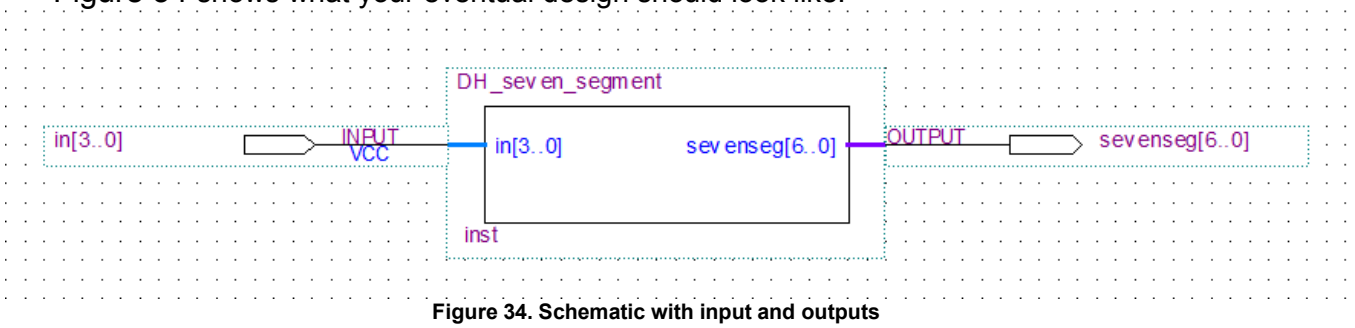


Figure 34. Schematic with input and outputs

Next, we want to assign the pin numbers using the Pin Planner, to enable us to eventually program the desired behaviour on the DE1 development board. **However, you must first compile your project design or nothing will appear in the Pin Planner!** To do so, click the purple 'play' shortcut button on the toolbar, or follow Processing → Start Compilation. You may be prompted to save some changes to files at this point. **Now, please refer again to the document "Exp 28 - Supporting Material" available on Canvas for a reminder about how to assign pin numbers using the Pin Planner.**

The details of your pin assignments will obviously be different to those in the example in the Supporting Material Section 3. If you haven't already, take a minute to read Section 2 in the same document, which gives you the information you need to understand the correct assignments in this case. The correct result of assigning the pins for the current project is shown in Figure 37 below. To reveal the pin assignments on your own schematic, you will probably need to right-click each new box with three dots, and select AutoFit.

Save your design (it may be safest to select File → Save All, or click the 'Save All' icon on the toolbar), and **recompile it** in preparation for simulating your design (next section).

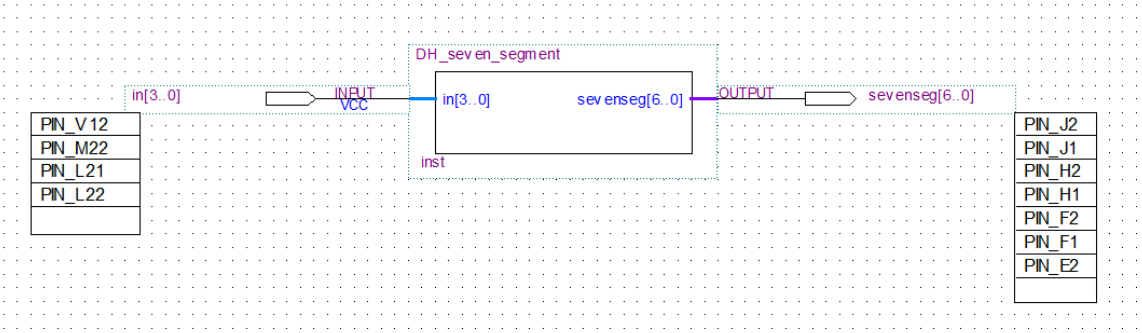


Figure 37. Design with pin allocations

2.6. Input Simulation for Decoder [4 Marks]

From **File** menu, click on the **University Program VWF**. Save the opened **.vwf** file in the project's folder with the name ***YI***_*7SegmentDecoder* (Figure 38; don't forget to insert your own last-letter initials as always, unlike the filename shown in the figure).

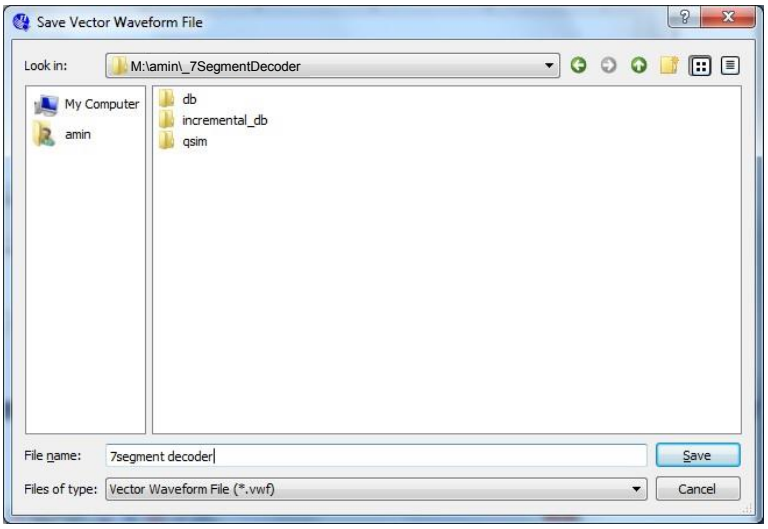


Figure 38. Save Vector Waveform File dialog

Open the **Node Finder** window by select **Edit → Insert → Insert Node or Bus (**Figure 39**)**.

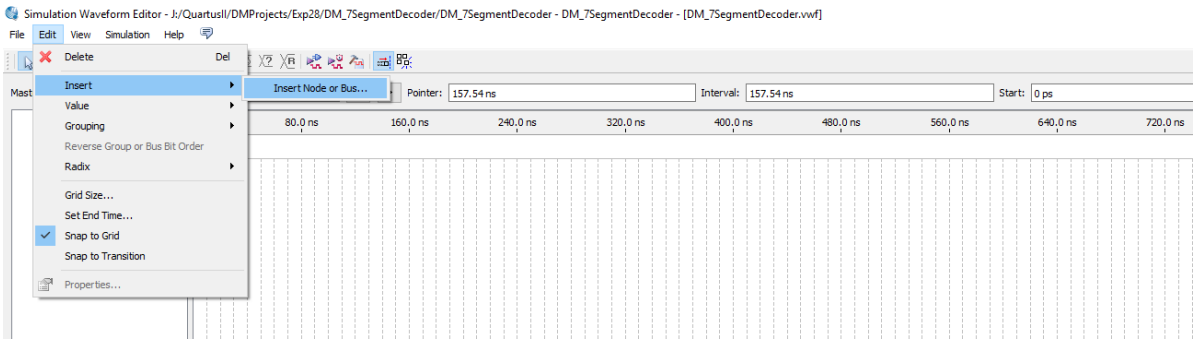


Figure 39. Insert Node or Bus Dialog

On the Node Finder Window click on the **List** button to list all the nodes. By clicking on the **>>** button, copy all the signals to the Selected Nodes area (Figure 40).

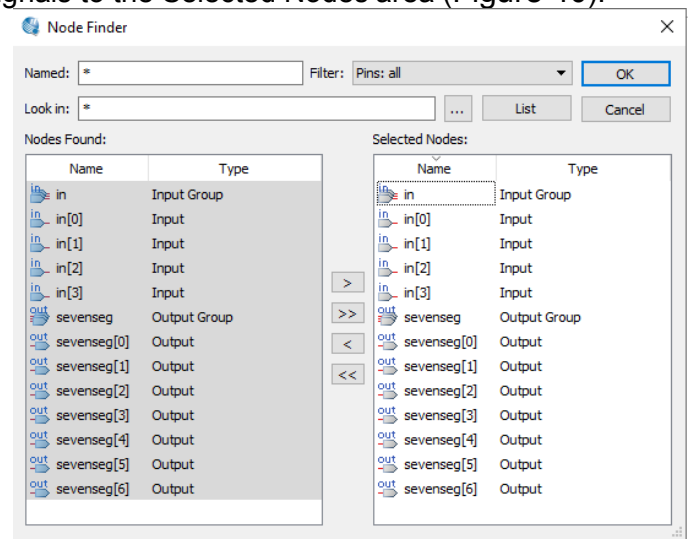


Figure 40. Selected Nodes

Click OK and then OK again to return to the Waveform editor (Figure 41).

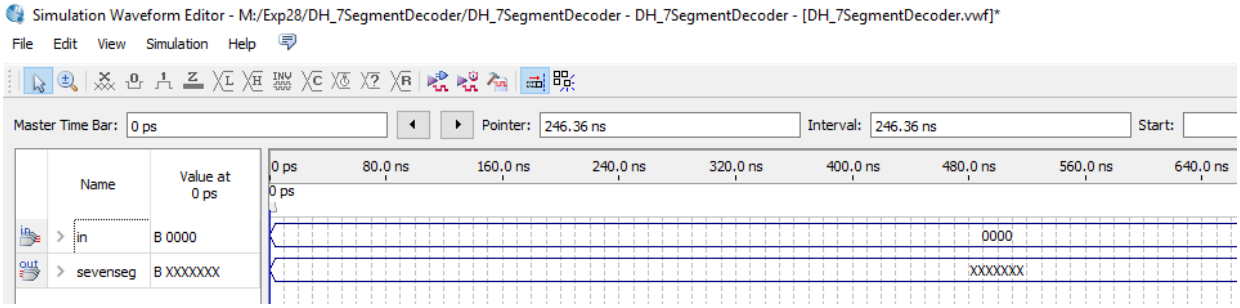


Figure 41. Waveform Editor

Note that the inputs are grouped together with a default base of “Binary”. When using buses it is often easier to only display the bus value in “Hexadecimal”. Right click on the **in** bus and change its radix to Hexadecimal (Figure 42).

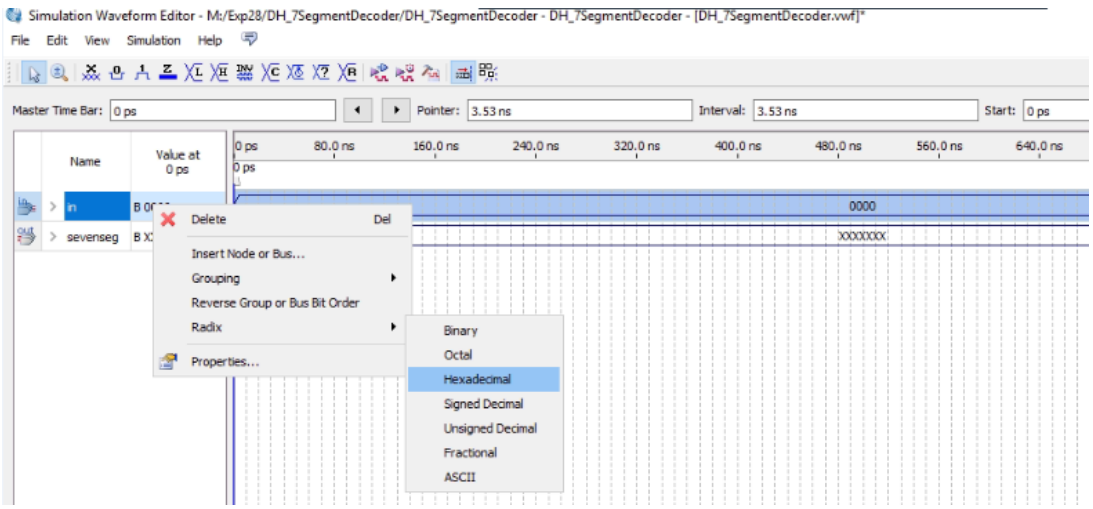


Figure 42. Change radix to Hexadecimal

Highlight the **in** bus by left clicking on its name, then click on the **Count Value** button. Change the Count period to 50 ns and then click OK (Figure 43).

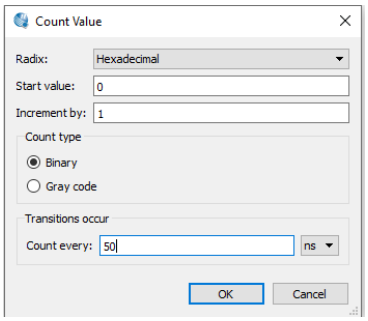


Figure 43. Count Value Dialog

This will generate a counting input (Figure 44). You will also notice in Figure 44 that I have expanded the “sevenseg” output group by clicking the arrow on its left.

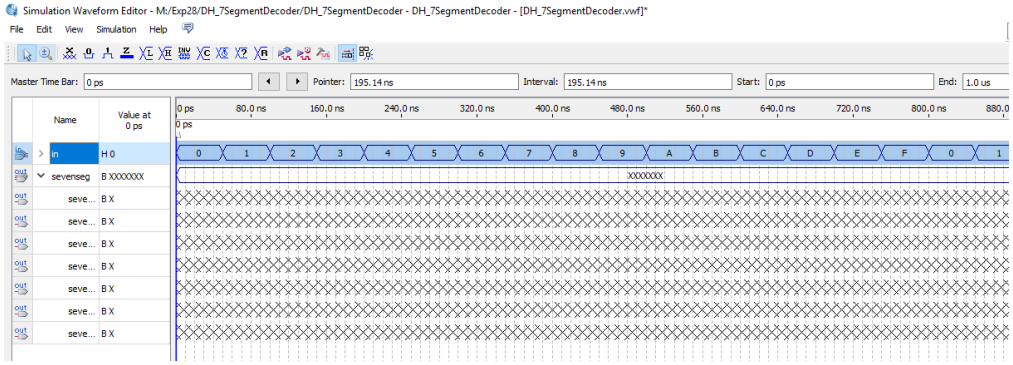


Figure 44. Waveform showing count simulation

Save the file. Select Simulation → Run Functional Simulation. *(If you are currently working on software downloaded to your own laptop or PC, then you must first – as described in the Supporting material Section 4.2 – select Options and check that ‘Quartus II Simulator’ is selected via Simulation → Options.)*

Notice that there is no delay between input and output in the Functional Simulation (Figure 45). Also notice that for an input value of “0” only out[0], which corresponds to the “g” LED, is disabled (high) whilst all the other LEDs are on (low). (The output out[6] corresponds to the “a” LED, out[5] to the “b” LED, etc. of a typical 7-segment display. **Of course, in your case they are labelled sevenseg[6], sevenseg[5], etc. and not out[6], out[5]…**)

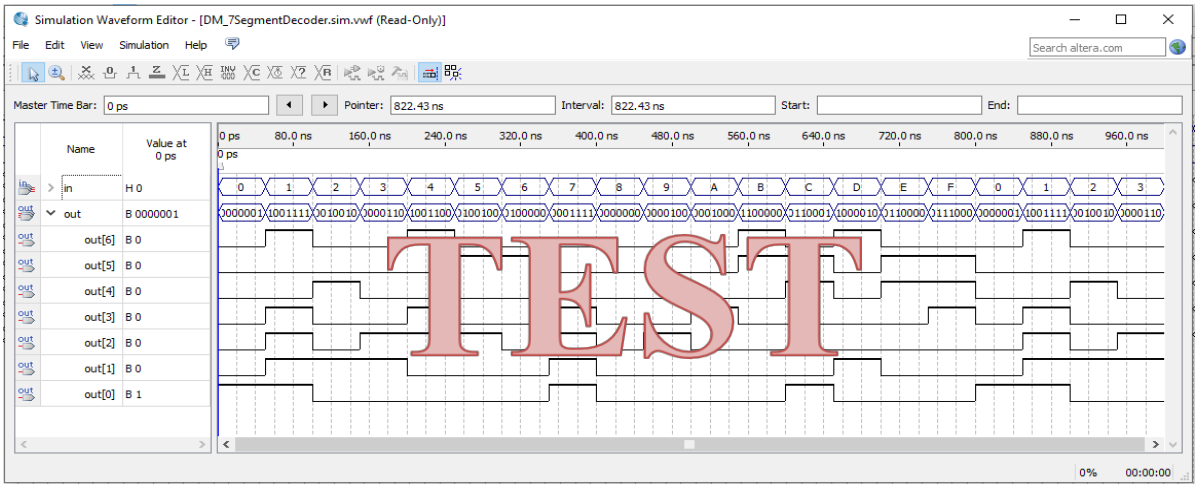


Figure 45. Functional Simulation Results

You must now program the FPGA by selecting the Programmer from the Tools menu (see the ‘Supporting Material’ document Section 3). Once the device has been programmed, you will be able to observe the LED outputs for the 16 combinations of DIP switch inputs. Document this for your Report with additional screenshots and photos.

3. Sequential design in Verilog [26 Marks]

3.1. Counter design in Verilog [8 Marks]

Perhaps the most straightforward **sequential** system to implement is a simple counter (Figure 46a):

```
1   module dec_count (output reg[3:0] count, output rco, input clk, clr, enc, ent);
2
3   assign rco = ((count == 4'd9) && ent);
4
5   always @ (posedge clk)
6      begin
7         if (clr)
8            count <= 0;
9         else if (enc && ent && count != 4'd9)
10           count <= count + 1;
11        else if (enc && ent && count == 4'd9)
12           count <= 0;
13        else
14           count <= count;
15     end
16  endmodule
```

Figure 46a. Counter design in Verilog

In the dec_count Verilog module above, a 4-bit counter is defined. There are four inputs consisting of the clock, two enables (ent & enc) and a clear. The outputs are the four count bits and also a RippleCarry out (rco), the latter being a 'continuous' assignment (see PreLab document). This code will cause the synthesis of four D type flipflops for the four output count bits. See the PreLab document, including the comments within the code there, for an explanation of the code's operation.

Close your previous project (File → Close Project), and then **create a new project (File → New Project Wizard)** called **YI_dec_counter** within a new folder also called **YI_dec_counter**, under your main '**Exp28**' folder (**see Section 2.5 if stuck**). Add a new, blank .**bdf** (block diagram / schematic) file to the project and save it as **YI_dec_counter.bdf**. This .bdf file is going to be our top-level design file again.

[Of course, if you really prefer it, having closed the previous project you can create the new project itself from this new .bdf file YI_dec_counter.bdf (if you close the previous project, create the new .bdf file and tick the correct box when saving it, you should be prompted to create a new project based on it). This was the way you created the very first project. However, it is recommended, instead, to get used to creating a project entirely from the New Project Wizard instead as instructed in Section 2.5.]

Enter the above design from Figure 46a in a new Verilog file, but as before, **don't forget to edit it to insert your own last-letter initials and underscore before the module name on line 1**, so the first line now begins: 'module **YI_dec_count**' (notice that this is deliberately slightly different to the project and top-level .bdf name – do not make it the same as them!) Again, see the PreLab document if you want to save yourself some typing, but take care!

Save the Verilog HDL file with this same second name as the module: **YI_dec_count.v.**

Then, choose **File → Create / Update → Create Symbol Files for Current File.** If the operation is successful (no errors in your file), this will allow you to subsequently enter the symbol into your **bdf** file by double- clicking (Figure 46b). Do this, and make sure to label all the wires as in Figure 46b… or (see next paragraph).

Now that you're familiar with input and output pins and what they are for, you might appreciate a handy feature to save time laboriously adding pins to each port: right-click your YI_dec_count symbol and select '**Generate Pins for Symbol Ports**'; this will do the job for you. Check the outcome to make sure the input and output pins are as you expected!

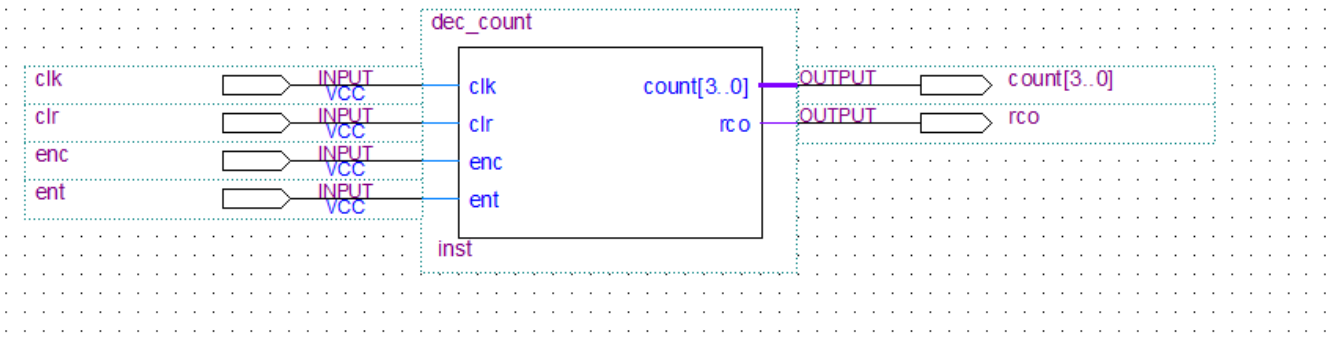


Figure 46b. Decade Counter Block

Now compile and simulate the design. For creating the clock waveform first of all highlight it by left-clicking on the clock. Then click on the **Overwrite Clock** button. In the dialog box (Figure 47) set the period and duty cycle of the clock. **Set the period so it's a 50 MHz clock (unlike Figure 47!)** to match the relevant clock signal provided on the DE1 board.

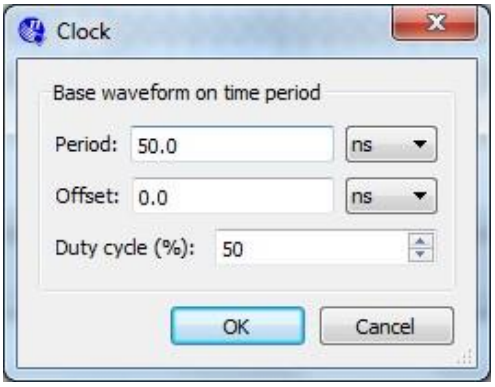


Figure 47. Clock Dialog

Figure 48 shows an **example** simulation input waveform.

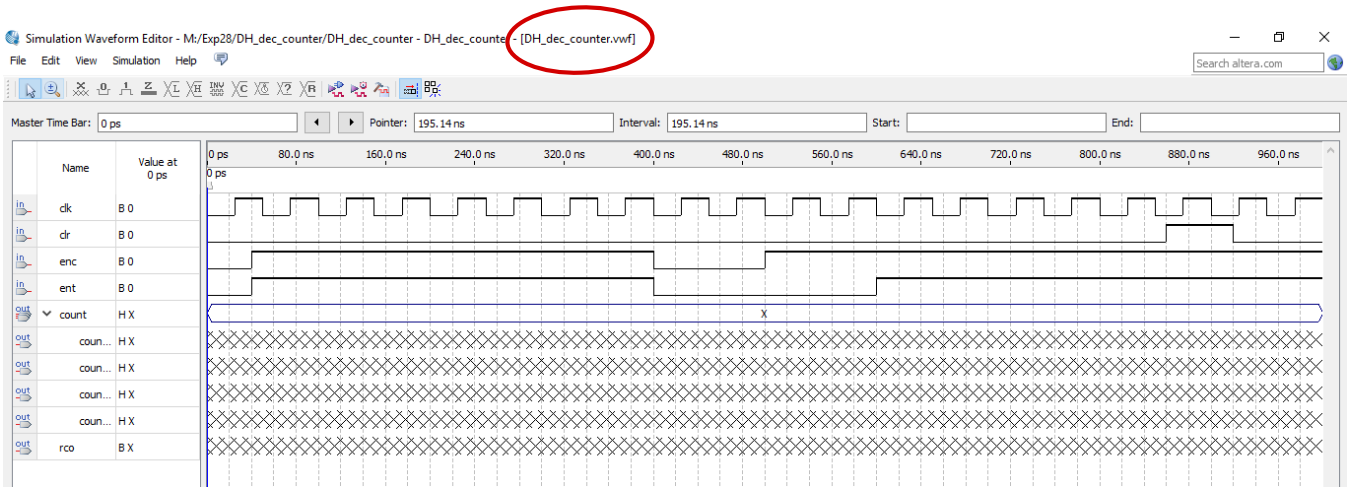


Figure 48. Simulation input waveform

If working at home then before you perform the simulation you will need to choose Simulation → Options (as mentioned before) and ensure that the **Quartus II Simulator** radio button is selected.

Choose a **functional** simulation, to avoid the slightly approximate timings mentioned in the Supporting Information. Your simulation file name should include your initials as always, in the form **YI_DecCounter.vwf** and as always this should be readable from the title of the window (Figure 49).

Note: the figures above/below show only one example of the simulation input signals. Your simulation files should not be exactly like Figures 48-49! For your report, you should choose a different combination of values for the other three control signals. In your report, you should explain the outputs at each significant period of time. For example, when enc is 0, the output is not changed even though there are two rising edges of the clock in this period.

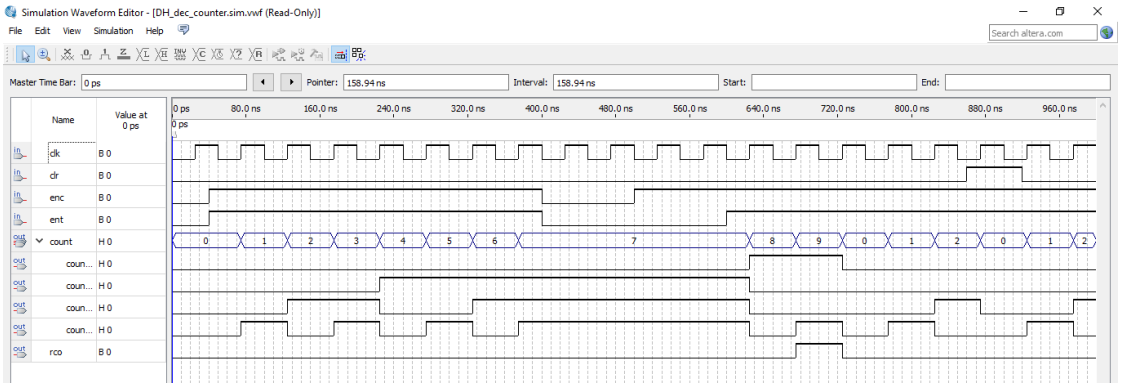


Figure 49. Example simulation output file

Once your dec_count simulations are done, as you have previously constructed a 7-segment decoder, it's time to connect the **dec_count** module to the input of the 7-segment decoder (Figure 50). **What's the best way to do this? It can be problematic to try and transfer files between projects**; it's probably safer to temporarily copy / paste your **YI_dec_count.v** code into a Notepad file. Then, close the current project and re-open your YI_7SegmentDecoder project (File → Open Project…), create a new Verilog HDL file within that project, copy and paste your YI_dec_count.v code into it and save it as a new part of the YI_7SegmentDecoder project. From the Verilog HDL file, select File → Create / Update → Create Symbol Files for Current File. Then, insert the YI_dec_count module to your **YI_7SegmentDecoder.bdf** schematic.

Check carefully that your resultant schematic design matches Figure 50. You'll need to delete the input pin from the YI_7segment module – its pin assignments will disappear too – and connect its input bus with the output bus of the YI_dec_count module (either place the latter so that it connects with the former, or mouse-hover and draw a connecting bus. You can easily check your connection by moving one of the modules.) You'll then need to compile your design and assign pins to the four new inputs as shown below. To achieve this, within the pin planner you'll also need to right-click → Edit → Delete (or select and press 'Del') the rows containing your previous pin assignments for the YI_seven_segment input bus, to avoid conflicts.

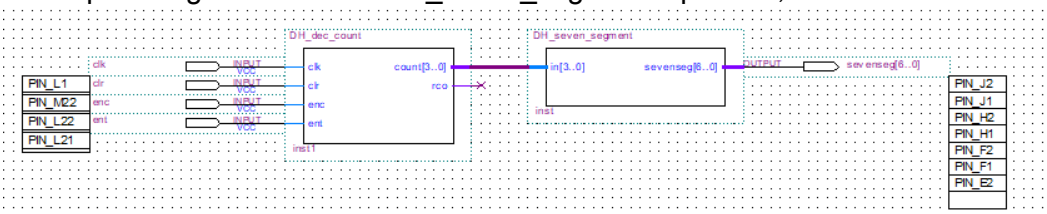


Figure 50. Design with Decade Counter

Your decade counter is now providing inputs to your 7-segment decoder. The inputs are the binary numbers equivalent to 0-9, which you were previously (Section 2.5) providing manually by flipping four of the toggle switches on the DE1 board in the appropriate combinations (see also 'Supporting material' Sections 2.2 and 3.1). The toggle switches now perform the functions associated with **ent**, **enc** and **clear** instead.

Your updated design now looks as if it might be ready for programming the Cyclone II FPGA again. First, perform a functional simulation of your design using the method already outlined. Save the .vwf file as '**YI_dec_count_7seg.vwf**' to distinguish it from your previous simulations of the 7-segment decoder. Remember that the clock input is on pin L1 which is a 50 MHz clock (reminder: see PreLab Section 2.1 / Table 1). Therefore, choose a 50 MHz clock, but as before, you should choose **your own** combinations of values for the other three control signals.

After this, you could try implementing your design on the FPGA. If so, though, you would find that all the segments of the display appear permanently on. This is actually what we would expect with a clock frequency of 50 MHz! The next section is about fixing this.

3.2. Once-per-second counter [18 Marks]

If we want the counter to increment approximately every second, we will need to enable the counter for only a single clock period every 50 million clock pulses. We can construct another counter, YI_sec_cnt, to do this and then use the output, 'second', to drive one of the enables. (Don't use the output of the counter to drive the clock itself, as this would make the design "Asynchronous"). Notice the 'continuous assignment' of 'second' (see PreLab). This ensures that 'second' goes to 1 immediately (not after another clock period) to become available as an enable input to the decimal counter at the next active edge of the clock.

```
module sec_cnt (input clk, output second);

reg [27:0] new_count;

assign second = (new_count == 27'd49999999);

   always @ (posedge clk)
         begin
               if (new_count == 27'd49999999)
                     begin
                           new_count <= 0;
                     end
               else
                     new_count <= new_count + 1;
         end

endmodule
```

Create a new Verilog file within the current project called **YI_sec_cnt.v** and enter the code to produce a pulse every second. As always, don't forget to insert your last-letter initials in front of the module name as well as the filename (so the first line should begin "module YI_sec_cnt" where YI are your last-letter initials). Then create a symbol file, and integrate this with the previous design to produce a circuit which counts every second (Figure 51). Don't forget to remove the enc input pin which is now no longer relevant. Again, if you need to check your new connections, move the sec_cnt module around to see if the relevant wires are really connected. If the connection is maintained, it may still be OK even if there isn't a 'blob' like below (a peculiarity of the software is behind this).

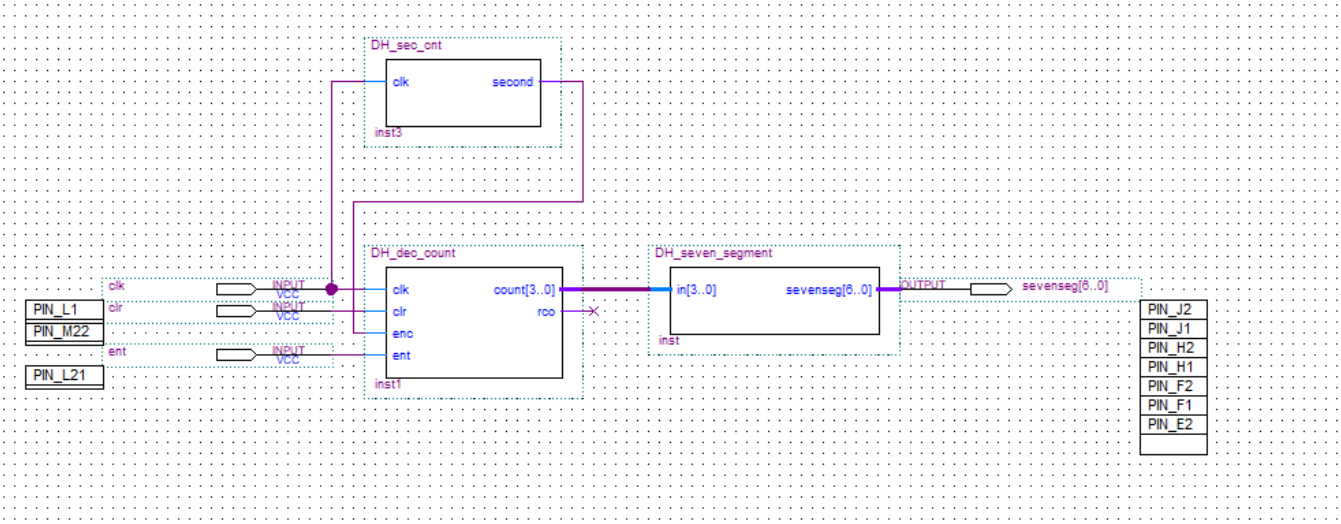


Figure 51. Modified design counting once per second

Now program the FPGA using the DE1 board. You should be able to see that the rightmost 7-segment display increments every second. You can also test the Clear and Enable switches. **As usual, document everything with further screenshots / photos.**

However, if you try and **simulate** this design you will find that a useful simulation period would be far, far beyond the maximum 'End Time' available (100 us). In fact, the counter would have to count to 49999999 before incrementing the **dec_count**. Standard practice is to **modify** the count value (in code) for simulation purposes, i.e. change the new_count value from 49999999 to a more manageable value like 5. **Do this and re-compile your design.**

In steps 1 to 3 below, you should continue using the modified count value as above.

1) Simulate your above design with the modified count value (call this one **YI_sec_count_7seg.vwf**) – but see the Figure and tips in the next paragraph first. Calculate the expected time between increments and check that this matches your simulation. Were there any surprises? Mention/explain this in your discussion.

   Figure 52 is a sample simulation for the modified **dec_count**. Simulate your counter **differently** and explain the effect of the control inputs. Choose a different pattern of control inputs from Figure 52 (including choosing your own clock period), but still one that is useful for demonstration purposes, and choose a sensible 'end time' etc. You may need to play with the clock period and 'end time' to fit a full cycle of relevant behaviour on the simulation, for full demonstration of what your design achieves. ***(Note that your simulation window will also differ slightly from Figure 52 in terms of the output names, etc., as per Figure 45, and perhaps in other ways too.)***

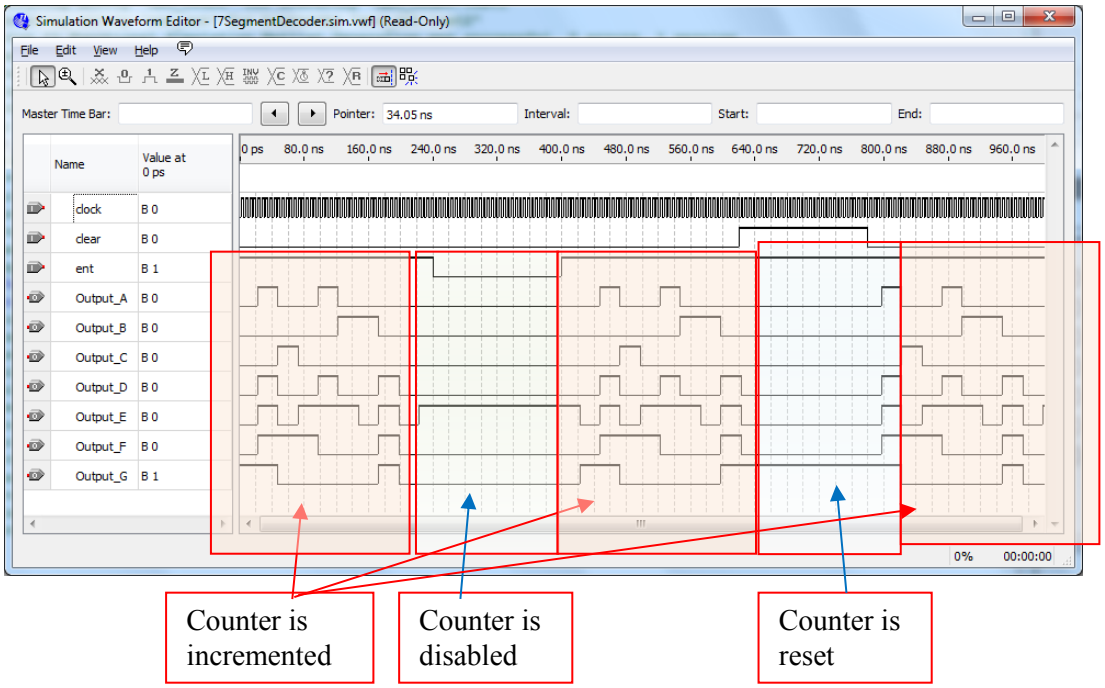


Figure 52. Example of annotated simulation waveform

2) Modify your design to make it a divide-by-12 counter (mod-12 counter – i.e. a counter which counts to 12), designed to display the results in hexadecimal as 0 to B on a single 7-segment display. **Record your results (screenshots & photos).**

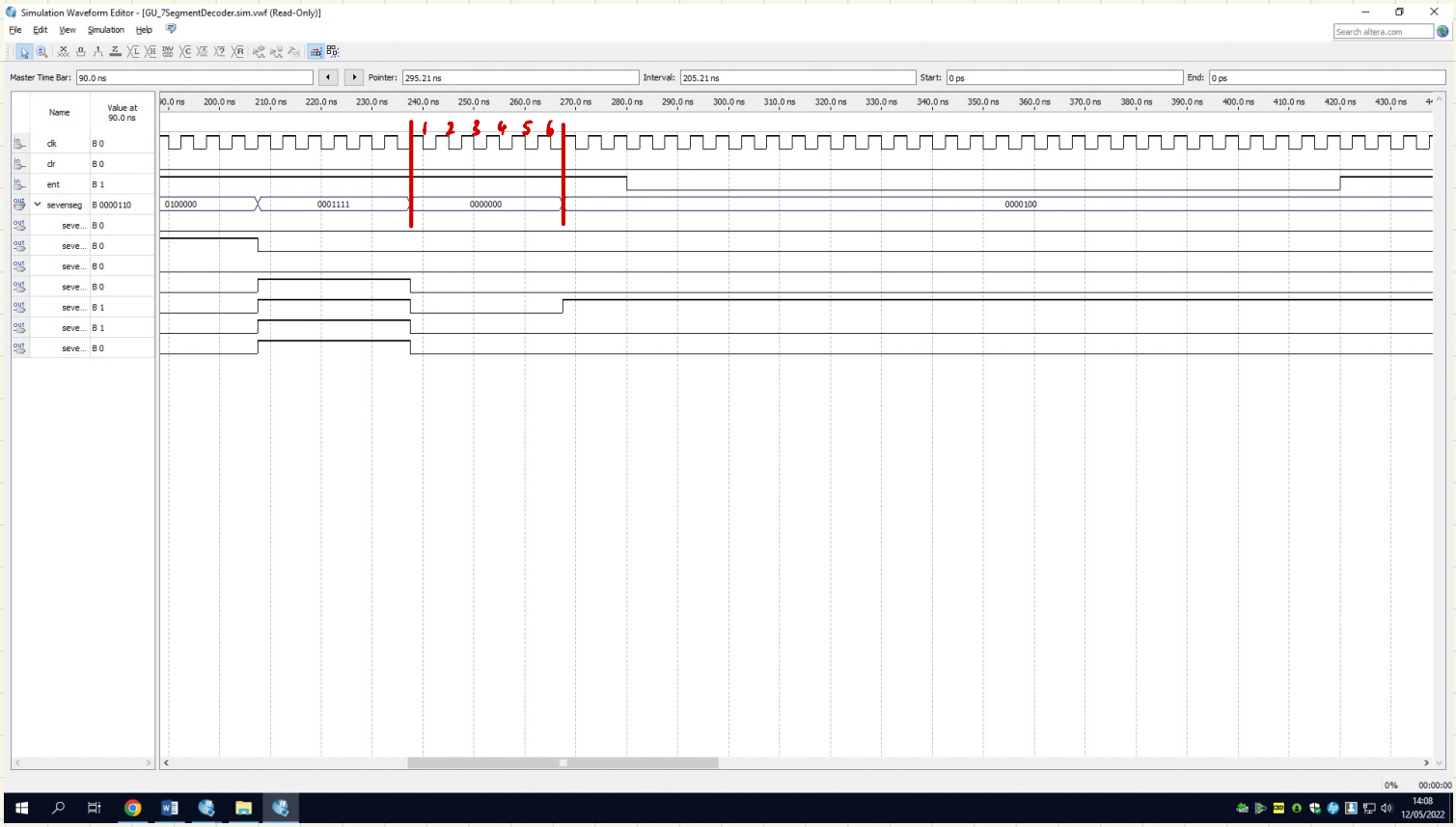
3) Now try cascading two divide-by-12 counters together, intended for displaying the resulting values on two segments of the quad 7-segment display. **Record some results (screenshots & photos)** showing that the counter counts from 00_H up to BB_H.

```
1  module GU_sec_cnt (input clk, output second);
2
3  reg [27:0] new_count;
4
5  //assign second = (new_count == 27'd49999999);
6  assign second = (new_count == 3'd5);
7  // Using a 50MHz clock. ADD takes one clock cycle. ADD 50M - 1 times takes one second.
8     always @ (posedge clk)
9        begin
10          //if (new_count == 27'd49999999)
11          if (new_count == 3'd5)
12             begin
13                new_count <= 0;
14             end
15          else
16             new_count <= new_count + 1;
17       end
18 endmodule
```

takes 5 clock cycles

takes 1 clock cycle

Total takes 5+1 clock cycle



4. Design Assignments [30 Marks]

Now that you are familiar with state machine entry in Verilog (Prelab), design and simulate digital circuits with the specifications below. Begin all folder names, filenames and Verilog module names with "**YI_**", and **do not include spaces** in file or folder names to avoid problems.

4.1. Single Pulser [10 Marks]

Generate a single pulser state machine which produces an output for 1 clock period only, when a pushbutton is pressed. Your design does **not** need to include pin assignments (yet – see Section 4.2).

Your design must produce output for only 1 clock period, even when the input pulse is much longer than this. The state diagram[3] in Figure 53 is a hint about how to achieve this (although it's incomplete as it does not show you the output / input values associated with states and transitions.) Please use a Moore machine[4] for your design. Your system should be **rising-edge** triggered. You will need to consider the following problems:

1) Generating a single output pulse

   A moment's thought tells us that a pushbutton (input) pulse is usually much longer (ms range) than a single clock period (us or ns). However, we require that this pushbutton pulse would still generate only a single clock-period output pulse. Your design must account for this point and you need to verify it by simulation.

2) Pulse synchronisation

   Because we are using a Moore machine[4], if the manual pushbutton input pulse was somehow less than one clock period (violating some guidance in the PreLab document), an output pulse might or might not be generated. This is due to the active edge of the clock and the *setup* and *hold* times for the state flipflops. Explore the limits of your design in this respect, by simulating with different pushbutton input pulse lengths and positions in the clock cycle. (It will need to be a **timing** simulation.)

Your report must include your Verilog code, a discussion of your design and an **annotated** simulation to cover various possible timings. Use a **timing simulation** to demonstrate/discuss points (1) and (2). (Simulate a **50MHz** clock frequency, and **disable the 'Snap to Grid'** option.) Explain why your design satisfies the constraint in (1), and discuss your findings for (2).

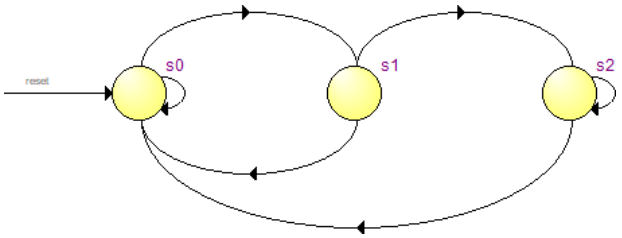


Figure 53.[3]

[3] This partial state graph was generated automatically in Quartus II using the State Machine Viewer. It does **not** follow our usual convention in ELEC211 (which you should follow on assessments), in which the state names (s0, s1 and s2) ought to be **inside** the state bubble.

[4] Note: it is also possible to design this finite state machine as a Mealy machine. In general, however, the timing is less critical in a Moore machine than in a Mealy machine. You are instructed to use a Moore machine in this instance for simplicity.

4.2. Pushbutton Enabled Counter [10 Marks]

Modify your latest '7-Segment with Decade Counter' schematic design (from Section 3.2) so that instead of the output of the 'second counter' being used as an enable, the output of the single pulser is now used to enable the counter. The counter should therefore increment by one when a pushbutton is pressed.

Your schematic should include pin assignments so that the design is ready to be programmed to the DE1 board. (See Supporting material Section 2.3 for information relevant to the input pin assignments in this case.) Use either KEY0 or KEY1 for the pushbutton: to decide, convert the final digit of your student number into binary.If the result ends in "0", use KEY0. If it ends in "1", use KEY1. (This pushbutton should be connected to the input of the single pulser circuit). Assign one of the toggle switches to be the reset input.) Test your design with a simulation.

Your report must include the schematic diagram and an **annotated** simulation.

4.3. Adder/Subtractor Logic Design [10 Marks]

Design a logic circuit using Verilog (recommended as more suitable for best achievement of the objectives in this section) or the schematic entry method if you prefer (but may be harder to achieve some objectives this way), to add or subtract two signed 4-bit numbers. The most significant bit of each number is the sign bit; therefore the range of input numbers would be **-8** to **+7**. The result would be a signed 4-bit number too.

A control input, **Add/Sub**, will determine the addition or subtraction operation. If the result is out of the range then an **overflow** flag must be set. For example the addition of **4** and **5** or the subtraction of **-3** and **7** should set the **overflow** flag.

The circuit must be synchronous, i.e. all inputs and outputs registered by a master clock.

The circuit should have a master reset connected to all of the registers and flip-flops.

In adding signed numbers, the overflow can be detected by XORing the two inputs' sign bits with the result's sign bit and carry_out bit (a 4-input XOR is needed). You can detect the overflow with other methods.

If you decide to use the schematic design entry you can insert the **7483** (4-bit Full-Adder) and **74171** (quad-DFF) symbols in your design from the library (but as noted above, behavioural Verilog is probably more suitable to some aspects of the current task than schematic capture).

The Adder circuit can be used for subtraction if you use the **two's complement** of the subtracted number.

Simulate your design to prove that it works with all possible combinations of inputs. Include combinations of positive and negative inputs with the addition or subtraction operations, to generate or not to generate overflows.

Your schematic should include pin assignments, planned for implementation of your design on Altera's Cyclone II FPGA board using the Toggle switches and seven segments. Use the left-hand seven-segment to display the **sign**, and the right-hand seven-segment for the absolute value of the number.

You will need to convert the signed result to an absolute value and use a flipflop for the sign signal.

Your report must include the schematic diagram and **annotated** simulation.

5. Assessment and Marking Scheme

This experiment is assessed by means of a pre-lab test and practical design / simulation work.

The marking scheme for this workbook is as follows:

- The pre-lab test: 30 Marks
- The practical work: 70 Marks (use submission template available on VITAL)

6. Plagiarism and Collusion

Plagiarism (including close paraphrasing) and collusion, or fabrication of data, is always treated seriously, and action appropriate to the circumstances is always taken. The procedure followed by the University in all cases where plagiarism, collusion or fabrication is suspected is detailed in the University's Policy for Dealing with Plagiarism, Collusion and Fabrication of Data, Code of Practice on Assessment, Category C, available at:
http://www.liv.ac.uk/tqsd/pol_strat_cop/cop_assess/appendix_L_cop_assess.pdf.

Follow these guidelines to avoid any problems:

(1) Do your work yourself.

(2) Acknowledge all your sources.

(3) Present your results as they are.

(4) Restrict access to your work.

Version history

Name	Date	Version
Dr D G McIntosh	April 2022	Ver. 8.2
Dr D G McIntosh	April 2022	Ver. 8.1
Dr D G McIntosh	April 2021	Ver. 8
Dr D G McIntosh	April 2020	Ver. 7
Dr D G McIntosh	March 2020	Ver. 6.5
Dr M López-Benítez	September 2019	Ver. 6.4
Dr M López-Benítez	December 2017	Ver. 6.3
Dr L Esteban and Dr M López-Benítez	November 2017	Ver. 6.2
Dr A Al-Ataby	October 2014	Ver. 6.1
Dr A Al-Ataby, F Davoodi Samirmi and Prof J Smith	October 2013	Ver. 6
Dr A Al-Ataby	October 2012	Ver. 5.8
Prof J S Smith	October 2011	Ver. 5.7
Dr S Amin-Nejad	July 2011	Ver. 5.6

Feedback:

If you have any feedback on your laboratory experience for this experiment (e.g. timing, difficulty, clarity of script, demonstration...etc) and/or suggestions about how the experiment may be improved in the future, please write them down in the space below and copy/paste this page to the end of your Report before the Appendix. This feedback is important for future versions of this script and to enhance the laboratory experience, and will not be assessed. If you wish to provide this feedback anonymously, you may do so by emailing this page to the EEE Student Experience Team (studyenq@liverpool.ac.uk) with a request that the feedback should remain anonymous but be forwarded to the ELEC211 module coordinators. Or, you can print this page and pass it in by hand to the Student Experience Team in the fifth floor office of the EEE building, with the same request.

Script re-writing award

If you think that this experiment could do with enhancement or changes and you have some ideas that you'd like to share, why not have a go at re-writing the relevant part of this script yourself? If it's good, you may get an award from the lab organisers with an official letter of thanks, and your name added to the version history list in future versions of the script. Something good for your CV. Contact one of the lab organisers for more details.