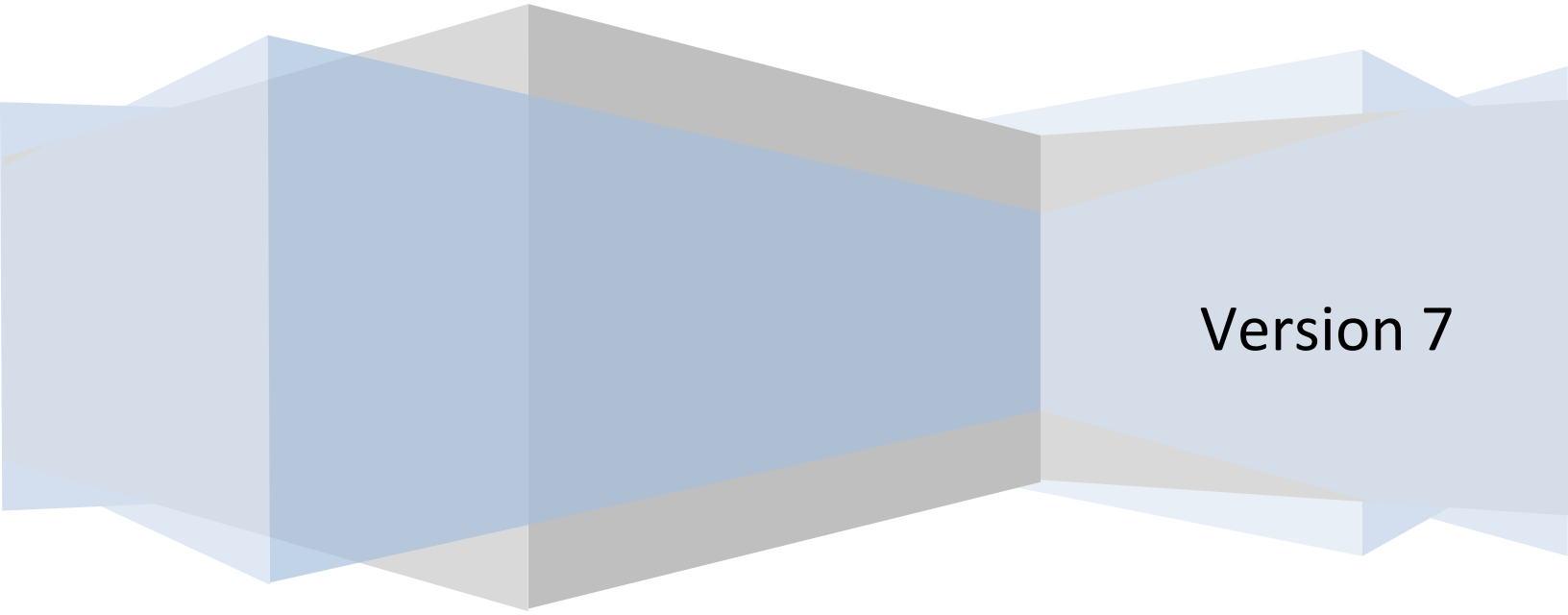
NEW JERSEY INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

LABORATORY MANUAL

ECE 641 Laboratory for High Performance Digital Signal Processing

Ali N. Akansu and Mustafa U. Torun
(akansu@njit.edu)



Version 7

© Copyright January 2013

ACKNOWLEDGMENT

The authors would like to thank Zhi Yong (Arthur) Li for his efforts during the development of this laboratory, Wes Weydig for providing us the ModelSim-Altera test bench files, Matthew Bajor, Andrew Hofmaier, and Jialiang Wu for their help in the development of the experiment “Memory Blocks, Megafunctions, Flash memory, and Building a Simple Function Generator,” Su Yu for his help in the development of the experiment "Radix-2 Cooley-Tukey Algorithm for 4-Length FFT,” Cem Benar for his effort to update FAQs in v7, and all students who shared their suggestions with us to improve this course.

We also acknowledge Altera North America for providing us DE2-70 Cyclone II boards used in the course.

Ali N. Akansu
Mustafa U. Torun
January 2023

TABLE OF CONTENTS

ACKNOWLEDGMENT 2

FPGA EXPERIMENT 1 Laboratory Orientation and Software Installation 4

FPGA EXPERIMENT 2 Finite Impulse Response (FIR) Digital Filters 11

FPGA EXPERIMENT 3 Infinite Impulse Response (IIR) Digital Filters 27

FPGA EXPERIMENT 4 Two-Channel Orthogonal Filter Bank 35

FPGA EXPERIMENT 5 Interacting with the Peripherals on the DE2-70 Board 43

FPGA EXPERIMENT 6 Radix-2 Cooley-Tukey Algorithm for 4-Length FFT 51

FPGA EXPERIMENT 7 Modulation Using CORDIC Algorithm 67

FPGA EXPERIMENT 8 Memory Blocks, Megafunctions, Flash Memory, and Building a Simple Function Generator 71

FPGA EXPERIMENT 9 Fast Convolution Using Number Theoretic Transform 83

FPGA EXPERIMENT 10 Error Control and Cryptography 87

FPGA EXPERIMENT 11 Adaptive Filters and LMS Algorithm 92

GPU EXPERIMENT 1 GPU Computing: Lab Orientation, Software Installation, and Vector Operations 98

GPU EXPERIMENT 2 Memory, Blocks, Threads, Synchronization, and Matrix Multiplication 107

GPU EXPERIMENT 3 Discrete Cosine Transform on GPU 116

FPGA EXPERIMENT 1
Laboratory Orientation and Software Installation

TABLE OF CONTENTS

A. Introduction
B. Quartus II 9.1 Installation
C. USB Blaster Driver Installation on Windows
D. DE2-70 Control Panel

A. INTRODUCTION

In this experiment we will install the software and drivers needed for the laboratory. **Please read FAQ at the end of the experiment in the manual if you face any issues**. First, go to course webpage and download the "ECE641 Materials" zip file. Unzip the file into a convenient folder and name the folder as "ECE641 Materials." The course web page is located at http://web.njit.edu/~akansu/COURSES/ECE641.html

The software required for this course are:

1. Quartus II 9.1 Web Edition
2. Nios II Embedded Design Suite 9.1

Installation files are available on the course webpage or on DVDs or shared by the instructor. We will use the simulator that comes with Quartus II 9.1. Students who are interested in more advanced simulation environments should also install ModelSim-Altera. Test bench files for each experiment are located in the "ECE641 Materials" folder.

In this manual it is assumed that you are using Windows. MacOS and Unix users are encouraged consult the instructor if they need additional help with the installation and read FAQ if you face any installation issues.

B. QUARTUS II 9.1 INSTALLATION

1. Obtain a DE2-70 board from the stock room. Download the installer and double click on the installation file or insert the installation DVD from the course instructor. Wait for the menu to load. Click on "Install Free Package" button. (See Fig 1.1) Click on the button **ONLY FOR ONE TIME** and be patient to proceed to the next screen. Initialization of the installer may take up to five minutes. Avoid performing any other task while the installer is initializing.



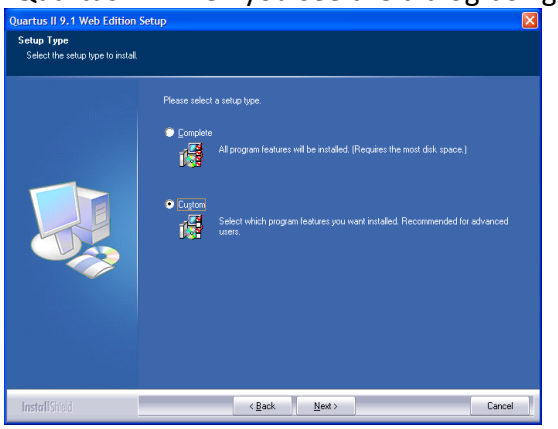
Figure 1.1

2. Once the "Quartus II Web Edition" installation starts, follow the instructions on the screen. Though the installation process is straightforward you may want to modify the components to be installed in order to save space. If you don't want to customize the installation skip this step and proceed to step 3. When you see the screen given in Fig. 1.2 select "Custom" and click on "Next" button. In the component selection list, mark only the following: Quartus II, SOPC Builder Support, IP Library, Cyclone II Support, Third-party EDA tool interfaces, and Tutorial files. You will not need the other components in the list in the laboratory.

3. When the installation is completed, you will be asked for consent to provide feedback to Altera. It is up to you to provide feedback or not (See Fig 1.3.)

4. Once the "Quartus II Web Edition" installation is complete the menu will lead you to installations for "ModelSim" and "Nios II." Make sure you install Nios II but you may skip installation for ModelSim.

5. Now double click on the "Quartus II 9.1 Web Edition" icon on your desktop. Select "Quartus II" when you see the dialog box given in Fig. 1.4.



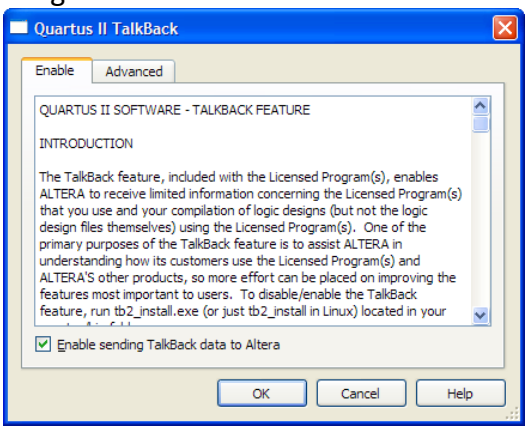


Figure 1.2

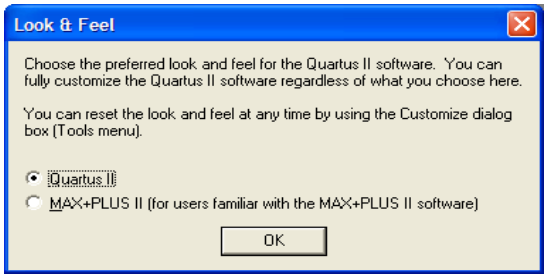


Figure 1.4

Figure 1.3

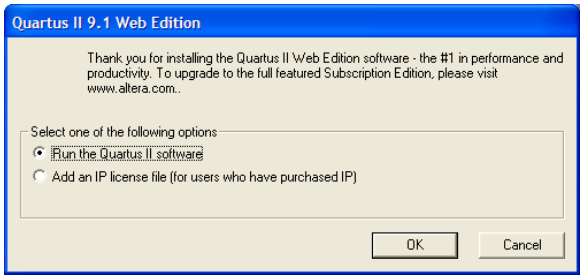


Figure 1.5

6. Select "Run the Quartus II software" when you see the dialog box given in Fig 1.5.

Congratulations! You completed the Quartus II software installation. Now we will install the USB driver for the DE2-70 board.

C. USB-BLASTER DRIVER INSTALLATION

1. Remove the DE2-70 board from the box. **Very Important: Always, always try to keep touching to the wafer of the board to a minimum. As you know, static electric charged in your body can damage the circuit elements if you fail to do so.** Turn on the DE2-70 board. Make sure that the USB cable is connected between your PC and board. **Important:** Connect the USB cable to the "BLASTER" port (the one on the top left corner) of the board. Go to Device Manager, right click to USB-Blaster, and click "Update Driver".

2. A new window will pop up. Click "Browse my computer for drivers". Select "Search for the best driver in these locations." Assuming that you installed Quartus II into its default location set the path to "C:\altera\91\quartus\drivers\usb-blaster\"

3. Click on "Finish." Congratulations! Now, you are ready to start programming your FPGA on your DE2-70 board! If you face any problem during the driver installation, please read "F. ALTERA USB BLASTER DRIVER INSTALLATION INSTRUCTIONS" at the end of the experiment 1 in the manual.

D. DE2-70 CONTROL PANEL

The DE2-70 user manual reads "The DE2-70 board comes with a Control Panel facility that allows users to access various components on the board from a host computer. The host computer communicates with the board through an USB connection. The facility can be used to verify the functionality of components on the board or be used as a debug tool while developing RTL code." In this context RTL is an acronym for "Register Transfer Level."

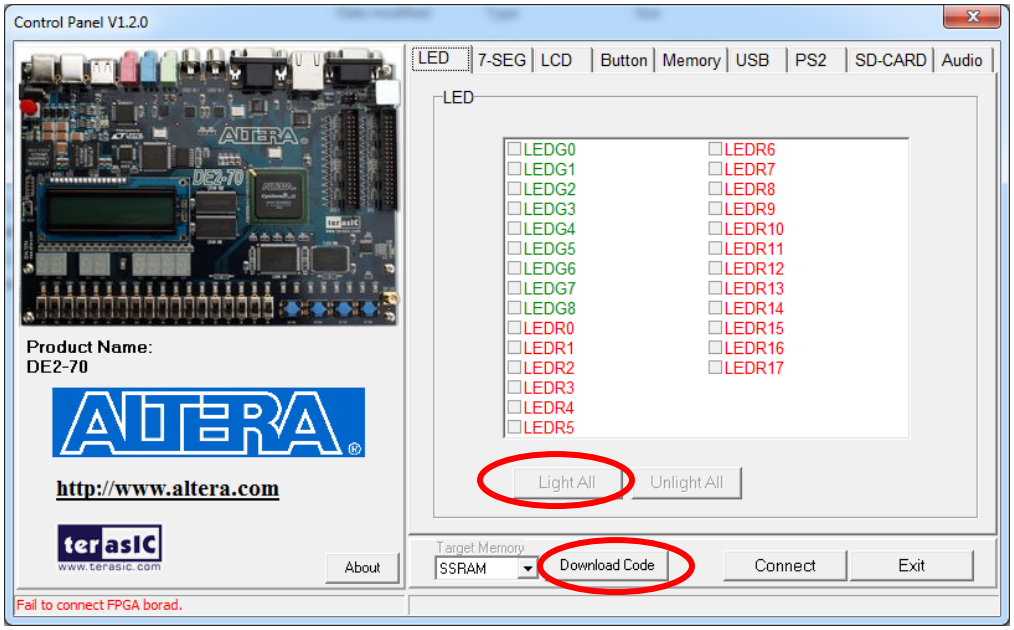


Figure 1.7

1. Go to "DE2_70_control_panel" under "ECE641 Materials" folder. Run the control panel by double clicking on "DE2_70_Control_Panel.exe" executable. You will see the screen given in Fig. 1.7. Ignore the "Fail to connect FPGA board" error.

2. Make sure the board is open and USB cable is connected properly. Click on "Download Code" button (See Fig. 1.7.) Wait for the code to be downloaded into the board successfully. Be patient, it may take up to 2 minutes to complete. When the code download is complete, click on "Connect" button.

3. If connection is successful, click on "Light All" button (See Fig. 1.7.) All the LEDs located on the bottom row of the board will lit. Navigate freely through the GUI and enjoy your first interaction with the DE2-70 board. You can use this control panel application to debug your code, understand how the peripherals on the board work, and read/write data from/to the various memory units available on the board etc.

4. Click on "Disconnect" and then on "Exit" when you are done with the control panel application.

E. FAQ FOR EXPERIMENT 1

1. Can we use an operating system (OS) such as MAC OS x or Linux based (Ubuntu) other than Windows for the FPGA experiments?

Answer: Unfortunately, you can only use Windows XP/Vista/7/8/10/11 OS because

the development environment you will use for the FPGA experiments only supports Windows XP/Vista/7/8/10/11 OS. If you have a different OS than Windows on your PC, we recommend you to install Windows using virtual machine software such as VMware (shareware), Parallels Desktop, Boot Camp, Oracle VM VirtualBox (freeware). VirtualBox is not recommended. Please note that Apple ARM-based M1 and M2 chips do not support the installation of Quartus II 9.1 USB-Blaster driver. Only Apple Intel-based chips support the installation of the driver.

2. When I connect the USB cable to my PC and the board, "The New Hardware Wizard" does not appear as shown in "C. USB-BLASTER DRIVER INSTALLATION" section. Why?

Answer: In new versions of Windows (Windows 7/8/10/11), "The New Hardware Wizard" do not appear and you need to install the driver in order to connect to board. "The New Hardware Wizard" can be started from the "Device Manager". Right click on the "Computer" icon on the desktop and click "Properties". A window will appear on the screen. Click "Device Manager" on the left menu. In the Device Manager window, you will see a tree menu. Find the node "USB-Blaster" and double click on it to start the "The New Hardware Wizard". Then, follow the procedure as instructed in the manual to install the driver. More detailed instructions for the driver installation can be found at the end of this document (see "Altera USB Blaster Driver Installation Instructions").

3. When I click to "Download Code" button on the DE2-70 Control Panel, it cannot complete the download and gives an error.

Answer: There is a software bug in the DE2-70 Control Panel and it sometimes gives this error. Unfortunately, there is no solution for this error. On the other hand, that does not mean your PC cannot communicate with the board. You will see that you are able to download and run your projects on the board.

4. I installed Quartus II 9.1 software, but it says that it is not installed.

Answer: Uninstall the software and install it again.

F. ALTERA USB BLASTER DRIVER INSTALLATION INSTRUCTIONS
https://mil.ufl.edu/3701/docs/quartus/ByteBlaster/USB-Blaster_driver_install.pdf

To install the driver for your USB Blaster cable, follow the instruction for your particular operating system.

For Windows 10/11, do the following:
1. Plug the USB-Blaster into your PC.
2. Open the Device and Printers (Control Panel | Devices and Printers).
3. Under Unspecified, USB Blaster should be listed. Right mouse click on this and then select **Properties.**
4. Select the **Hardware** tab and select **Properties.**
5. A new window should pop up with the **General** tab already selected. Select **Change Settings**.
6. Again a new window should pop up with the **General** tab already selected. Select **Update.**
7. Select **Browse my computer for driver software.**
8. Find **c:\altera\91\quartus\drivers** (**Note 1**: Your altera file is located at the location you selected when you first installed quartus. The location listed in this document is the default location) (**Note 2**: Stop at the drivers folder, i.e., do **NOT** go deeper by opening a folder within the drivers folder)
9. Select **OK**. Make sure the proper path was selected then select **Next**.
10. If the Windows security window pops up Check the **Always trust software from "Altera Corporation"** box and select **Install**.

For Windows 7/8, do the following:
1. Plug the USB-Blaster download cable into the PC. The following will appear on the screen: **Device driver software was not successfully install.** Select **Click here for details**.
2. Select **Change settings…**
3. Now cancel this (probably by selecting **Cancel**)
4. Open the Device Manager (Windows Key | Control Panel | Device Manage)
5. Under Other devices, **USB Blaster** should be listed. Right mouse click on this and then select **Update Driver Software…**
6. Select **Browse my computer for driver software**
7. Find **c:\altera\91\quartus\drivers**, then select **Next;** (**Note:** Stop at this folder, i.e., do **NOT** go deeper by opening a folder within the drivers folder)
8. Check the **Always trust …** box and then select **Install**

For Windows Vista, do the following:

1. Plug the USB-Blaster download cable into the PC. The **Found New Hardware** dialog box appears.
2. Select **Locate and install driver software (recommended)**.
3. Select **Don't search online**.
4. When you are prompted to **Insert the disc that came with your USB-Blaster**, select **I don't have the disc. Show me other options**.
5. Select **Browse my computer for driver software (advanced)** when you see the **Windows couldn't find driver software for your device**.
6. Click **Browse...** and browse to the <*Path to Quartus II installation*>**\drivers\usb-blaster** directory. Click **OK**.
7. Select the **Include subfolders** option and click **Next**.
8. If you are prompted **Windows can't verify the publisher of this driver software**, select **Install this driver software anyway** in the **Window Security** dialog box.
9. The installation begins.
10. When **The software for this device has been successfully installed** appears, click **Close**.

FPGA EXPERIMENT 2
Finite Impulse Response (FIR) Digital Filters

TABLE OF CONTENTS

A. FIR Filters and VHDL basics
B. Compilation
C. Waveform/Vector Files
D. Functional Analysis
E. Timing Analysis
F. Downloading the Image File to the FPGA
G. Questions and Report

A. FIR FILTERS AND VHDL BASICS

See and study carefully Page 167, Example 3.1 of the textbook. Also see the program "fir_gen.vhd" in the "VHDL/lab_2" folder under your "ECE641 Materials" folder.

This program realizes the discrete-time convolution operation for an FIR filter $g(n)$ with the input sequence $x(n)$ as

$$y(n) = x(n) * g(n) \tag{2.1}$$

Since the order of FIR filter in the example is 4, we have

$$y(n) = x(n)g(0) + x(n-1)g(1) + x(n-2)g(2) + x(n-3)g(3) \tag{2.2}$$

In this example, we use Daubechies 4 (DB4) filter coefficients for $g(n)$. Transfer function for a DB4 filter is given as

$$G(z) = 0.48301 + 0.8365z^{-1} + 0.2241z^{-2} - 0.1294z^{-3} \tag{2.3}$$

We will apply an impulse function with a magnitude 100 to the input of this filter. Since $X(z) = 100$ the output of the filter $y(n)$ is equal to

$$y(n) = \{\underset{\uparrow}{48.301}, 83.65, 22.41, -12.94\} \tag{2.4}$$

Let's quantize the filter coefficients to eight bits (plus a sign bit) of precision and rewrite the quantized version of the transfer function given in Eq. (2.3) as

$$\hat{G}(z) = \frac{124}{256} + \frac{214}{256}z^{-1} + \frac{57}{256}z^{-2} - \frac{33}{256}z^{-3} \tag{2.5}$$

In this example we will use a filter with a transfer function given as

$$C(z) = 124 + 214z^{-1} + 57z^{-2} - 33z^{-3} \tag{2.6}$$

and divide the output of the new filter to 256. It is straightforward to see that this operation is equal to using a filter with a transfer function given in Eq. (2.5) If the input is $x(n) = \{\underset{\uparrow}{1}00\}$ then the output of the proposed algorithm will be equal to

$$y(n) = \left[c(n) * x(n)\right] / 256 = \{48, 83, 22, -13\} \tag{2.7}$$

Let's confirm this result in MATLAB by writing

```
>> int8(conv([124 214 57 -33],100)/256)

ans =
   48   84   22   -13
```

Result checks with our calculations. Now the problem is to implement this filter using an FPGA. The circuit of this FIR filter is given in Fig. 2.1.

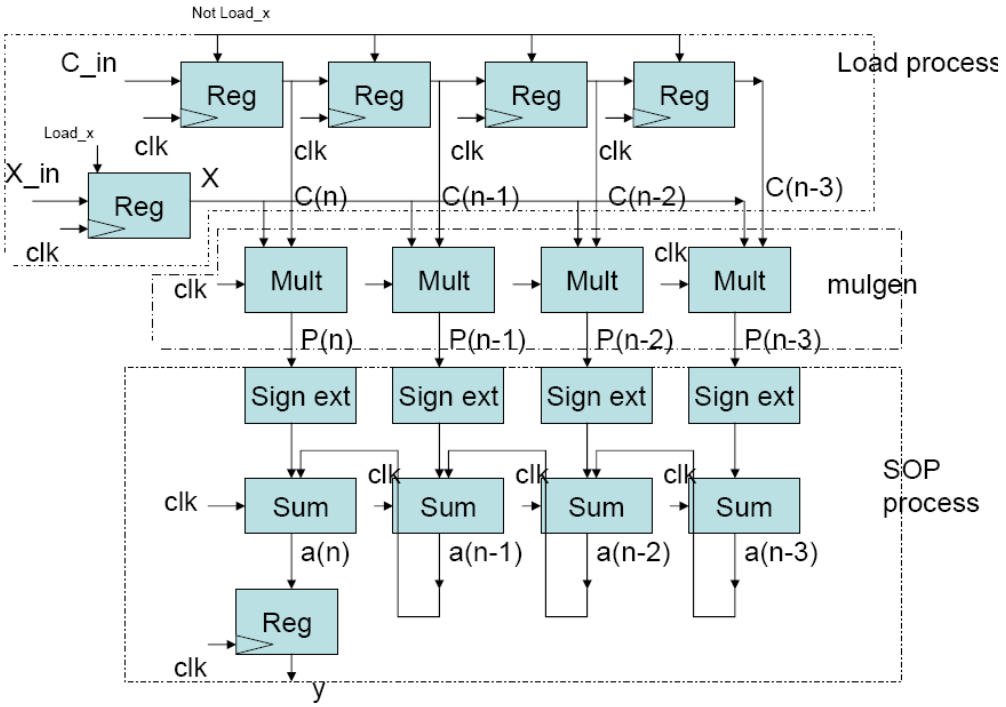


Figure 2.1: The circuit of the FIR filter.

We observe from Fig. 2.1 that the load process will store the *L-1* numbers of the input of the filter `C_in` in a `W1` width register at rising edge of `clk` when register is enabled. Since *L = 4* in this case, we have 4 registers to store the input data and the FIR filter coefficients c(3), c(2), c(1), c(0). `X_in` is stored in another register when `load_x` is high at the rising edge of `clk`.

The three pipeline steps of multiplier will multiply $c(i)$ with input x simultaneously and their product will be delayed in the SOP process and summed up. Finally, we will get

$$y(n) = (1/256)\left[x(n)c(0) + x(n-1)c(1) + x(n-2)c(2) + x(n-3)c(3)\right] \qquad (2.8)$$

Luckily the FPGA has all the elements listed above. All we need to do is to tell the compiler how to interconnect these elements together. For that purpose, we use VHDL language. For this example, the VHDL file is already written by the author of the textbook and provided in a Quartus II project file. Now we will compile this project. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

B. COMPILATION

1. Start Quartus II 9.1 software. Select "Open Project" from "File" menu. Go to the VHDL directory in your "ECE641 Materials" folder. Select the file "fir_gen.qpf" located in "VHDL/lab_2" folder and click on "Open." See Fig 2.2.

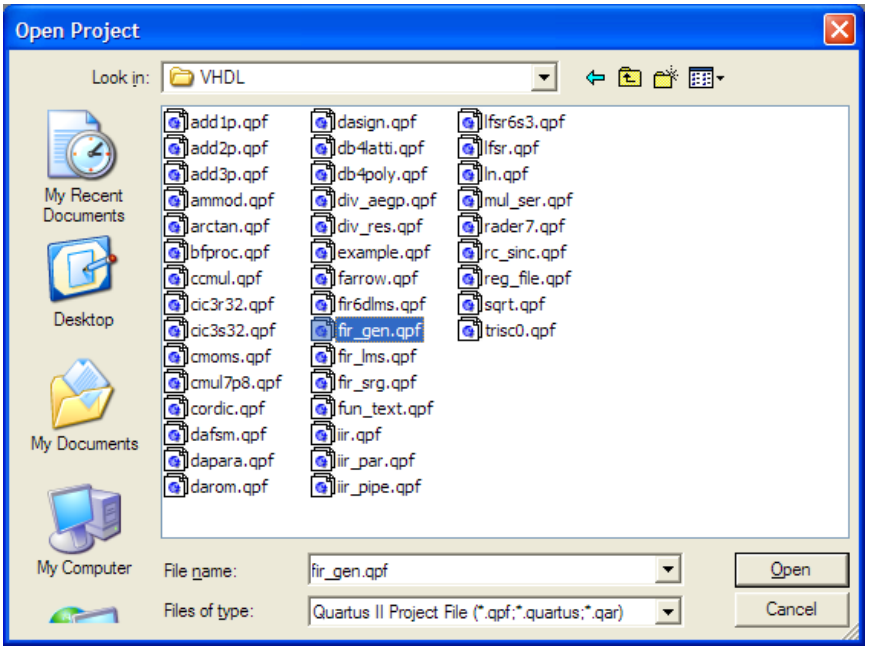


Figure 2.2

2. Double click on the "fir_gen" program in the project navigator window and then the VHDL code of "fir_gen" will be displayed. Make sure that it is the same program as the one on page 168 of the textbook. See Figs. 2.3 and 2.4.

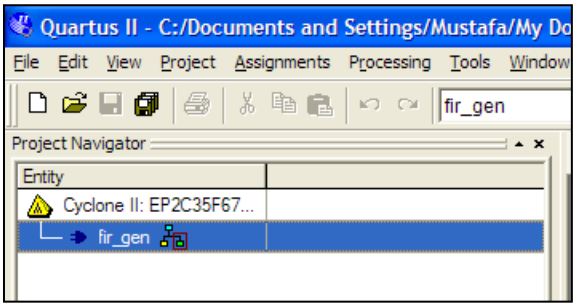


Figure 2.3

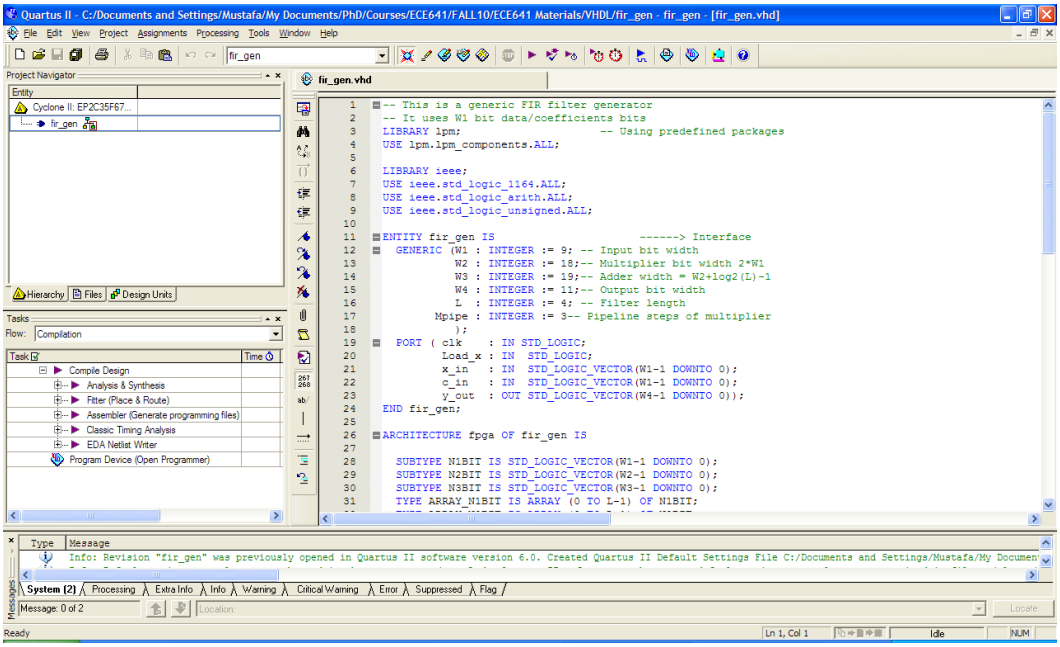


Figure 2.4

3. Double click on "CycloneII: EP2CxxFxxxxx" (original device selected in the project file) and then change it to FPGA part on your DE2-70 demo board, which is "CycloneII: EP2C70F896C6". Make sure that you pick the right FPGA chip model number. Can you tell the model number by carefully inspecting the DE2-70 board? See Fig 2.5.1 (do not include the last letter "N") and Fig 2.5.2.



Figure 2.5.1

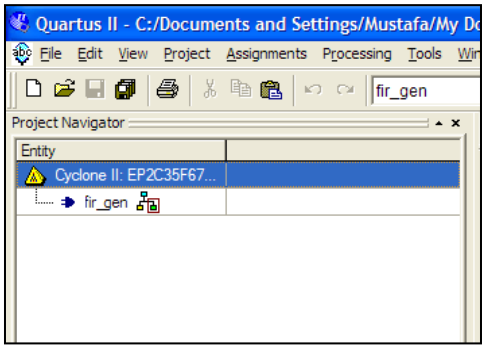


Figure 2.5.2

4. Select "Start Compilation" from the "Processing" menu. The compilation should be successful with one critical warning: "Timing Requirement is not met." See Figs. 2.6 and 2.7.

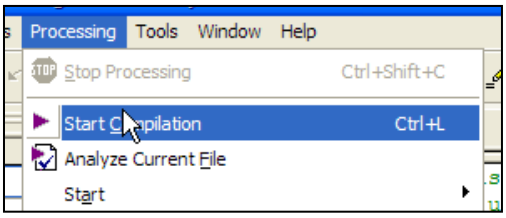


Figure 2.6

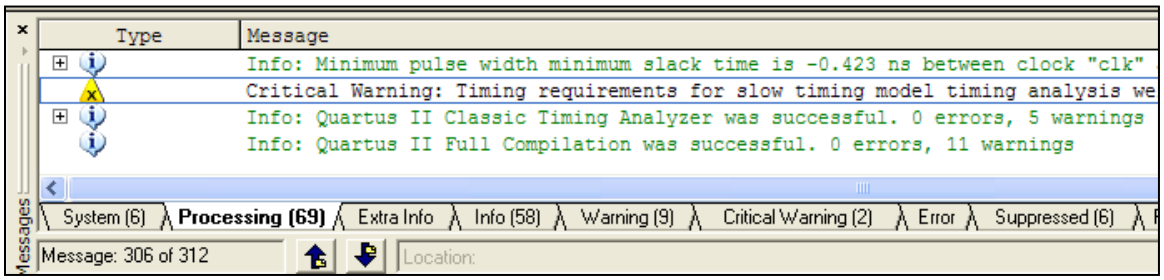


Figure 2.7

5. Select the "Settings" under the "Assignments" menu (See Fig 2.8) and change the "Fmax" from 3 ns to 10 ns in "Classic Timing Analyzer Settings" (See Fig 2.9), and run the compilation again. You will observe that the critical warning "Timing Requirement is Not Met" will disappear. Fmax specifies the minimum acceptable clock frequency, that is, the maximum clock frequency that can be achieved without violating internal setup and hold time requirements.

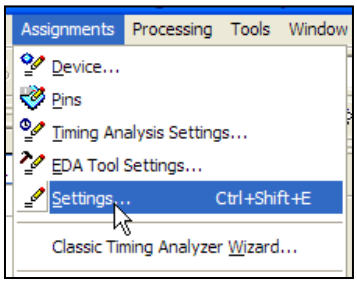


Figure 2.8

6. Make sure that the "fir_gen.sof" file is generated. It is the compiled image file which will be downloaded to the FPGA later (check "Assembler/Generated Files" in the "Compilation Report", See Fig. 2.10).

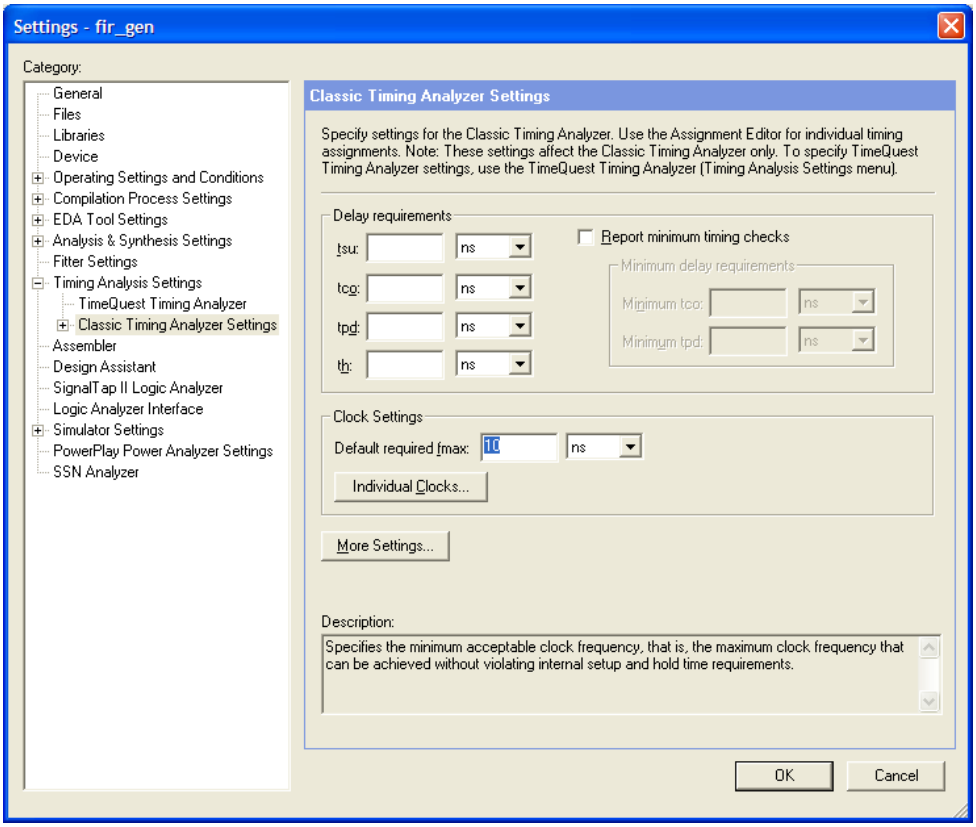


Figure 2.9

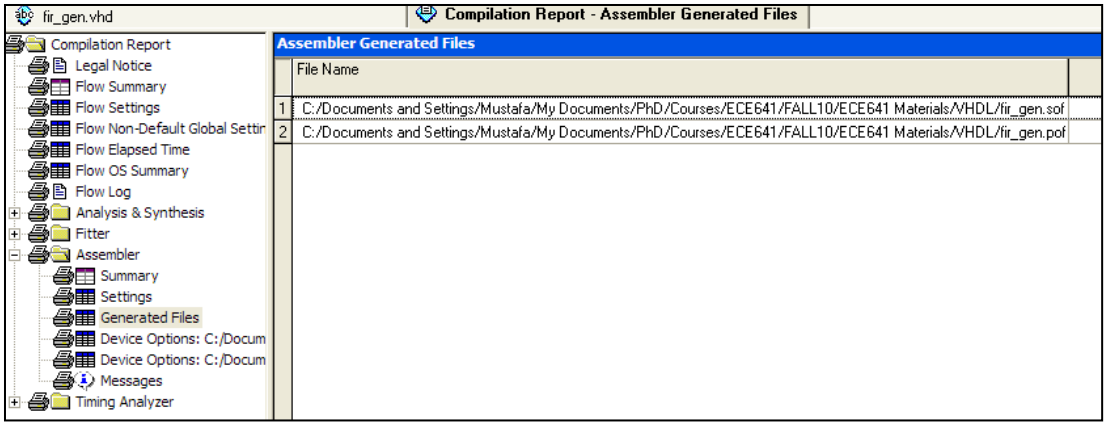


Figure 2.10

C. WAVEFORM/VECTOR FILES

Waveform/Vector files (VWF) are used for simulation purposes. Since compiling, downloading, and testing the project is a computationally extensive and time consuming process, it is much more efficient to simulate the project first. You should compile and download the project into the FPGA only when you are sure that your project is running correctly in the simulations.

Waveform/ Vector files provide us the input and output signals for the simulation. In this section we will learn how to open and edit a Waveform/ Vector file.

1. Open the input waveform file for the FIR project by selecting “Open” in the “File” menu. Select the file type as “Waveform/Vector Files” in the dialog window. And then select “fir_gen.vwf” which exists in your VHDL directory. See Fig. 2.11. When you see the message box given in Fig 2.12, click on “Don’t show this again” and continue.

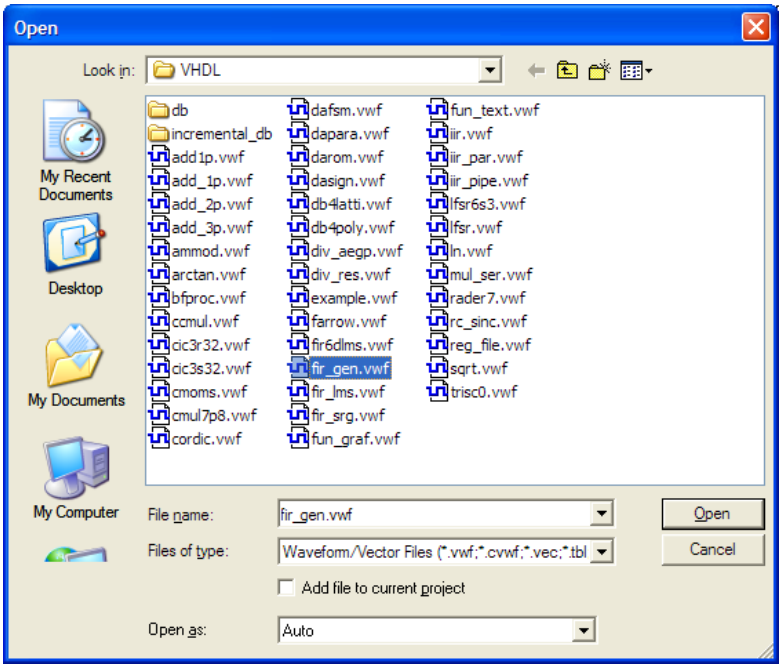


Figure 2.11

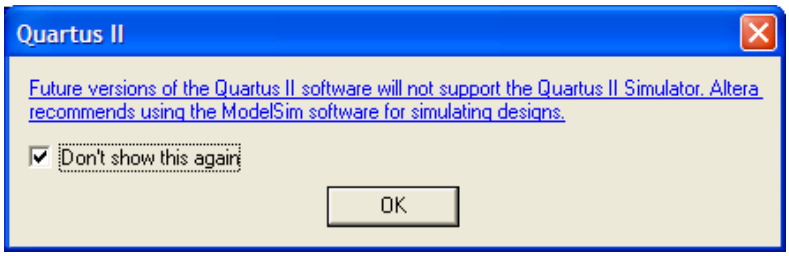


Figure 2.12

2. Editing the waveform file is an important step. Make sure that `load_x`, `x_in`, `c_in`, `y_out`, and `clk` signals are available in the waveform file. Also, the `clk` should be less than "Fmax" defined in your timing requirements (100MHz).

Now, we will add c(0), c(1), c(2), c(3) coefficients. Right click on clear part of the window, and select "Node or Bus…" under "Insert" menu (See Fig. 2.13.) Click on "Node Finder" button. Then, pick "Post-synthesis" from the "Filter" pull-down menu and click on "List" button. From the list on the left, add c[0], c[1], c[2], c[3] to the right window. And Click on "OK" button to close the dialog box. Before closing the "Insert Node or Bus" dialog box make sure that the "Radix" option is chosen as "Signed decimal". Click again on "OK" button and close the dialog box. Then, save the waveform file that you have just edited by selecting "Save" from "File" menu. Your waveform file should look like the one given in Fig 2.14.

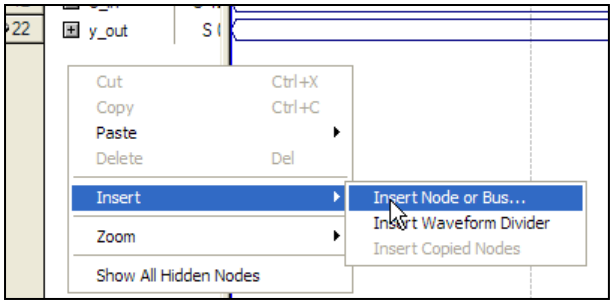


Figure 2.13

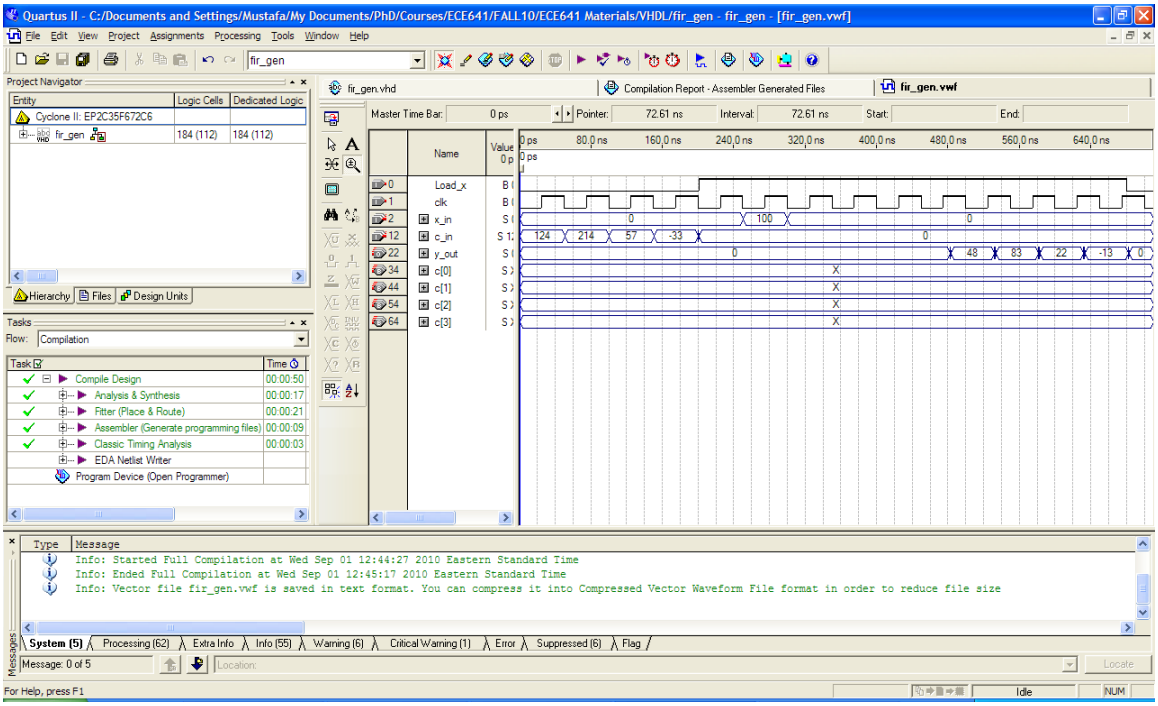


Figure 2.14

D. FUNCTIONAL ANALYSIS

Now we have the VHDL file compiled and the Waveform file ready. In this section we will simulate the project in "Functional Analysis" mode, which is not 100% realistic, because it does not take "timing" into consideration (Remember that every digital circuit element has a delay!). However, it is still very useful to check if our project is "functioning" correctly.

1. Bring up the simulator tool by clicking on "Simulator Tool" in the "Processing" menu (See Fig 2.15.)

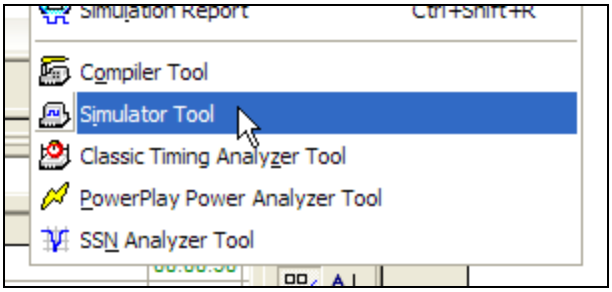


Figure 2.15

2. Make sure that the "Simulation mode" is set as "Functional", and the "Simulation input" is selected as "fir_gen.vwf." Then, click on "Generate Functional Simulation Netlist" button. After it is successfully generated click on "Start" that is in the left bottom of the simulator tool (See Fig 2.16.). This will start the functional simulation.

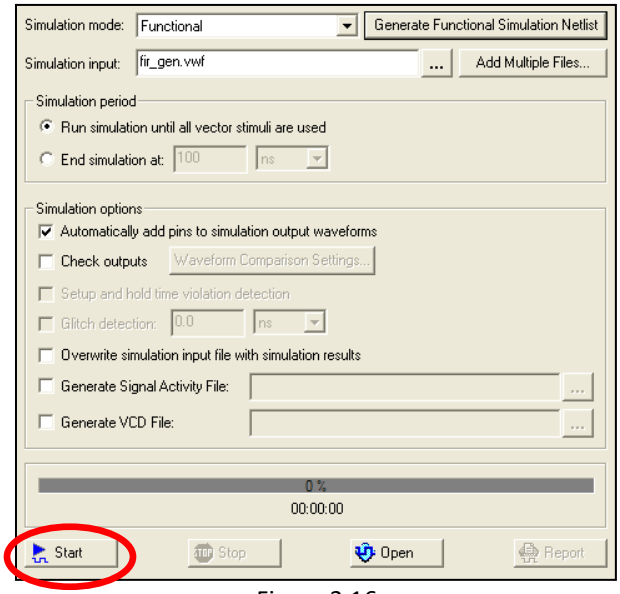


Figure 2.16

3. When the function simulation is completed, click on "Report" button on the right side and check the filter output `y_out`. You should see that it is [48, 83, 22, -13] and the first value occurs four clocks after 4 after `x_in` is 100, the filter input (See Fig 2.17.) It is due to the fact that it takes 4 cycles to make these calculations. Note that in the function simulation, all the data such as "c(n)" appear right after the rising clock edge. This is due to the fact that functional analysis assumes that the time delay for the circuit elements is zero. This is different from the timing simulations that will be performed next.

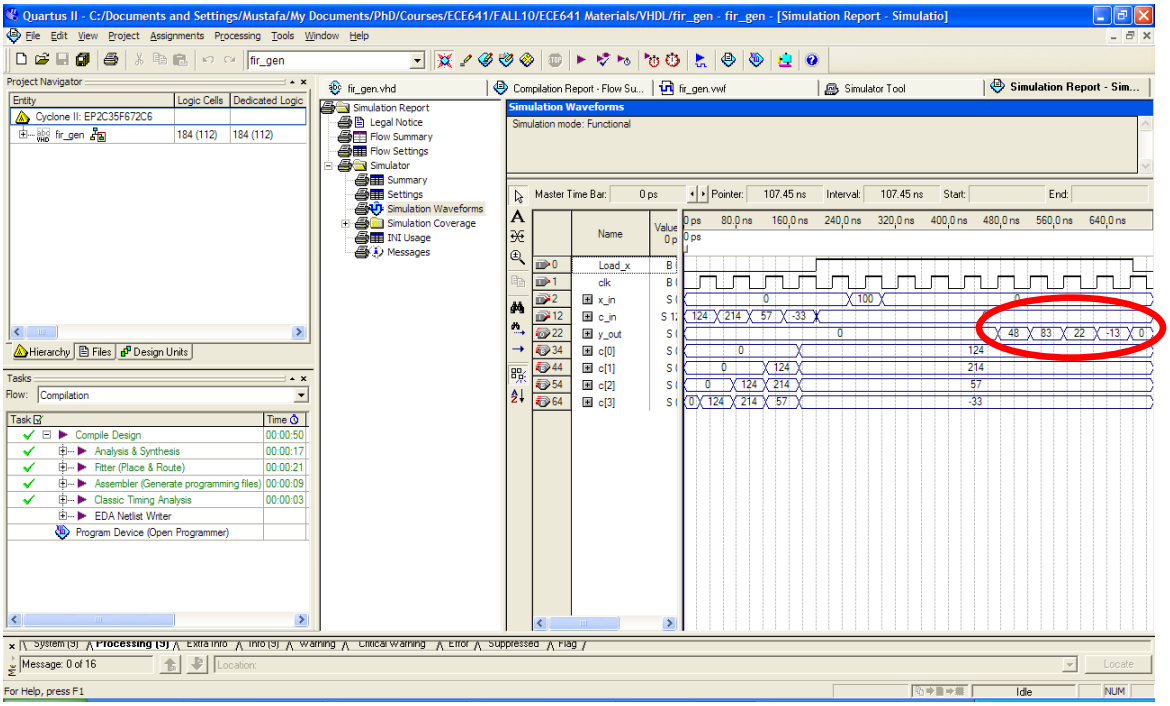


Figure 2.17

E. TIMING ANALYSIS

In this type of simulation, we will not only check if our project is running "functionally" but also make sure that the timing of the circuit is proper.

1. Go back to the "Simulator tool" window and set the simulation mode to "Timing". Then click on "Start" button to start the simulation.

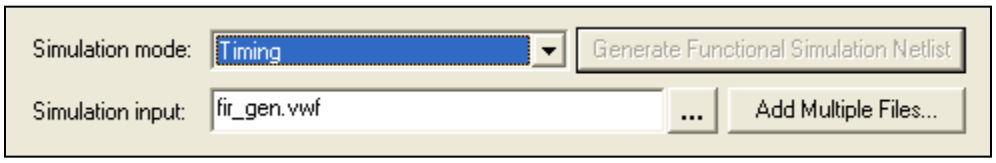


Figure 2.18

2. After the timing simulation is completed, click on "Report" button in the bottom right of your screen. The "Timing Simulation Report" should look like the one given in Fig 2.19.

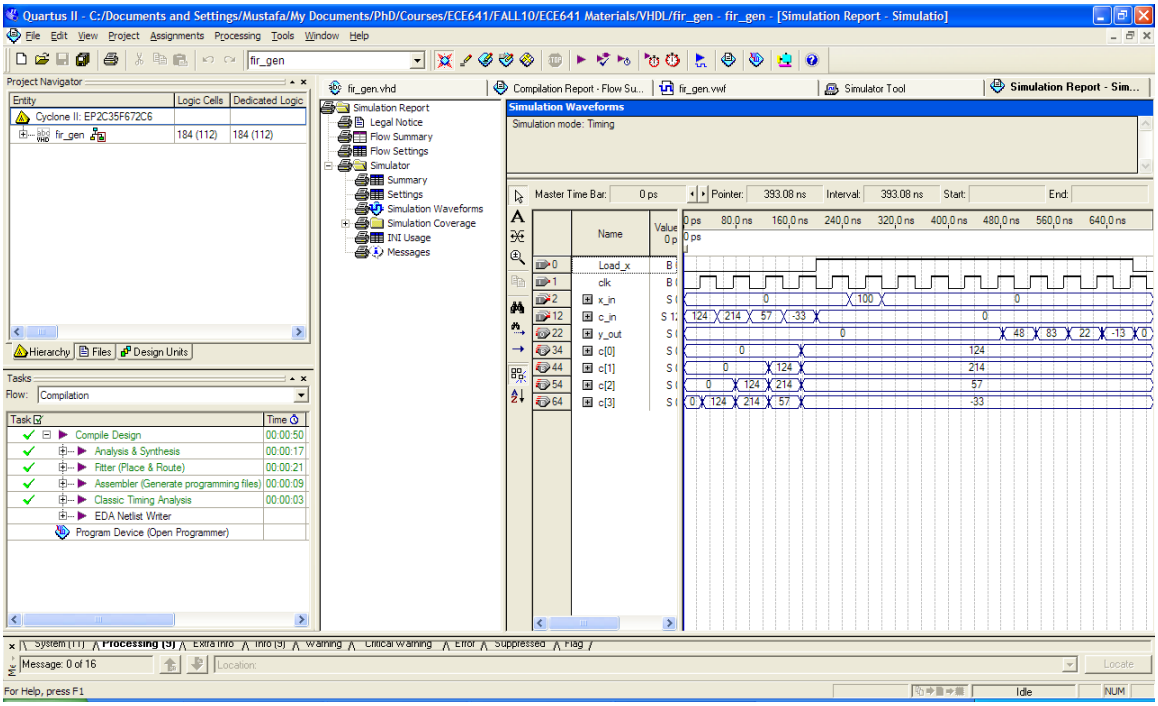


Figure 2.19

3. Now we will measure the Tco of c(3). Tco is the time needed for an output to be stable relative to the rising edge of the clock. Add two timing bars and measure their relative timing period from the clock rising edge and the data start edge of 124. In order to do that you need to right click on top of the simulation report and select "Insert Time Bar…" from "Time Bar" menu (See Fig 2.20.)

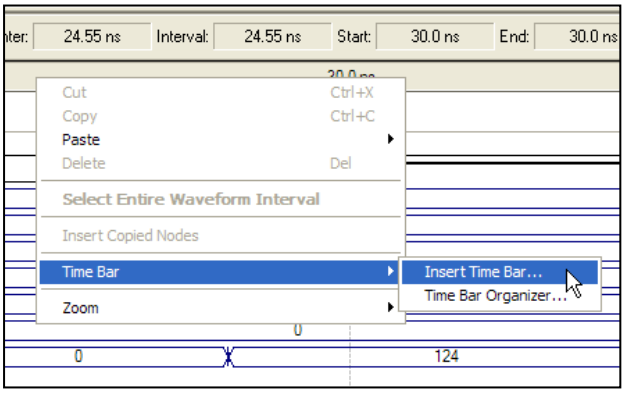


Figure 2.20

When adding the first "Time Bar," make sure that it is the "Master Time Bar." Select "Make master time bar" on the dialog box (See Fig 2.21.) You should read a Tco for c(3) which is around 3ns (See Fig 2.22.) Record the exact value to include it in your report.



Figure 2.21

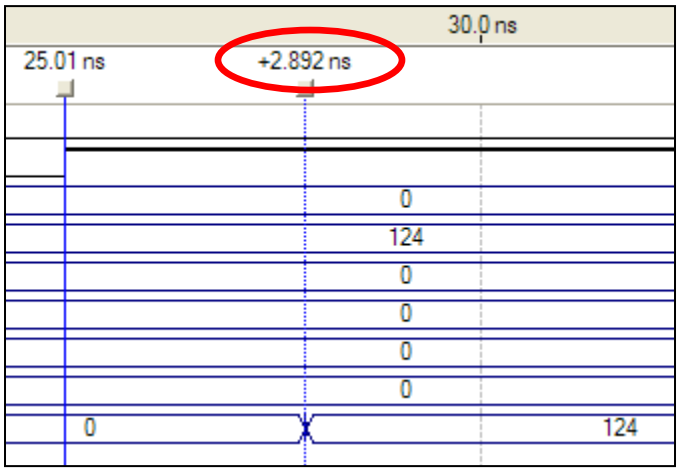


Figure 2.22

F. DOWNLOADING THE IMAGE FILE TO THE FPGA

We compiled the project, simulated it using both functional and timing analysis tools and now we are ready to send (download) the compiled file, the "image," into the FPGA.

1. Select the "Programmer" in the "Tools" menu (See Fig 2.23.)

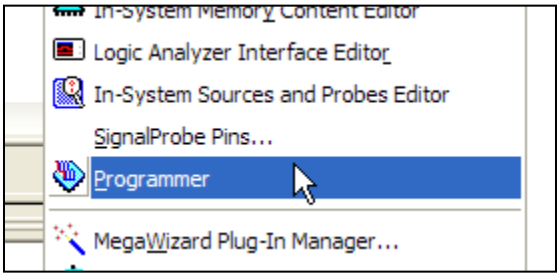


Figure 2.23

2. Turn on the power of your DE2-70 board. Then, in the "Programmer" window, click on the "Hardware Setup" and select "USB-Blaster [USB-0]". Then, select the "JTAG" in the "Mode" pull-down menu. Now, click on "Add File" and select "fir_gen.sof" that you have generated earlier. If it is already in the list, you don't need to add it again. Make sure that you have checked "Program/Configure" in the list. See Fig. 2.24.

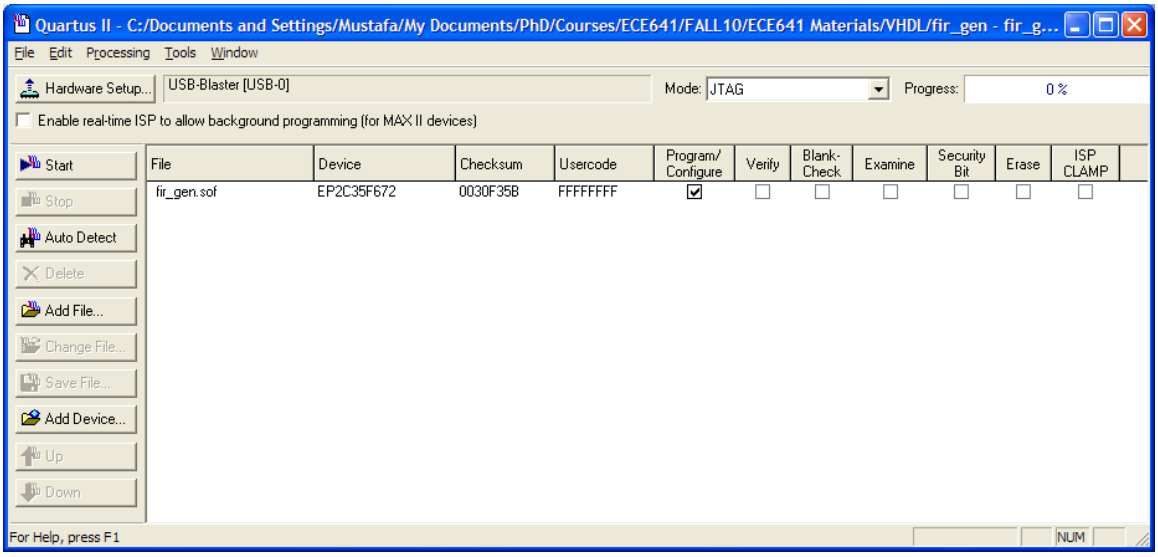


Figure 2.24

3. Click on "Start" in the programmer window. The "Status Window" should give you the current status report. See Fig. 2.25.

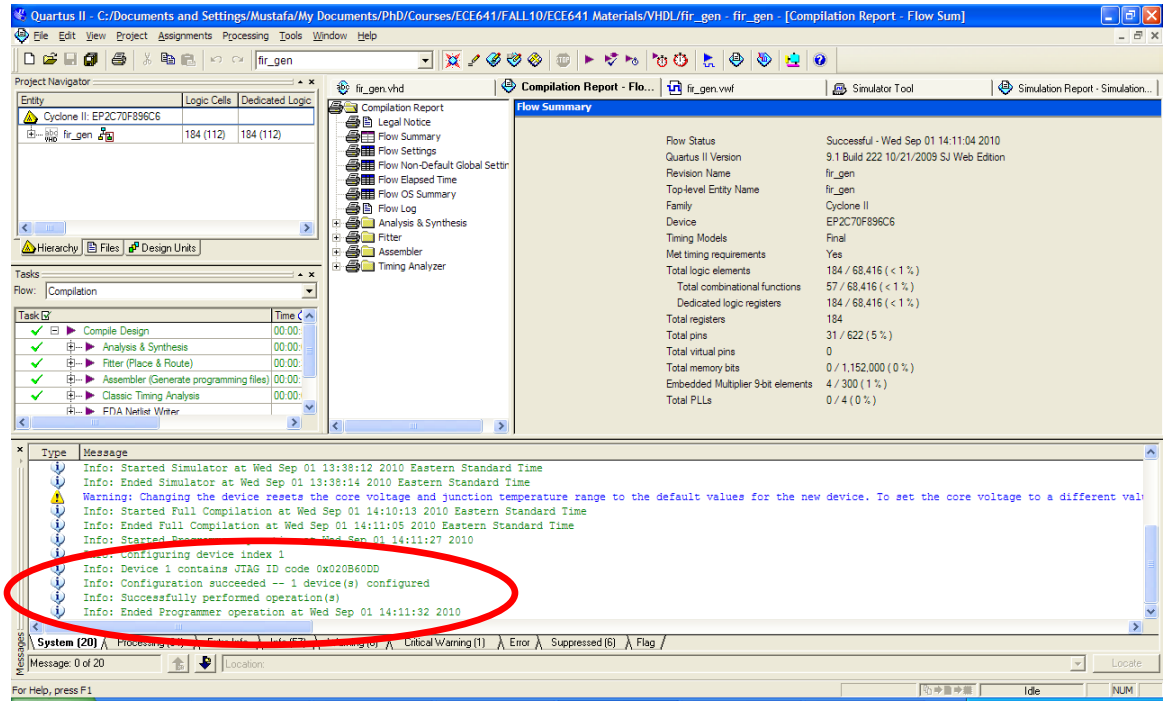


Figure 2.25

Congratulations! You have just turned your FPGA into a 4-tap digital FIR filter. The FPGA is doing its job but you can't see it on the board. This is due to the fact that we are not using the peripherals provided with the DE2-70 board yet.

G. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. Derive Eq. 2.4
2. What is the difference between functional analysis and timing analysis?
3. In the simulations why do you think we needed 100 as the input? What would happen if we set the input to 1? Try to edit the waveform file, run the functional analysis, and see the result. What is the theoretical result? Why is the simulation result different than the theoretical result? Here is how to modify the input in the waveform file:

    a. Double click on the value 100 (See Fig. 2.26.)
    b. Enter the new value (In this case 1) in "Numeric of named value" field (See Fig. 2.27.) Click on "OK" button.
    c. Save the waveform file.

Make sure that you include the question numbers to your answers. Don't hesitate to get in touch with the instructor if you have any questions regarding the report.

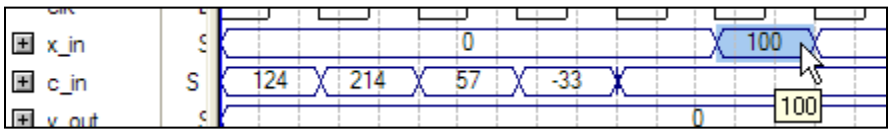


Figure 2.26

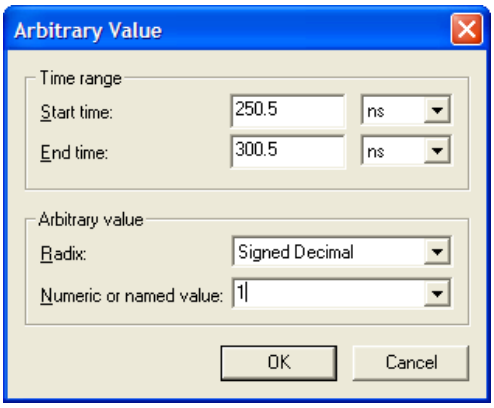


Figure 2.27

H. FAQ FOR EXPERIMENT 2

1. How can I put the "Time Bar" to a certain point on the waveform? Because it is usually not possible to put it to right spot with drag and drop.

Answer: Unfortunately, it is not easy to put it to right sport with drag and drop. You need to guess the time of the spot correctly by examining the waveform when you are creating the time bar. If it cannot guess the time correctly, you should delete that time bar and create a new one with a better time input.

2. What are Fmax and tco?

Answer: Fmax specifies the minimum acceptable clock frequency, that is, the maximum clock frequency that can be achieved without violating internal setup and hold time requirements. Generally, the slowest part or operation of the design (such as a division operation) specifies the Fmax. Tco is the time needed for an output to be stable relative to the rising edge of the clock. It is basically the maximum time required to obtain a valid output at an output pin after a clock transition at an input clock pin, directly or through a register.

3. I downloaded the given FIR code to the board but I did not see anything that shows me it is running on the board.

Answer: The FPGA is doing its job but you can't see it on the board. This is due to the fact that we are not using the peripherals provided with the DE2-70 board yet. You will learn how to use the peripherals in Experiment 5.

4. When I wanted to perform Timing Analysis, Quartus gives me an error.

Answer: For Timing Analysis, you need to compile the design rather than doing just Analysis and Synthesis. On the other hand, it is enough to do just Analysis and Synthesis for Functional Analysis.

FPGA EXPERIMENT 3
Infinite Impulse Response (IIR) Digital Filters

TABLE OF CONTENTS

A. IIR Filter Implementation
B. Direct Realization
C. Look-Ahead Pipeline Realization
D. Parallel Processing Realization
E. Questions and Report
F. Optional Homework

A. IIR FILTER IMPLEMENTATION

In this experiment we will design an IIR low-pass filter. The transfer function of the filter is given as

$$H(z) = \frac{z^{-1}}{1 - 0.75z^{-1}} \tag{3.1}$$

We will implement this filter by utilizing three different approaches: direct realization, pipeline, and parallel processing. Students are encouraged to read the corresponding material in the textbook to understand the differences between these three methods. Students will also find other methods in the textbook which are not mentioned here. In this experiment, we will examine the differences of aforementioned methods in terms of circuit architecture, required power, speed, and required resources.

You will also learn how to use the “RTL viewer”, the “fitter”, the “timing analyzer”, and the “power analyzer” in Quartus II software. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

The project files for these three different methods are located in your “VHDL/lab_3” directory under “ECE641 Materials” folder. For Direct realization, use “iir.qpf”, for pipeline realization use “iir_pipe.qpf”, and for parallel processing realization use “iir_par.qpf.”

B. DIRECT REALIZATION

1. Open “iir.qpf” file in Quartus II. Set the correct FPGA model number. Select “Start Analysis & Synthesis” under “Start” menu under “Processing” menu. See Fig 3.1.

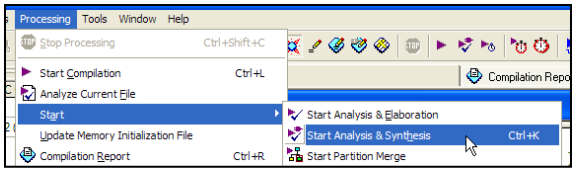


Figure 3.1

2. Now we will check how the VHDL code is realized as a circuit in the register transfer level (RTL). Select "RTL Viewer" under "Netlist Viewers" menu under "Tools" menu. See Fig 3.2.

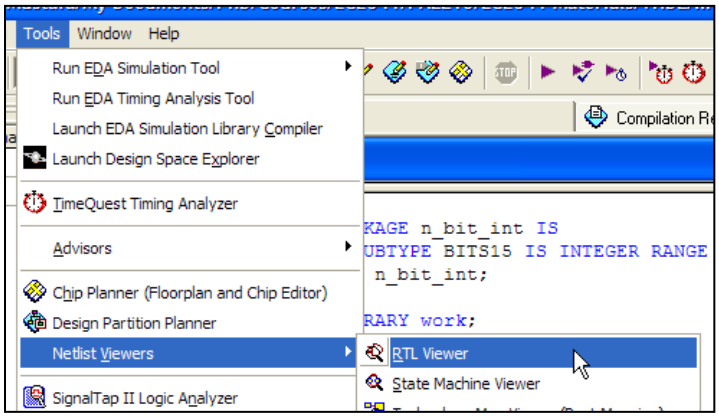


Figure 3.2

The "RTL Viewer" window will come up. In Quartus II, sometimes you may want to "Detach" the window to focus on the content by clicking on "Detach Window" button in the left tool box. It is also possible to "fit" the content into the window you are working on by clicking on "Fit in Window" under "View" menu. See Fig 3.3 for the detached and fitted RTL view of the VHDL code "iir.vhd."

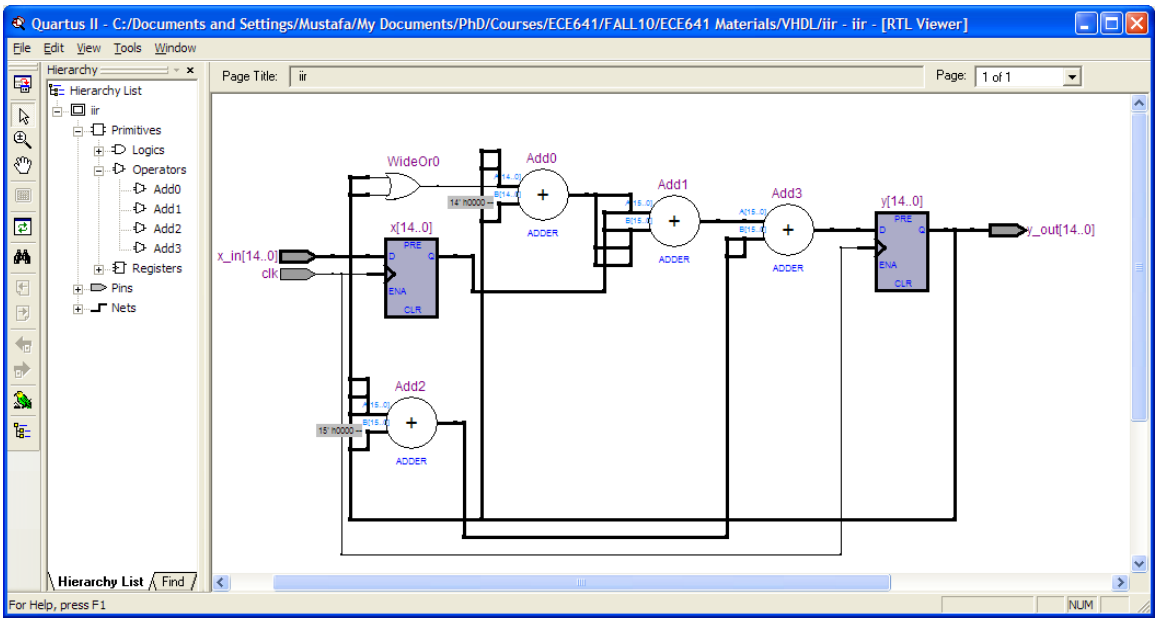


Figure 3.3

In order to see the relationship between your VHDL program and the circuit schematic, you can select any part of the circuit, right click on it, and select “Locate in Design File” under “Locate” menu. Quartus II will lead you to the corresponding VHDL statement and mark the statement. Try this for “Add1” component. See Figs. 3.4 and 3.5.

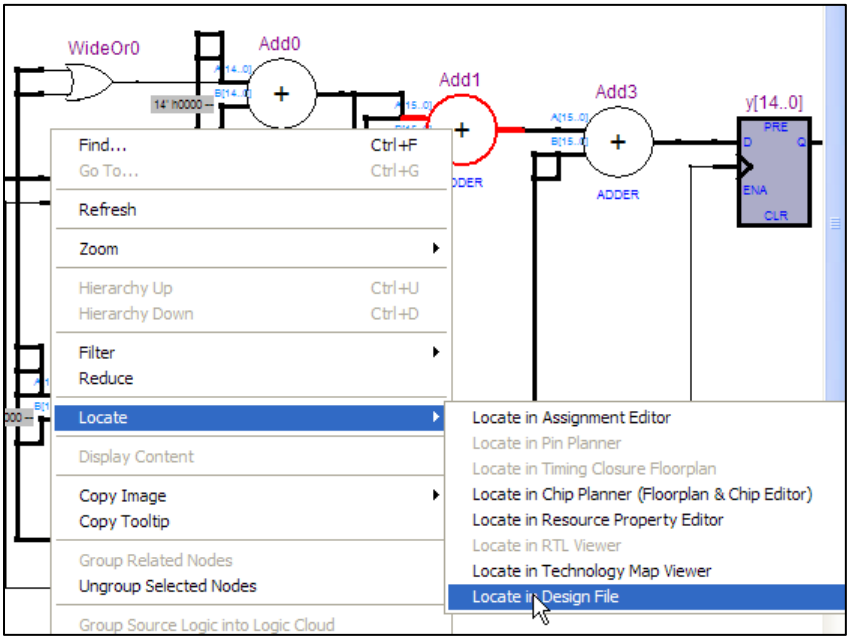


Figure 3.4

```
20    SIGNAL x, y : BITS15 := 0;
21
22  BEGIN
23
24    PROCESS     -- Use FF for input and recursive part
25    BEGIN
26      WAIT UNTIL clk = '1';
27      x  <= x_in;
28      y  <= x + y / 4 + y / 2;
29    end process;
30
31    y_out <= y;           -- Connect y to output pins
```

Figure 3.5

3. Compile your project. Make sure you have chosen the correct FPGA model. The model number is printed on the chip on your DE2-70 Board.

4. Check the resources used by the FPGA. In order to do that, click on summary of “Fitter” and find out how much computational resource you are using for this design, such as “Logic Elements (LE)” and “Registers.” See Fig 3.6.

5. Now we will find out the “fmax” of register performance. In order to do that, select “TimeQuest Timing Analyzer” under “Tools” menu (See Fig 3.7.) If you see the dialog box given in Fig 3.8 click on “Yes.” Double click on “Report Fmax Summary” in the “Tasks”

pane in the "TimeQuest Timing Analyzer" window (See Fig 3.9.) You should get the "Fmax" for registers. It should be around 170 MHz. Write down the exact value to include it in your report. You may now close the "TimeQuest Timing Analyzer" window.

Flow Status	Successful - Thu Sep 02 12:20:12 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	iir
Top-level Entity Name	iir
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Met timing requirements	No
Total logic elements	73 / 68,416 (< 1 %)
Total combinational functions	62 / 68,416 (< 1 %)
Dedicated logic registers	30 / 68,416 (< 1 %)
Total registers	30
Total pins	31 / 622 (5 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 3.6

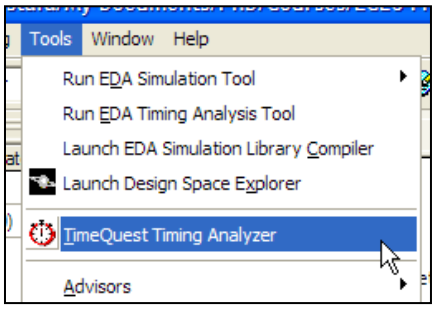


Figure 3.7

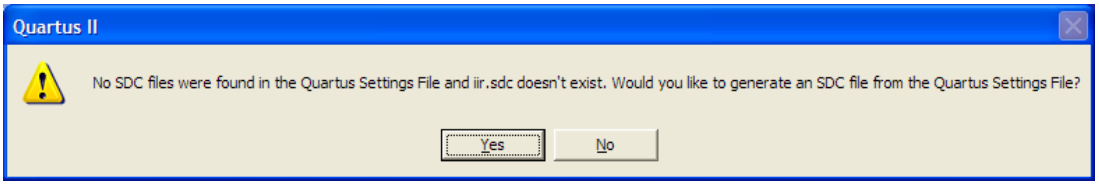


Figure 3.8

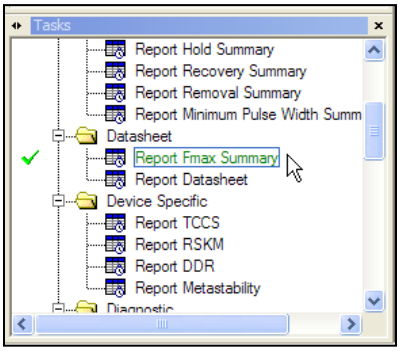


Figure 3.9

6. Now we will find out the "worst case Tco." In order to do that, go back to "Compilation Report" window and check "Timing Analyzer Summary" (See Fig 3.10.) The value should be around 7 ns. Write down the exact value to include it in your report.

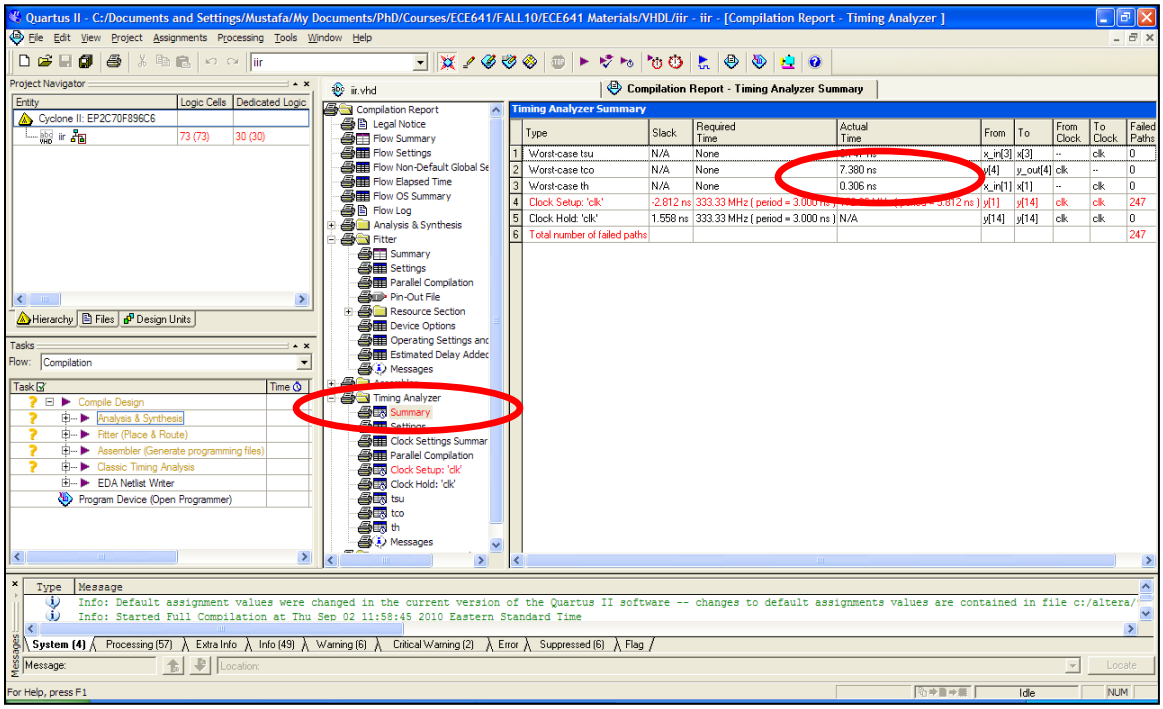


Figure 3.10

7. Now we will find out the power consumption of the design. Select "PowerPlay Power Analyzer Tool" under the "Processing" menu. See Fig 3.11. Click on "Start" in the coming "PowerPlay Power Analyzer Tool" window. Then, click on "Report" on the same window (See Fig. 3.12.) You should read a "total thermal power dissipation" which is around 250 mW. Write down the exact value for your report. See Fig 3.13.

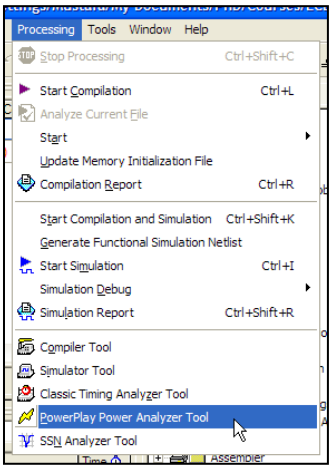


Figure 3.11

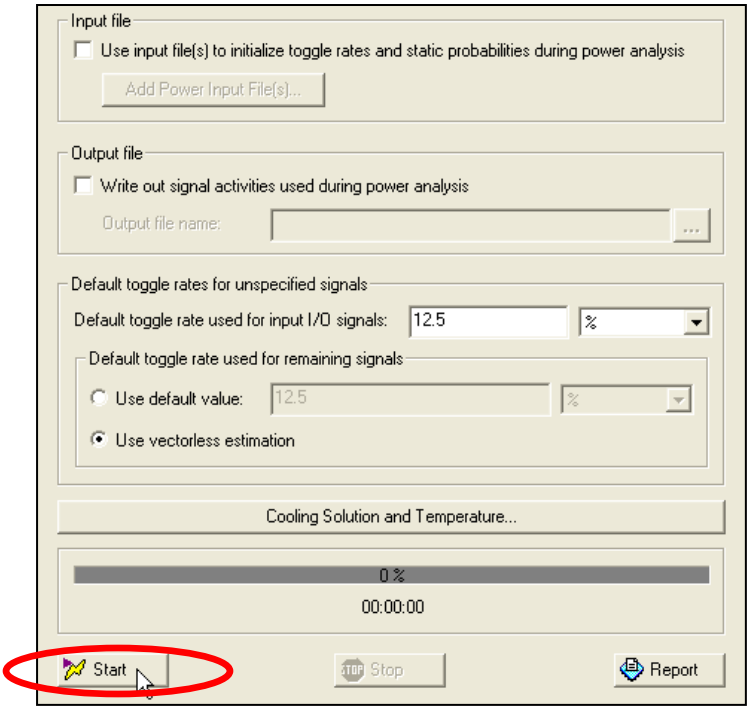


Figure 3.12

PowerPlay Power Analyzer Status	Successful - Thu Sep 02 12:21:49 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	iir
Top-level Entity Name	iir
Family	Cyclone II
Device	EP2C70F896C6
Power Models	Final
Total Thermal Power Dissipation	257.80 mW
Core Dynamic Thermal Power Dissipation	20.75 mW
Core Static Thermal Power Dissipation	155.17 mW
I/O Thermal Power Dissipation	81.88 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Figure 3.13

C. LOOK-AHEAD PIPELINE REALIZATION

Close the project file and open the project named “iir_pipe.qpf.” Follow the steps given in Section B for this project and record the “total number of LE’s”, “worst case Tco”, “fmax,” and “total thermal power dissipation” for this project.

D. PARALLEL PROCESSING REALIZATION

Close the project file and open the project named “iir_par.qpf.” Follow the steps given in Section B for this project and record the “total number of LE’s”, “worst case Tco”, “fmax,” and “total thermal power dissipation” for this project.

E. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures **excluding the codes** in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment. **Your report should answer the questions raised in the previous sections**. Also provide following three sections in your report:

1. Comparison: Provide a table such as the one in given in Table 3.1. Fill in all the cells in the table. Summarize the advantages and disadvantages of each implementation method based on your observations.

Implementation Method	Fmax	Worst Case Tco	Resources	Total Thermal Power Dissipation
Direct				
Pipeline				
Parallel				

Table 3.1

2. IIR Filter: Consider the following transfer function for a low-pass IIR filter

$$H(z) = \frac{z^{-1}}{1 - 0.6z^{-1}} \tag{3.2}$$

Do the necessary calculations to implement this filter using the look-ahead pipeline method. Refer to Section 4.4.1 of the textbook on page 231.

3. Coding: Implement this filter in VHDL. Refer to Example 4.3 of the textbook on page 231. You may modify the iir_pipe.vhd file. Provide the complete code with your report. Report the speed, power, and required resources for your filter.

4. Speed Up: Refer to "fractional coding" on page 59. Come up with a solution to speed up your implementation via fractional coding. Provide the code and compare your new design with the old design in terms of speed and resources.

F. OPTIONAL HOMEWORK

Use direct implementation to design a 10^{th} order Butterworth IIR Filter by designing five second-order systems as given in Example 4.2 of the textbook on page 227. Report speed, power, and required resources of your Butterworth IIR Filter design. Do a functional analysis to test your design. Though this homework is optional and there is no due date, it is for your own benefit, and the students are encouraged to do it.

G. FAQ FOR EXPERIMENT 3

1. I cannot get the Fmax using the TimeQuest Timing Analyzer. When I double clicked on the "Report Fmax Summary" as shown in the Fig. 3.9 in the manual, nothing appeared on the screen.

Answer: You need to compile the design rather than doing just Analysis and Synthesis.

2. I cannot find the "Timing Analyzer Summary" in the Compilation Report as shown in Fig. 3.10.

Answer: You need to compile the design rather than doing just Analysis and Synthesis.

3. Which one in Fig. 3.6 shows the total logic element (LE) of the design?

Answer: The total logic element of the design is shown by the number near the "Total logic element" field in Fig. 3.6. You do not need to report the "Total combinational functions" and "Dedicated logic registers" fields.

4. I am not getting close Logic Element (LE) and Fmax values with the ones in the lab manual for particular design. Why?

Answer: Make sure you have chosen the correct FPGA model.

FPGA EXPERIMENT 4
Two-Channel Orthogonal Filter Bank

TABLE OF CONTENTS

A. Introduction
B. Polyphase Decomposition
C. Lattice Filter Realization and the Critical Path
D. Questions and Report

A. INTRODUCTION

Two-channel orthogonal filter banks play an important role in multi-rate signal processing. Generally they are used to design more complex filter banks as well as to generate wavelets. An example block diagram of a two-channel filter bank is given in Fig. 4.1.

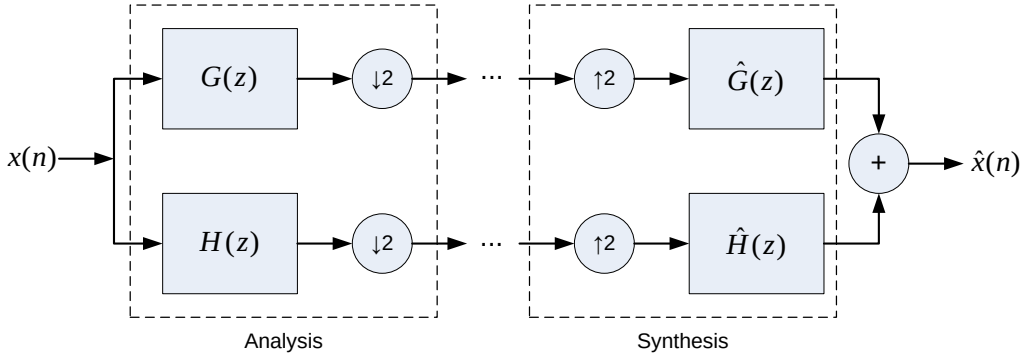


Figure 4.1

In Fig. 4.1 the circles with up-arrows and down-arrows represent up-samplers and down-samplers respectively. The number “2” in the circle indicates that the sub-sampling is done by 2, meaning even indexed samples are kept and odd indexed samples are discarded in the down-sampling case.

In this experiment we will implement the “analysis” part of the filter bank, using two different methods: Polyphase Decomposition and Lattice Filter Realization. We are going to use “db4poly.qpf” located in “VHDL/lab_4/db4poly” folder and “db4latti.qpf” located in “VHDL/lab_4/db4latti” folder in your “ECE641 Materials” folder. We will compare the two designs; highlight the strengths and weaknesses of each method.

For the first time we will go into details of a VHDL code and also learn how to use “RTL Viewer” to trace the “critical path” which helps designers to improve the timing of the design at hand.

B. POLYPHASE DECOMPOSITION

Refer to Example 5.1 in the textbook on page 250 for this experiment. Details of polyphase decomposition can be found in Section 5.2 of the textbook. In this experiment we will obtain and implement the polyphase decomposition of a DB4 filter with the transfer function

$$G(z) = 0.48301 + 0.8365z^{-1} + 0.2241z^{-2} - 0.1294z^{-3} \tag{4.1}$$

It is possible to rewrite the transfer function given in Eq. (4.1) as

$$\begin{aligned} G(z) &= G_0(z^2) + z^{-1}G_1(z^2) \\ &= \left(\frac{124}{256} + \frac{57}{256}z^{-2}\right) + z^{-1}\left(\frac{214}{256} - \frac{33}{256}z^{-2}\right) \end{aligned} \tag{4.2}$$

This is called as “polyphase decomposition” and it is implemented for this specific filter by the pre-written VHDL code named “db4poly.vhd” available in “VHDL/lab_4/db4poly” folder. Now we will run simulations and check the performance of this design.

1. Open Quartus II software and open the project named “db4poly.qpf” under the VHDL folder in your “ECE641 Materials” folder.

2. Set the correct FPGA chip model number.

3. Compile the project.

4. Check the total number of logic elements in the compilation report. Check the Fmax using the “TimeQuest Timing Analyzer” under “Tools” menu. Refer to previous experiments if you have trouble with these steps. Write down the exact values to include it in your report. We will compare these values to the ones of the other method, lattice filter realization.

C. LATTICE FILTER REALIZATION AND THE CRITICAL PATH

Refer to Example 5.24 in the textbook on page 324 for this experiment. A two-channel orthogonal filter bank obeys the condition given as

$$H(z) = z^{-N}G(-z^{-1}) \tag{4.3}$$

where N is the filter length, $H(z)$ and $G(z)$ are the transfer functions of each filter in the bank. In this example we will use the same DB4 filter for $G(z)$. Therefore, the transfer functions of these two filters are given as

$$G(z) = 0.48301+0.8365z^{-1}+0.2241z^{-2}-0.1294z^{-3}$$
$$H(z) = 0.1294+0.2241z^{-1}-0.8365z^{-2}+0.48301z^{-3} \quad (4.4)$$

It is possible to realize this filter bank using a two stage lattice filter. Let's rewrite Eq. (4.4) as

$$G(z) = s\left(1+a[0]z^{-1}-a[0]a[1]z^{-2}+a[1]z^{-3}\right)$$
$$H(z) = s\left(-a[1]-a[0]a[1]z^{-1}-a[0]z^{-2}+z^{-3}\right) \quad (4.5)$$

where values of s, $a[0]$, and $a[1]$ are equal to 0.48, 1.73 and -0.27 respectively. When we quantize (as we always do) values of s, $a[0]$, and $a[1]$ we get $124/256$, $2^{1}-2^{-2}-2^{-6}-2^{-8}$ and $-\left(2^{-2}+2^{-6}+2^{-8}\right)$ respectively. The block diagram of the lattice filter is given in Fig. 4.2.

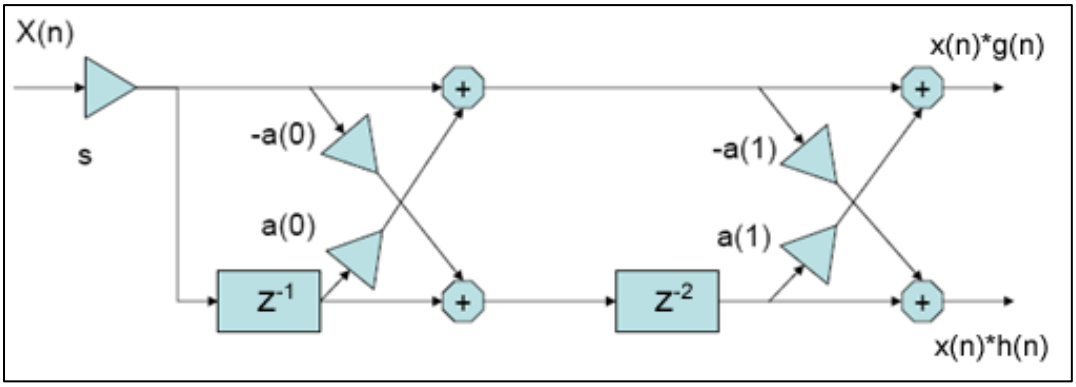


Figure 4.2

1. Now let's study the pre-written VHDL code for the implementation of this filter. Open "db4latti.vhd" file located in "VHDL/lab_4/db4latti/" in Quartus II. Examine the code line by line. In the VHDL code:

a) The process named `Multiplex` starting on Line 35 splits the input `x_in` into its even (`x_in`) and odd samples (`x_wait`). Then it generates two new variables named `sx_up` and `sx_low` which are nothing else but the `x_in` and `x_wait` multiplied by `s` respectively (See lines 43 and 44.)

b) Then, two new variables named `sxa0_up` and `sxa0_low` are created which are equal to `sx_up * a(0)` and `sx_low * a(0)` respectively (See lines 56 – 59.)

c) Next, two new variables are created. And they are equal to

```
up0(n) = sxa0_low(n-1) + sx_up(n)
low0(n) = sx_low(n-1) - sxa0_up(n)
```

Note that here *n* and *n-1* represent the current and the previous samples respectively. This is the first stage of the lattice filter (See lines 60 – 68.)

d) In the second stage (See lines 70 – 76) two new variables are created as follows

```
up1 = up0 + low0(n-2) * a(1)
low1 = low0(n-2) - up0 * a(1)
```

e) Finally, the process named `OutputScale` generates the output variables `g` and `h` by dividing the `up1` and `low1` to 256 respectively. Block diagram of this procedure is given in Fig. 4.3.

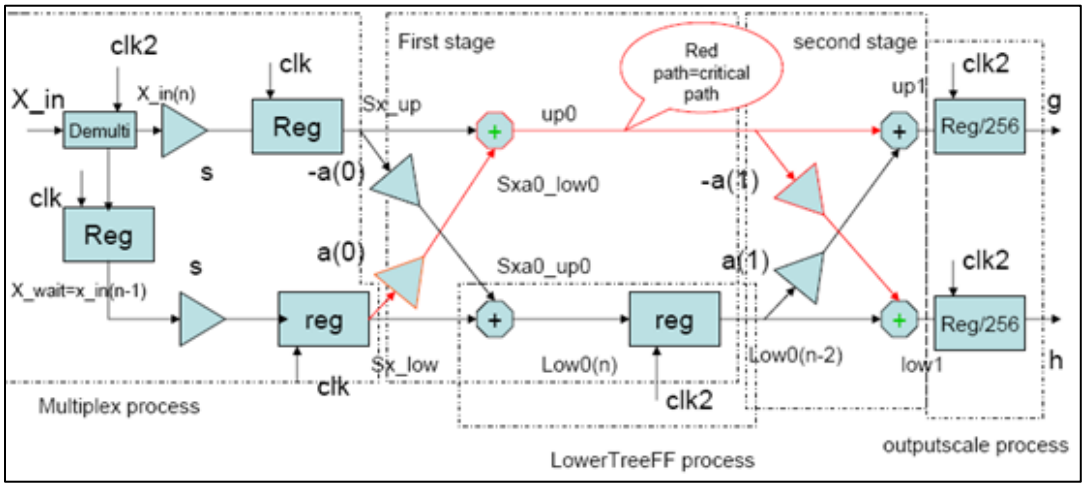


Figure 4.3

2. What do we expect to observe at the outputs of the filters if the input is given as the following?

$$x = \left\{ \underset{\uparrow}{100}, 0, 0, 0, 0, 100, 0, 0, 0, 0 \right\} \tag{4.6}$$

Difference equations of the two filters can easily be derived as

$$\begin{aligned} g(n) &= 0.48x(n) + 0.83x(n-1) + 0.22x(n-2) - 0.13x(n-3) \\ h(n) &= 0.13x(n) + 0.22x(n-1) - 0.83x(n-2) + 0.48x(n-3) \end{aligned} \tag{4.7}$$

Let's run the following code in MATLAB to obtain the response of the filters to the input given in Eq. (4.6)

```
x = [100 0 0 0 0 100 0 0 0 0]; % Load the input
```

```
G = [0.48, 0.83, 0.22, -0.13]; % G filter coefficients
H = [0.13, 0.22, -0.83, 0.48]; % H filter coefficients
g0 = filter(G,1,x); % filtering the input with G
h0 = filter(H,1,x); % filtering the input with H
g = g0(1:2:end) % down-sampling
h = h0(1:2:end) % down-sampling
```

The result is

```
g =
    48    22     0    83   -13
h =
    13   -83     0    22    48
```

We expect to see the same result when we simulate the "db4latti.vhd" file.

3. Open the file "db4latti.qpf" in Quartus II. And change the FPGA device to the one used on your DE2-70.

4. Don't compile the project. Instead select "Start Analysis & Synthesis" under "Start" menu under "Processing" menu. See Fig. 4.4.

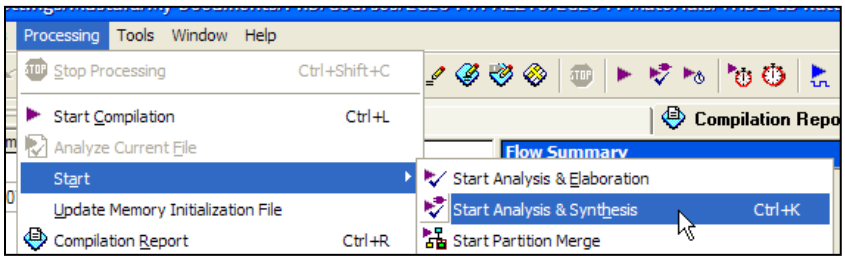


Figure 4.4

5. Open the waveform file named "db4latti.vwf." Change the values of all output signals of "db4latti.vwf" to "Uninitialized" state by right clicking your mouse on each variable and selecting "Uninitialized" under "Value" (See Fig 4.5.) Save the waveform file after you are finished.

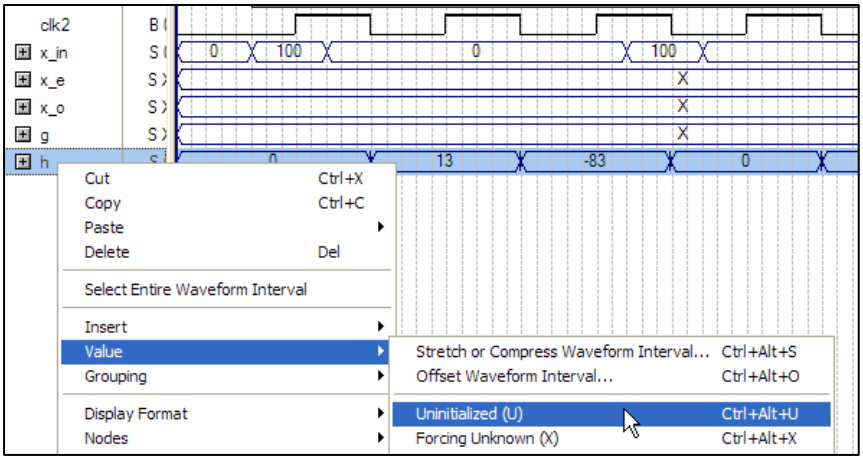


Figure 4.5

6. Open the "Simulator Tool", and make sure you are going to use the file "db4latti.vwf." Start the simulation. You should get a simulation result similar to the one given in Fig. 4.6. You may also want to get a snapshot of your screen to include it in your report later.

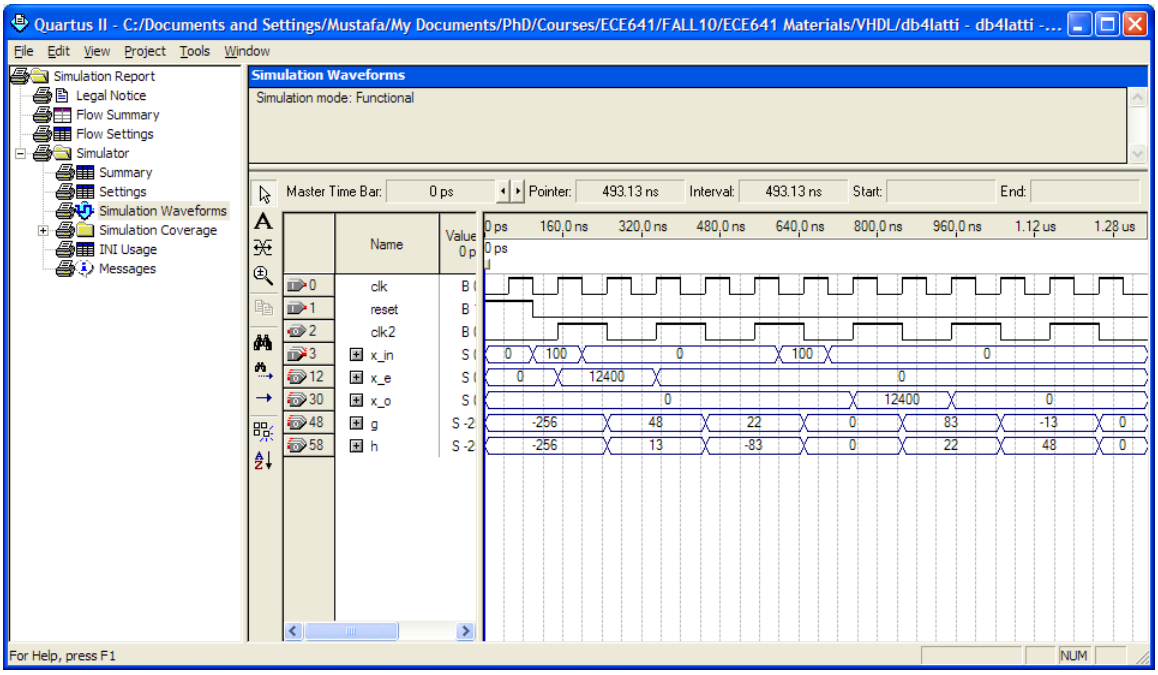


Figure 4.6

By examining the report check if the simulation results match with the theoretical calculations we have done in the second step.

IMPORTANT: As you can see, we do not need to compile the project to simulate it. "Analysis & Synthesis" is the only thing we need to simulate our VHDL codes. Therefore, you do not need to re-compile the VHDL code every time you make a change. This will save you a great amount of time while you are doing your term project.

7. Check how much resource we have used and also check the Fmax of this design. By now you should be able to do these tasks easily. If not, please refer to the previous experiments. Write down the exact values to include it in your report.

8. Now we will determine the "critical path" that has the major effect on the speed of the design. Go back to the compilation Report (Select "Compilation Report" from "Processing" menu.) Click on "clock setup: 'clk' " in the "Timing Analyzer" folder of the report. Right click on the first row of the "clock setup: 'clk'" report and select "Locate in RTL Viewer" under "Locate" menu (See Fig. 4.7.)

9. You will see the "Critical Path" on the schematic in red color and now we know why it takes too much time to propagate to the output `h` from `sx_low`. The reason is the extensive number of adders along the path (See Fig. 4.8.)

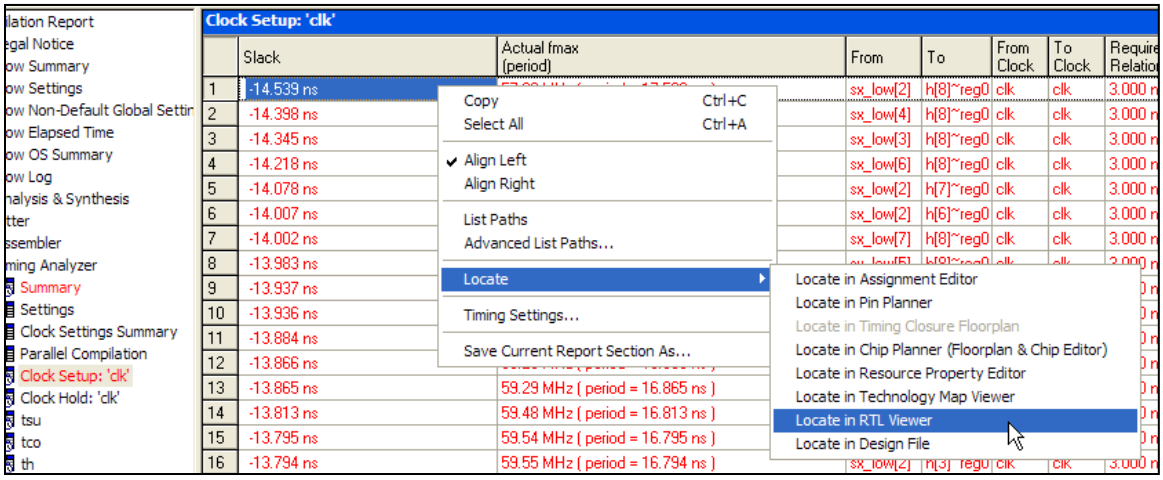


Figure 4.7

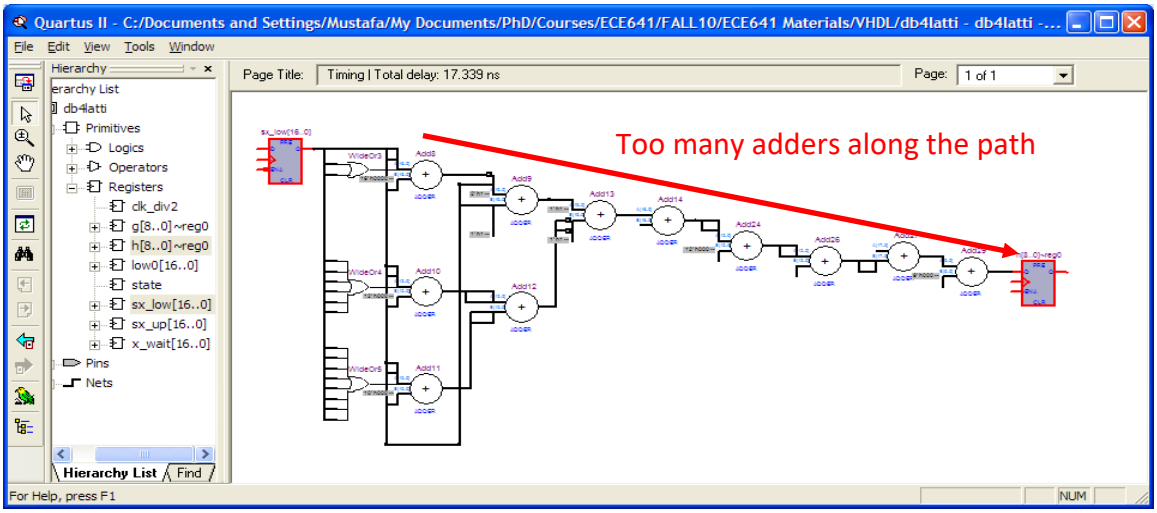


Figure 4.8

D. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures **excluding the codes** in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. Compare two designs in terms of speed and resource used. Provide your observations and make comments.

2. Let's define total number of logic elements (LE) used by polyphase design and lattice design approaches as N_P and N_L, respectively. Let's say you are asked to compare two design approaches in terms of "total number of LEs." Is simply checking which one is greater than the other, i.e. $N_L > N_P$ or $N_L > N_P$, enough to answer this question? Or do we need to something more?

   Hint: Carefully examine Fig. 4.1. Which blocks in that figure are implemented in each design approach? Also, consider the problem shown in Figure 4.8. Which of the two design approaches has a longer critical path?

3. Do a functional analysis on the "db4poly.qpf" project by using the default waveform file named "db4poly.vwf." Include the snapshot of the simulation report.

4. (Optional) Analyze db4poly.vhd file and try to write a summary of the code as we did in Section C.1. (If you choose to answer to this question your report can be three pages long.)

Make sure that you include the question numbers to your answers.

FPGA EXPERIMENT 5
Interacting with the Peripherals on the DE2-70 Board

TABLE OF CONTENTS

A. Introduction
B. Connecting Switches to LEDs
C. Binary Calculator
D. Clocks and Frequency Division
E. More Resources
F. Questions and Report
G. Optional Homework (Running LEDs)

A. INTRODUCTION

The Altera DE2-70 board we use in this laboratory comes with many peripherals including (but not limited to), an A/D converter, video processor, LAN processor, LCD screen, 7-segment displays, switches etc. See Fig 5.1 for an overlay of the peripherals on the board.

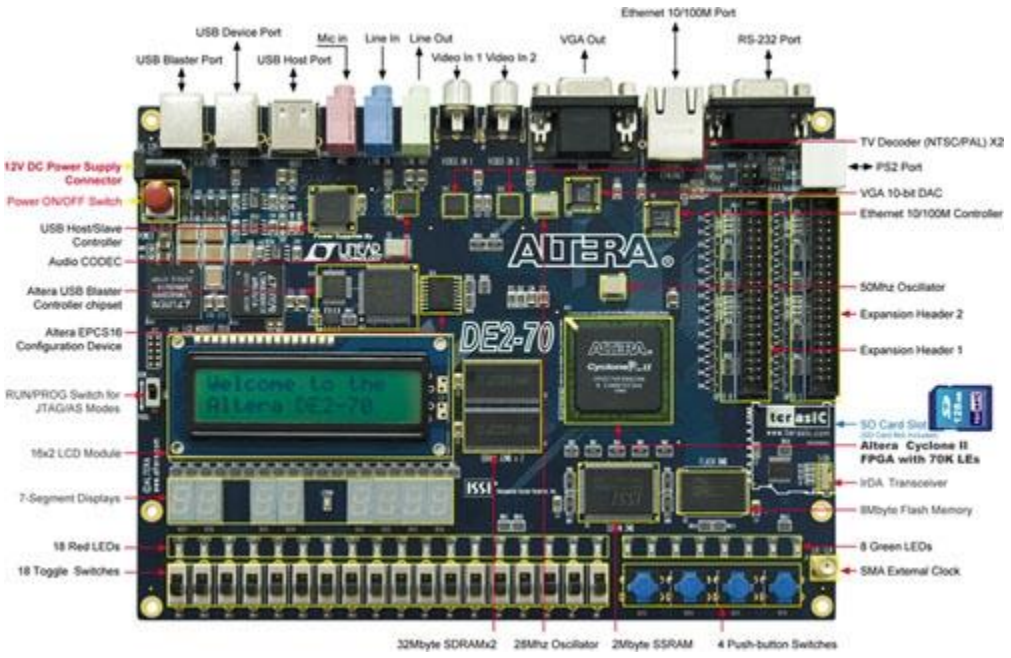


Figure 5.1

In this experiment we will start interacting with the DE2-70 board. We will connect the switches to the LEDs available on the board. If you are interested in (probably you are) using the other peripherals on the board, you will find references to more resources in Section D of this experiment. As a side track you will learn how to create a new Quartus

II project. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

B. CONNECTING SWITCHES TO LEDS

1. Let's start with creating the VHDL file. Select "New" from "File" menu and select "VHDL File" in the prompt dialog box. Copy and paste the following VHDL code into the editor. Save the VHDL file and name it as "calculator.vhd" (See Fig. 5.2.)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY calculator IS
      PORT ( switches: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                LEDs: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END calculator;

ARCHITECTURE calculator_code OF calculator IS

BEGIN
      LEDs <= switches;
END calculator_code;
```

Important: Always be careful when copying pasting code from this manual into the editor in Quartus II. Sometimes the underscore character, "_" is not copied from PDF files into the editor.

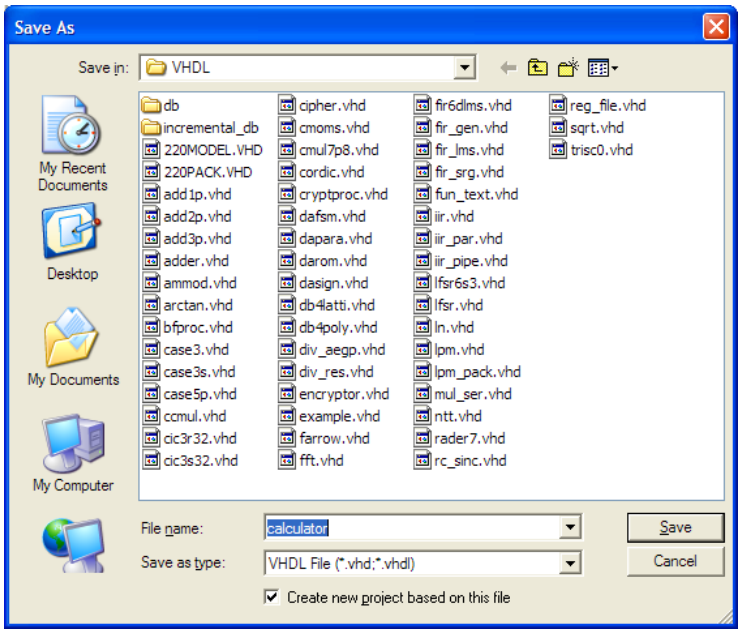


Figure 5.2

2. Let's create a new project. Open Quartus II software and select "New Project Wizard…" under "File" menu (See Fig. 5.3.) In the prompt window, enter the name of your new project which should be "calculator" (without quotation marks.)

3. Select "No" if you see the message box given in Fig. 5.4.

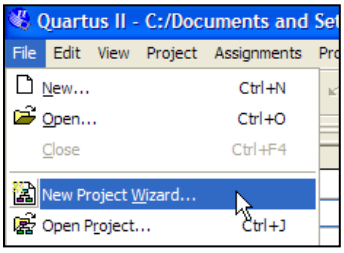


Figure 5.3

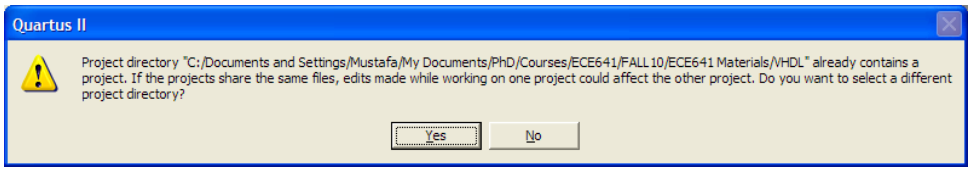


Figure 5.4

4. Write "calculator.vhd" as "File name" (or select it from the Folder by clicking on "…" button) and click on "Add" button (See Fig. 5.5.) Click on "Next" button.

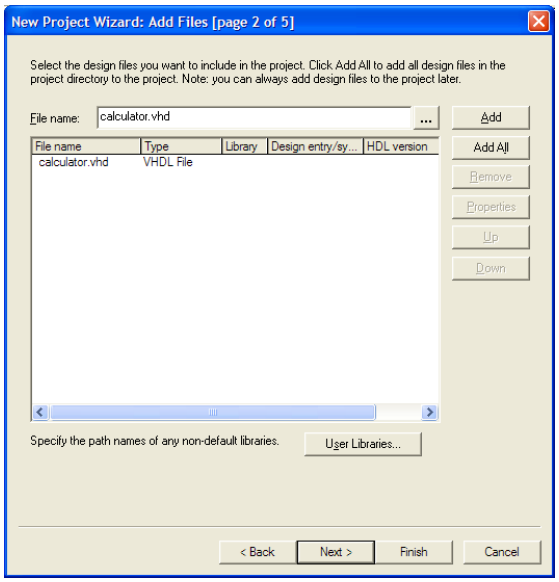


Figure 5.5

5. Select the correct FPGA chip from the list. Then, click on "Finish" button. Congratulations! You have created your first project in Quartus II!

6. Perform an "Analysis and Synthesis." Make sure it is successfully completed.

7. Select “Pin Planner” under “Assignments” menu (See Fig. 5.6.) On the bottom of the screen you will find the pane we use to assign “Pins” to “ports.” Pins are the names of the hardware input and output of the FPGA chip. Think them as the postal address of each input/output. Each peripheral on your DE2-70 board is connected to a pin. On the other hand, ports are the names of software input and output. They are defined in the VHDL file and can be changed anytime. Now click on cell in the first row in “Location” column. Write “PIN_AJ5” to this field (without quotation marks.) This is the name of the pin that is connected to the red colored LED labeled as “LEDR2” on your board. Assign pins to the other five ports as given in Fig. 5.7.

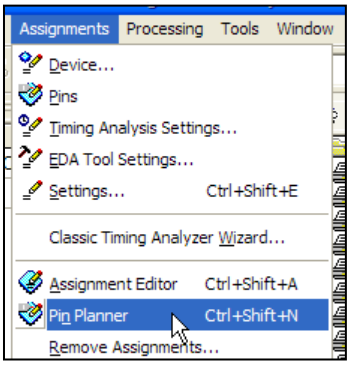


Figure 5.6

Note: If you need the “map” of your pins for DE2-70 board, go to “ECE641 Materials” folder in Windows Explorer and locate the file named “DE2_70_pin_assignments.csv.” You will need either a text editor or spreadsheet software such as Microsoft Excel in order to open that file. The file is self-explanatory.

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	LEDs[2]	Output	PIN_AJ5	8	B8_N3	3.3-V LVTTL (default)	
2	LEDs[1]	Output	PIN_AK5	8	B8_N3	3.3-V LVTTL (default)	
3	LEDs[0]	Output	PIN_AJ6	8	B8_N2	3.3-V LVTTL (default)	
4	switches[2]	Input	PIN_AB25	6	B6_N2	3.3-V LVTTL (default)	
5	switches[1]	Input	PIN_AB26	6	B6_N2	3.3-V LVTTL (default)	
6	switches[0]	Input	PIN_AA23	6	B6_N2	3.3-V LVTTL (default)	

Figure 5.7

8. Compile the project. Connect your DE2-70 board to your computer and power up the board. Download the image into the FPGA. If you need help, refer to Section F of the second experiment.

9. Enjoy your first interaction with the DE2-70 board! Try changing the state of the first three switches on the right. The red LEDs on top of each switch should give reaction (See Fig. 5.8.) Why do you think only the first three LEDs are reacting? Analyze the VHDL file in order to understand the reason.

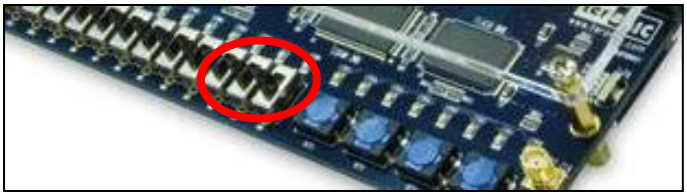


Figure 5.8

C. BINARY CALCULATOR

Your FGPA chip has tons of logical gates, but the previous example does not require any logical gate since it is just hard-wiring different components together. Let's make things more complicated and build a simple binary calculator:

1. Now, we will modify the VHDL file to perform binary AND operation via the switches. Open the VHDL file named "calculator.vhd" and modify the line 12 as follows:

Original:

```
      LEDs <= switches;
```

Modified:

```
      LEDs(2) <= switches(0) AND switches(1);
```

What we do here is to read values of the first and second bits of the 3-bit long switches port, perform a logical AND operation and assign the outcome to the third bit of the LEDs port. Save the file when you are finished.

2. Compile the project and download the image to the FPGA.

3. Change the states of the first and second switch on the right. At which state is the third red LED on? Is it something you expected?

D. CLOCKS AND FREQUENCY DIVISION

Now, we will learn how to utilize a clock in our circuits. We will also implement a frequency divider circuit to slow down the clock. As a side track, we will emphasize on the functionality of for-loops.

One of the many ways of utilizing a clock in VHDL is as follows

```
ENTITY clock_circuit IS
  PORT (clk    : IN STD_LOGIC);
END clock_circuit;
```

```
ARCHITECTURE clock_circuit_arch OF clock_circuit IS

BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL clk = '1';
      -- Whenever a rising-edge is observed in clk input,
      -- every operation is done at once, in parallel in this block.
  END PROCESS;
END clock_circuit_arch;
```

Please note that `clk` is defined as a binary input to the circuit on the second line. The VHDL code has only one process which operates whenever a rising edge is observed in the `clk` input. This functionality is provided by the `WAIT` keyword. Now, let's create a flip-flop implementation:

1. Create a Quartus project named as "freq_divider" and include "freq_divider.vhd" and "freq_divider.vwf" files into the project. Top level (and the only) design entity must be set to "freq_divider."

2. Perform analysis and synthesis. Perform functional simulation and observe the simulation report. Put a snapshot of the report into your report.

3. Go to pin planner and assign 50 MHz clock (iclk_50) to `clk`, and 8 green LEDS to `LED` port. Compile the project. Download it to the DE2-70 board. What do you see? Refer to "DE2_70_pin_assignments.csv" file on the root of course materials folder.

4. Edit "freq_divider.vhd" file so that `period_in_cycles` constant (originally equal to 5) is equal to 10,000,000 (don't forget to drop the commas!) Compile the project, download it to the board, and observe the result on the board.

5. Repeat step 4 with 50,000,000.

E. MORE RESOURCES

In this experiment, we started to interact with the board, but this only a tiny step. You are encouraged to go through the experiments provided with your board. They are available in your "ECE641 Materials" folder under "DE2_additional_lab_exercises."

As mentioned in the introduction section of this experiment, the DE2-70 board has a lot more peripherals than just a couple of switches and LEDs. For details about other peripherals and datasheets you are referred to the following files and folders located in your "ECE641 Materials" folder:

a) DE2_70_User_manual_v108.pdf
b) \DE2_70_schematics\DE2_70_schematic_v11.pdf
c) \Datasheets

Let us remind you that you must pick a DSP topic for your term-project, and you should utilize one or some of the peripherals on the DE2-70 board. Though the experiments in this manual do not specifically require the board, it is your responsibility to modify the provided codes in each experiment so that the final product lets the user interact with the board.

F. QUESTIONS AND REPORT

Your group report must be maximum **three pages long including the figures and codes** in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. Consider the first example given in this experiment. What modification to the VHDL code should be done in order to also hard-wire the fourth switch and the red LED, namely SW3 and LEDR3? Is modifying the VHDL code enough? If not, what else should be done?

2. Make your binary calculator complete by adding OR, XOR, and NOT operators. The final product should be able to do all four operations including the AND operator we covered in the experiment. You may utilize more switches and more LEDs.

3. What is the relation between `clk` and `slower_clk` ports? What did change between steps 3, 4, and 5? Why no blinking is observed (by humans) in step 3?

4. How is the “frequency division” implemented? What role does `freq_ctr` signal play in this operation?

5. Edit “freq_divider.vhd” file so that the following for-loop in the original code

```
for i in 0 to 7 loop
      LED(i) <= slower_clk;
end loop;
```

is replaced with the following

```
LED(0) <= slower_clk;
LED(1) <= slower_clk;
```

```
	LED(2) <= slower_clk;
	LED(3) <= slower_clk;
	LED(4) <= slower_clk;
	LED(5) <= slower_clk;
	LED(6) <= slower_clk;
	LED(7) <= slower_clk;
```

Compile the new code, download it, and observe the result. Did anything change in terms of circuit operation? What role does for-loop play in VHDL?

G. OPTIONAL HOMEWORK (RUNNING LEDS)

Modify the VHDL code given in this experiment so that the green LEDs on the board act as follows: First, the LEDG0 is on, then the LEDG0 goes off and LEDG1 is on, and so on. Finally LEDG7 goes off and LEDG0 lids again. When the clock is relatively slow, humans perceive this in a way that the LED is "running." This type of circuit is also called "running LED circuit." Submit the soft-copy of your code and demonstrate your circuit to the instructor.

H. FAQ FOR EXPERIMENT 5

1. How can I know the pin locations of the peripherals?

Answer: Go to "ECE641 Materials" folder (you downloaded from the course web page) in Windows Explorer and locate the file named "DE2_70_pin_assignments.csv."

2. I downloaded the design to the board but some of the peripherals are not working properly.

Answer: Please check your pin assignments carefully in the pin planner.

FPGA EXPERIMENT 6
Radix-2 Cooley-Tukey Algorithm for 4-Length FFT

TABLE OF CONTENTS

A. Introduction
B. Twiddle Factor Multiplier
C. Butterfly Processor
D. Radix-2 length-4 FFT Implementation
E. VHDL Method
F. Block Diagram Method
G. Questions and Report

A. INTRODUCTION

The Cooley–Tukey algorithm, named after J.W. Cooley and J. Tukey, is the most commonly used fast Fourier transform (FFT) algorithm[1]. In this experiment, we will study two different methods to connect different blocks written in VHDL together to perform length-4 FFT using Radix-2 Cooley-Tukey Algorithm on FPGA. We will also learn how to create a waveform file from scratch. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

The file "ccmul.vhd" in "VHDL/lab_6/lab_6_1" folder is a "Twiddle Factor Multiplier." You are required to read and understand the algorithm given on page 367 (Algorithm 6.10: Efficient Complex Multiplier) and the code in the folder. Though the code on the book on page 368 (Example 6.11: Twiddle Factor Multiplier) is different from our code, you are strongly encouraged to study this example. The file "bfproc.vhd" in "VHDL/lab_6/lab_6_1" is a "Butterfly Processor." The length-4 FFT system utilizes VHDL files named "ccmul.vhd", "bfproc1.vhd", and "bfproc2.vhd" in "VHDL/lab_6/ lab_6_2" folder.

B. TWIDDLE FACTOR MULTIPLIER

We need complex multiplication processors for the FFT algorithm. Twiddle factor multiplier is a preferred method in complex multiplication since it uses less number of multipliers. Refer to Algorithm 6.10 in the textbook on page 367 for details.

1. Open the project named "ccmul.qpf" located in "VHDL/lab_6/lab_6_1" folder.

[1] http://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm

2. Before you go any further in the lab read this step entirely. When *X = 64, Y = 0, C = Cos(-π/4)*, and *S = Sin(-π/4)* what is *R* and *I*? What is the theoretical result?

Following the similar steps for "functional simulation" we have done in the previous experiments, simulate the multiplication and save your result (Screenshot is a good idea.) Check if the simulation result matches with the theoretical result.

3. Repeat step 2 for *X = 64, Y = 0, C = Cos(-3π/4)*, and *S = Sin(-3π/4).* Modify ccmul.vwf waveform file. Note that *-3π/4* is in radian.

C. BUTTERFLY PROCESSOR

We also need butterfly processors for the FFT algorithm. The butterfly processor computes two equations. For complex inputs of the butterfly processor as given

$$\begin{aligned} A &= A_{re} + jA_{im} \\ B &= B_{re} + jB_{im} \end{aligned} \tag{6.1}$$

complex outputs are formulated as

$$\begin{aligned} D &= D_{re} + jD_{im} = (A_{re} + jA_{im}) + (B_{re} + jB_{im}) \\ E &= E_{re} + jE_{im} = (A_{re} + jA_{im}) - (B_{re} + jB_{im}) \end{aligned} \tag{6.2}$$

1. Open the project named "bfproc.qpf" located in "VHDL/lab_6/lab_6_1" folder. Read the code carefully and try to understand its operation.

2. When A *= 1 + j2, B = 3 + j4,* what is *D* and *E*? What is the theoretical result?

3. Perform a functional simulation and save your result (Screenshot is a good idea.) Check if the simulation result matches with the theoretical result.

4. Repeat step 3 for A *= 5 + j8, B = 6 + j12.*

D. RADIX-2 LENGTH-8 FFT IMPLEMENTATION

Intuitively, we know that the standard strategy to speed up an algorithm is to *divide and conquer.* We have to find some ways to group the terms in the forward DFT transform equation

$$Y(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}; \quad k = 0,1,...,N-1; \quad W = e^{j\frac{2\pi}{N}}; \quad N = even \tag{6.3}$$

Let's rearrange it by separating the first and second halves of $x(n)$ as follows

$$\begin{aligned} Y(k) &= \sum_{n=0}^{N/2-1} x(n)W_N^{kn} + W_N^{kN/2}\sum_{n=0}^{N/2-1} x(n+N/2)W_N^{kn} \\ &= \sum_{n=0}^{N/2-1} [x(n)+(-1)^k x(n+N/2)]W_N^{kn}; \quad k=0,1,\ldots,N-1 \end{aligned} \tag{6.4}$$

For *k* is even we have

$$Y(2r) = \sum_{n=0}^{N/2-1} [x(n)+x(n+N/2)]W_{N/2}^{rn} \tag{6.5}$$

and for *k* is odd we have

$$Y(2r+1) = \sum_{n=0}^{N/2-1} [x(n)-x(n+N/2)]W_N^n W_{N/2}^{rn} \tag{6.6}$$

where

$$\begin{aligned} x_0(n) &= x(n)+x(n+N/2) \\ x_1(n) &= [x(n)-x(n+N/2)]W_N^n \end{aligned} \tag{6.7}$$

For length-4 FFT, N=4, the signal flow diagram of the algorithm is shown in Fig. 6.1.

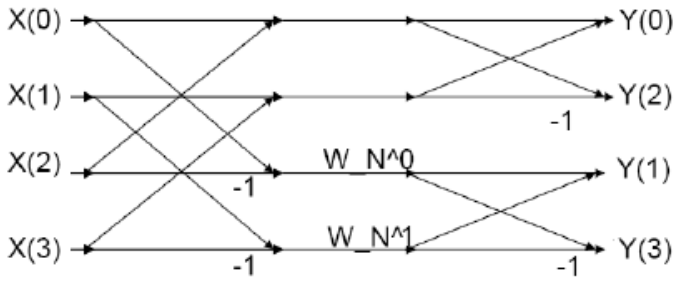


Figure 6.1: Signal Flow Diagram of length-4 FFT

From Fig. 6.1, we can see that the whole system could be divided into six blocks: four butterfly processors (red rectangles) and two twiddle factor multipliers (blue rectangles) as shown in Fig. 6.2. The block diagram of this system is shown in Fig. 6.3.

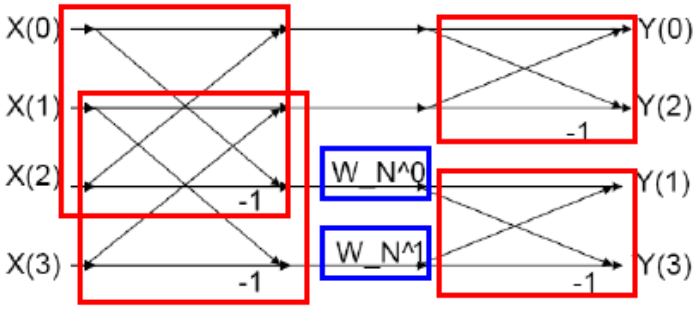


Figure 6.2: Six Blocks of length-4 FFT

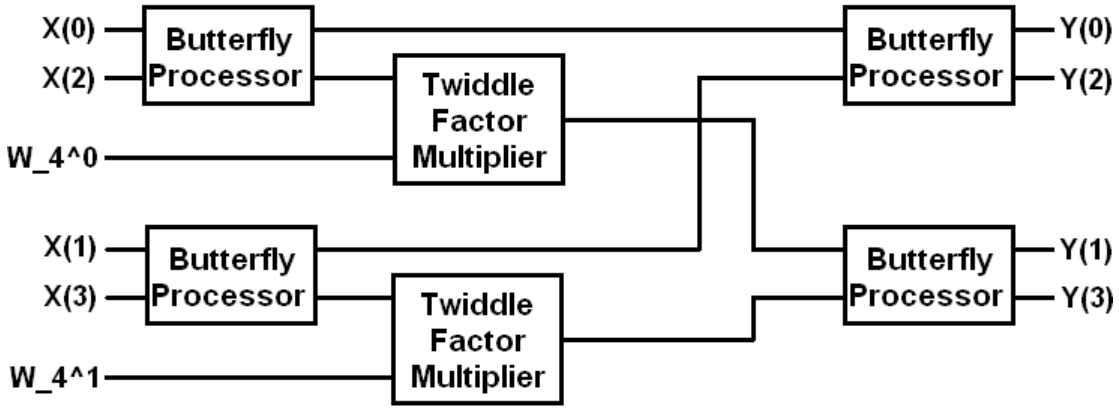


Figure 6.3: Block Diagram of length-4 FFT

In parts B and C of this experiment, we have already studied the VHDL code of twiddle factor multiplier and butterfly processor. Quartus II provides two methods to connect these blocks to each other to build the system: VHDL method and block diagram method. Let's explore the both methods starting with the former:

E. VHDL METHOD

1. Open the project named "fft4_vhdl.qpf" located in "VHDL/lab_6/lab_6_2" folder. Click on the "Files" tab in the "Project Navigator" window (See Fig. 6.4.) Make sure you are able to see all four files linked to the project.

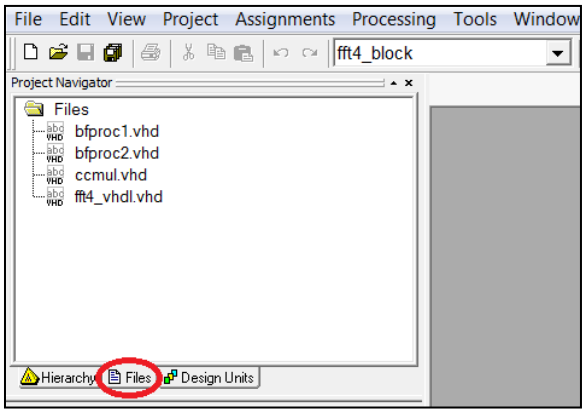


Figure 6.4

2. Double click on file "bfproc1.vhd" and "bfproc2.vhd". Read the code carefully and think about that why we use two different kinds of butterfly processors and what the difference between them is. Include your answers in your report.

3. Double click on file "fft4_vhdl.vhd" and read the code carefully. What you are reading is the first method to connect different VHDL code pieces to each other. The code can

be divided into three parts: First, we define the ports (inputs and outputs) of the system, from line 7 to line 31. Second, we declare the components that are used in the system, from line 35 to line 58. Third, we define the signals from line 62 to line 67 and perform mapping from line 60 to line 84.

4. Right click on file "fft4_vhdl.vhd" in "Project Navigator." Select "Set as Top-Level Entity" from the menu. See Fig. 6.5. Note that the top-level design entity name should be same as the project name; otherwise Quartus II gives a warning. Perform an Analysis and Synthesis.

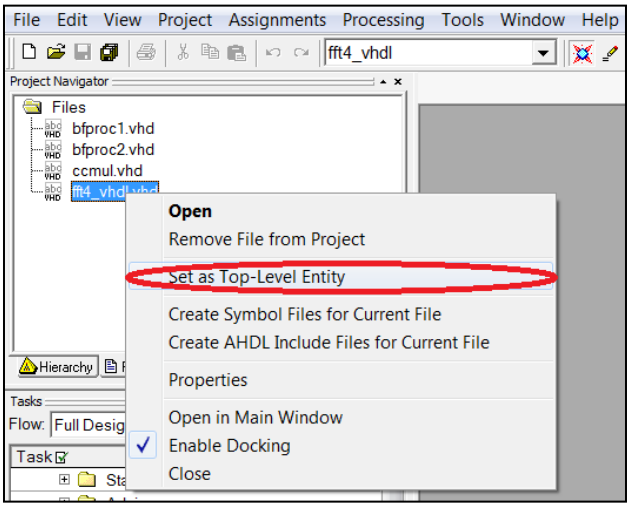
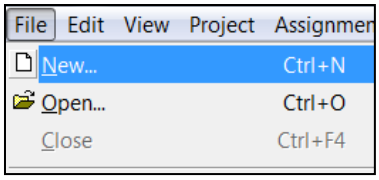


Figure 6.5

5. Now we will create a new waveform file to use it for simulation. Click on "New…" under "File" menu. See Fig. 6.6. Select "Vector Waveform File" in the dialog box. See Fig 6.7.

6. Click on "Save" under "File" menu to save the waveform file. See Fig. 6.8. Name the file as "fft4_vhdl." Make sure that "Add file to current project" is selected. See Fig. 6.9.

7. Add the following nodes to the waveform file: `clk_ref`, `w4_0_c`, `w4_0_cms`, `w4_0_cps`, `w4_1_c`, `w4_1_cms`, `w4_1_cps`, `x0_im`, `x0_re`, `x1_im`, `x1_re`, `x2_im`, `x2_re`, `x3_im`, `x3_re`, `y0_im`, `y0_re`, `y1_im`, `y1_re`, `y2_im`, `y2_re`, `y3_im`, and `y3_re`. Refer to step 2 of section C in the second experiment if you need help. Make sure that the "Radix" is "Signed Decimal" for all nodes **except** `clk_ref`. `clk_ref` is defined as "Binary". You can reorder these nodes by dragging and dropping them. Your waveform file should look like the one given in Fig. 6.10.



See Figure 6.6

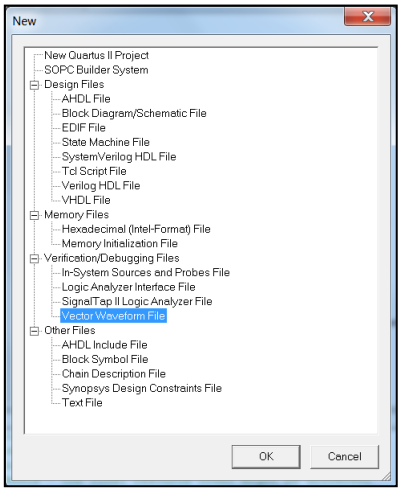


Figure 6.7

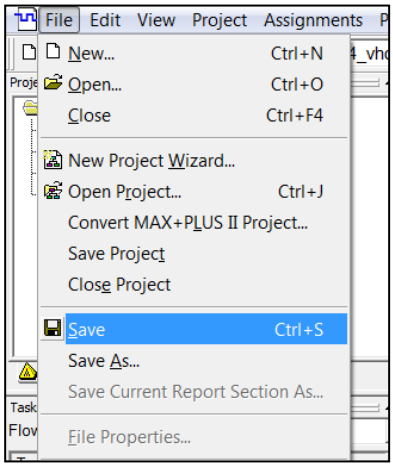


Figure 6.8

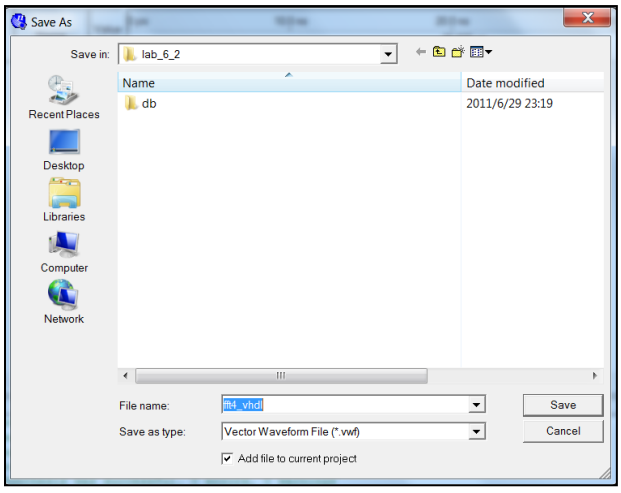


Figure 6.9

8. Select `clk_ref` and click on "Overwrite Clock" icon on the toolbar (See Fig. 6.10.) Set the "Period" of the clock to 50 ns.

9. Select x0_re and click on "Arbitrary Value" icon on the toolbar similar to previous step. Set the "Numeric or named value" to 127. Repeat this procedure for other nodes and the values given in Table 6.1. Your waveform file should look like the one given in Figure 6.11. Save the waveform file when you are finished.

Node	x0_re	x0_im	x1_re	x1_im	x2_re	x2_im	x3_re	x3_im

Value	127	-34	-80	12	-159	-63	-30	95
Node	**w4_0_c**	**w4_0_cps**	**w4_0_cms**	**w4_1_c**	**w4_1_cps**	**w4_1_cms**		
Value	500	500	500	0	-500	500		

Table 6.1

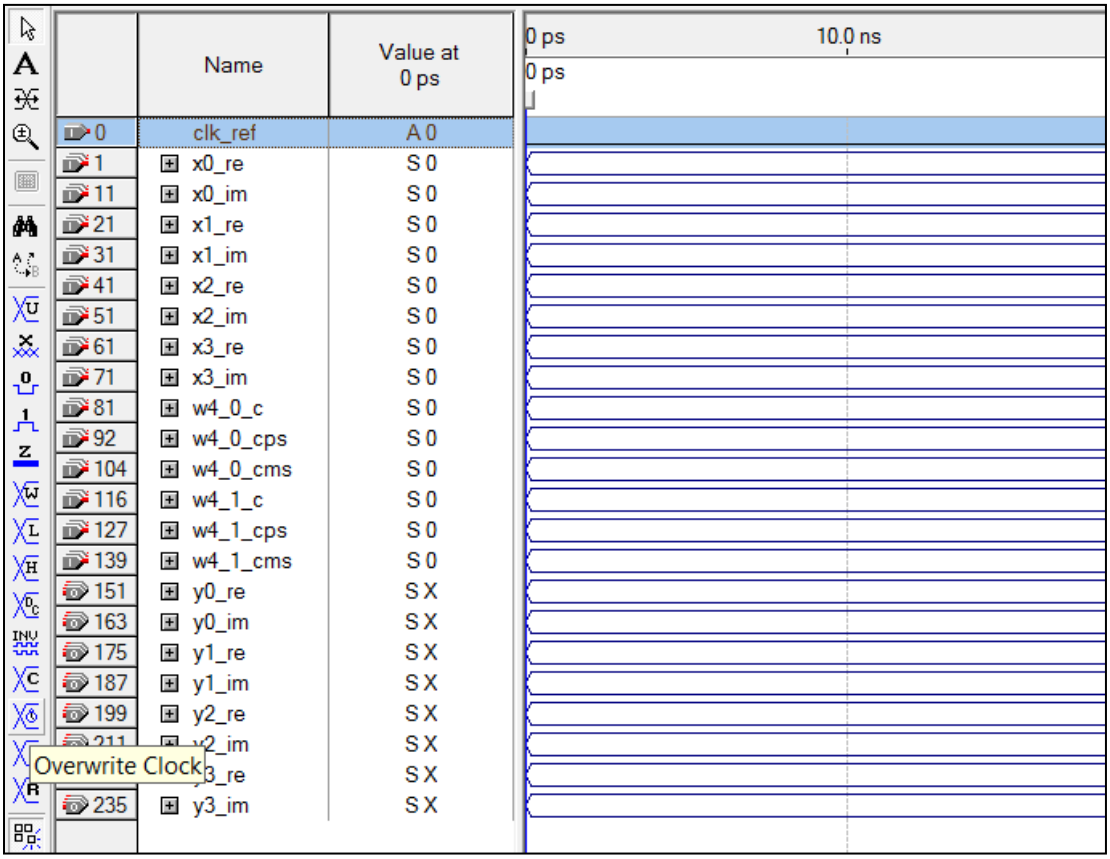


Figure 6.10

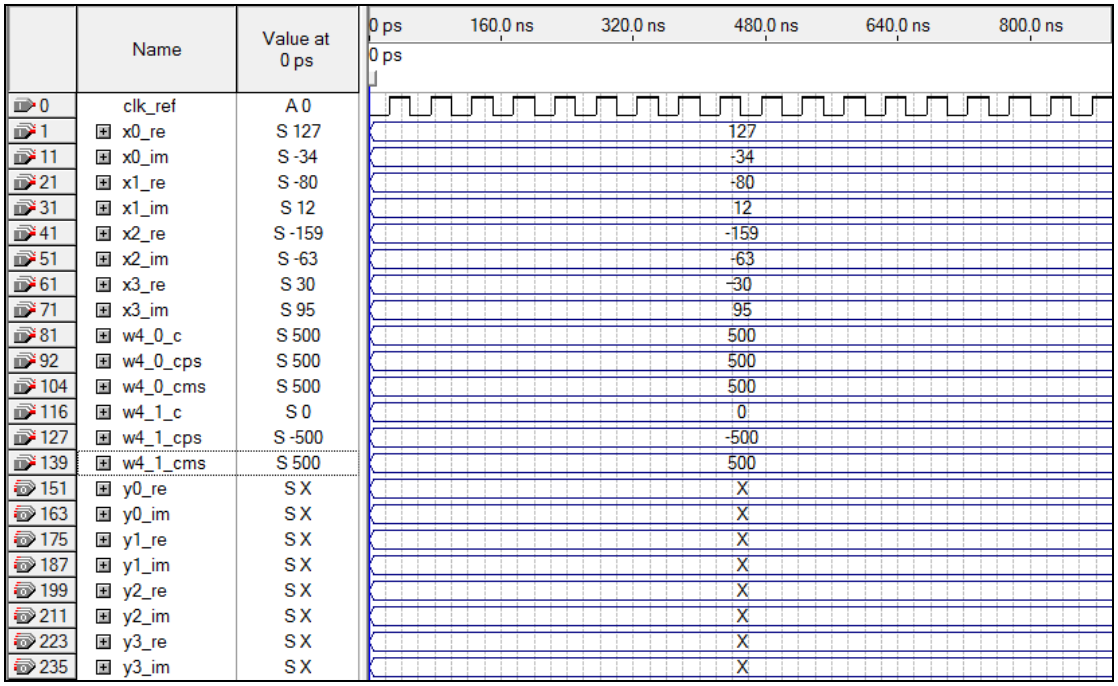


Figure 6.11

10. Let's run the following MATLAB code to find out the expected result

```
x=[127-34i, -80+12i, -159-63i, -30+95i]
y=fft(x)
```

The result is

```
y = -142 + 10i    203 + 79i    78 - 204i    369 - 21i
```

11. Run a functional simulation using the waveform file you have just created. Save the results and include it in your report. Does the result meet with the expected result?

12. Change the input and repeat the steps 10 and 11. Include the simulation result in your report.

13. Compile the project, and record the total number of logic elements, fmax, and worst case Tco. Include these results in your report. Refer to previous experiments if you have trouble in obtaining these results.

F. BLOCK DIAGRAM METHOD

1. Open the project named "fft4_block.qpf" located in "VHDL/lab_6/lab_6_2" folder.

2. Click on "Files" in "Project Navigator." See Fig. 6.12. Make sure you see all the three files in this project. Right click on file "bfproc1.vhd" in "Project Navigator." Select "Set as

Top-Level Entity." See Fig. 6.13.

3. Perform Analysis and Synthesis. For the time being, ignore the warning "Ignored assignments for entity fft4_block – entity does not exist in design", the entity "fft4_block" will be created later.

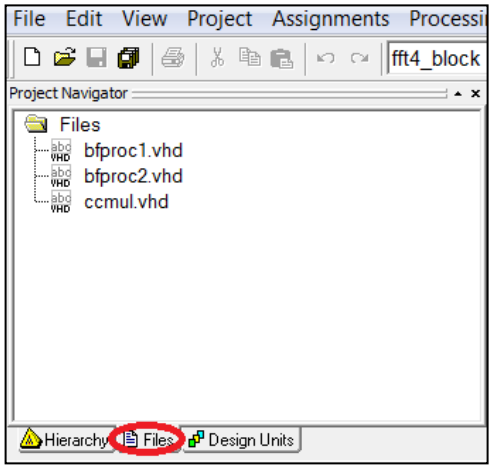


Figure 6.12

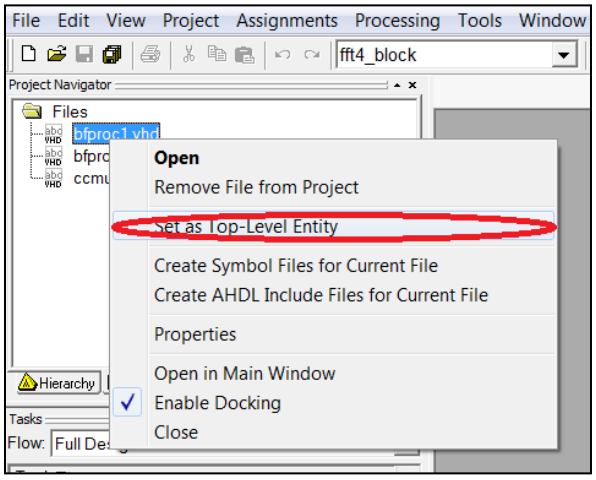


Figure 6.13

4. Right click on file "bfproc1.vhd" in "Project Navigator." Select "Create Symbol Files for Current File." See Fig. 6.14.

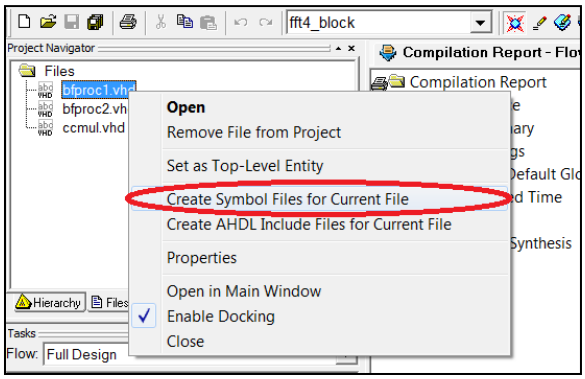


Figure 6.14

5. Repeat previous step for files "bfproc2.vhd" and "ccmul.vhd" to create their individual symbol files.

6. Now we will create a new block diagram file and use it for system construction. Click on "New…" under "File" menu (See Fig. 6.6). Select "Block Diagram / Schematic File" in the dialog box (See Fig. 6.15.) A file named "Block1.bdf" will be generated (See Fig. 6.16). This file looks like a clear circuit board. In the following steps we will put some IC blocks on it.

7. Double click on any blank area of file "Block1.bdf". The "Symbol" window will be opened. Open the "Project" menu in "Libraries" and we will see three symbols which we were generated in steps 4 and 5. Note that their name is same as their corresponding file names. See Fig. 6.17.

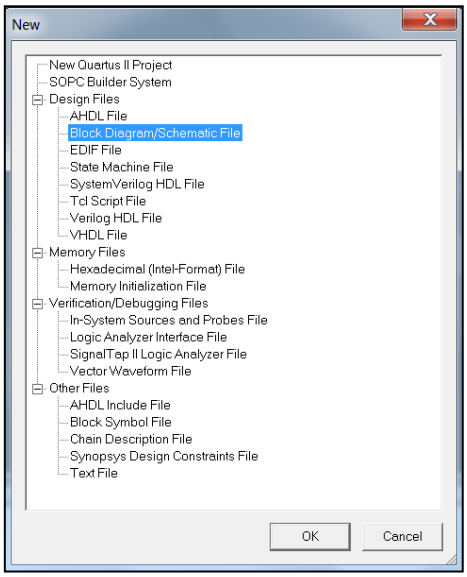


Figure 6.15

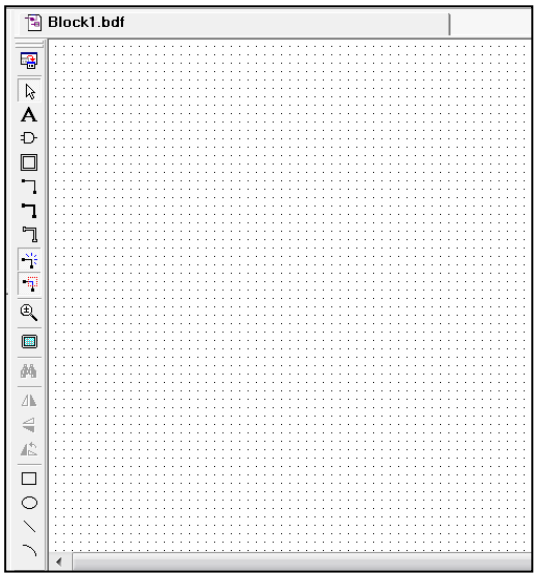


Figure 6.16

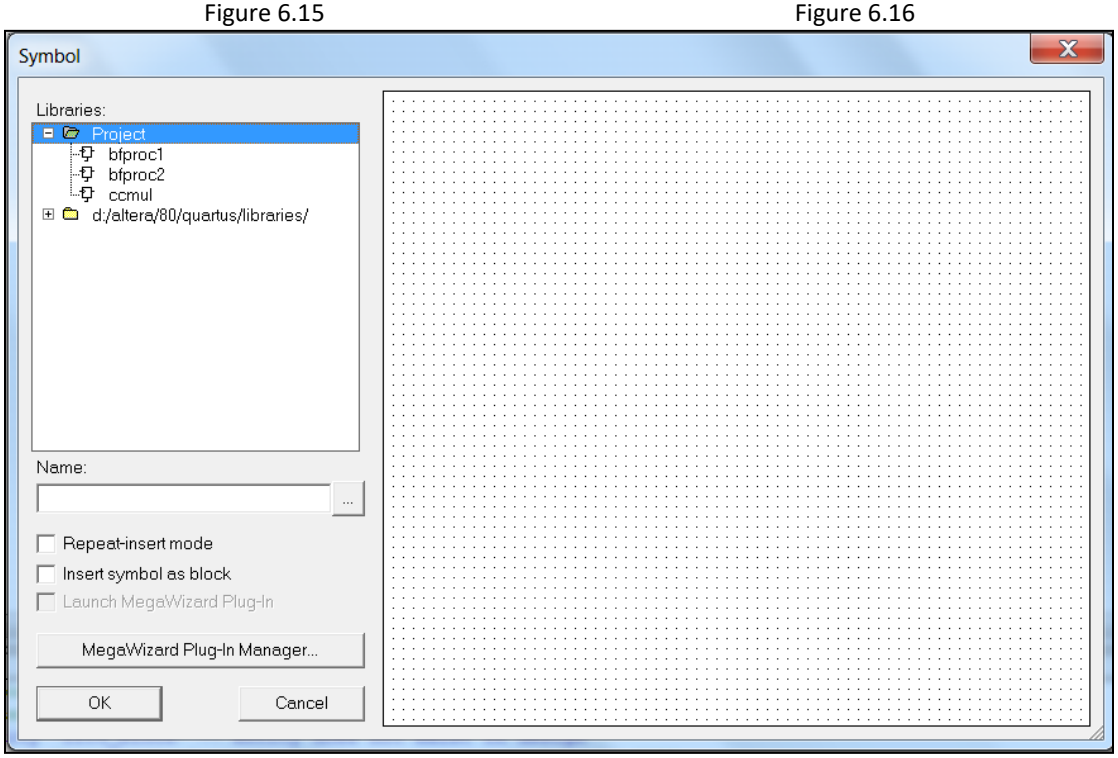


Figure 6.17

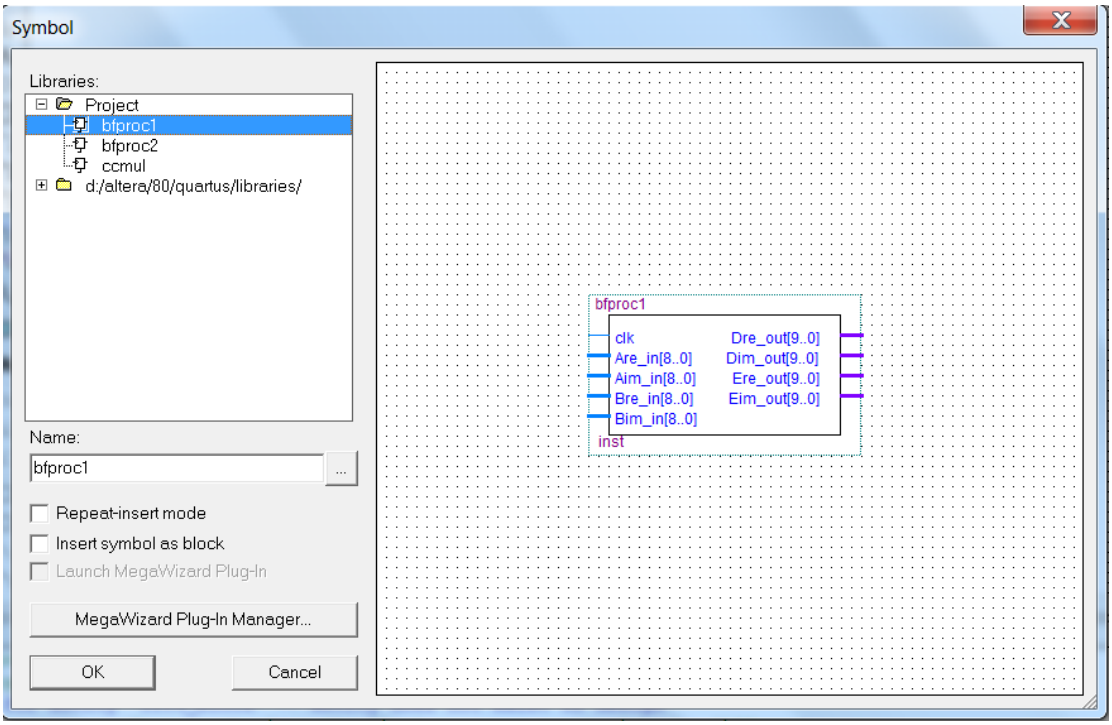


Figure 6.18

8. Select symbol "bfproc1" (See Fig. 6.18) and press on "OK" button. Put it on the blank area of file "Block1.bdf". You can change its position by drag-and-drop method.

9. To build a length-4 FFT system, we need two "bfproc1" symbols, two "ccmul" symbols and two "bfproc2" symbols. Repeat previous step to place all the symbols we need. It is suggested that you select "Detach Window" icon on the toolbar to give you a wider operation area. See Fig. 6.19.

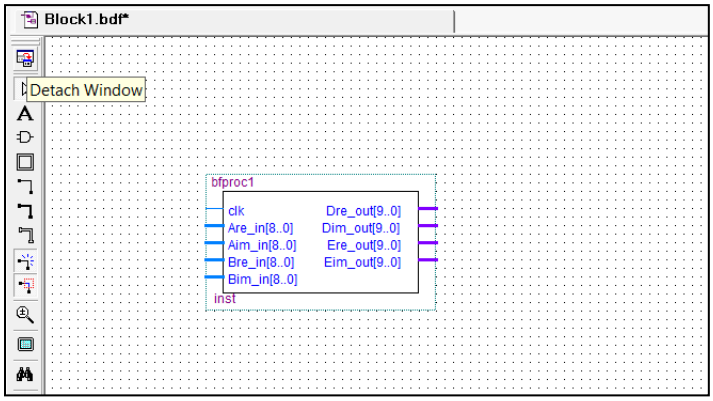


Figure 6.19

10. Next, we will place inputs of the entire system. Double click on any blank area of file "Block1.bdf". The "Symbol" window will be opened. Type "input" in "Name" field. An "input" symbol should appear in the list. See Fig. 6.20. Place an "input" symbol on file "Block1.bdf". Double click this "input" symbol and rename the "Pin name(s)" to be

"clk_ref." See Fig. 6.21.

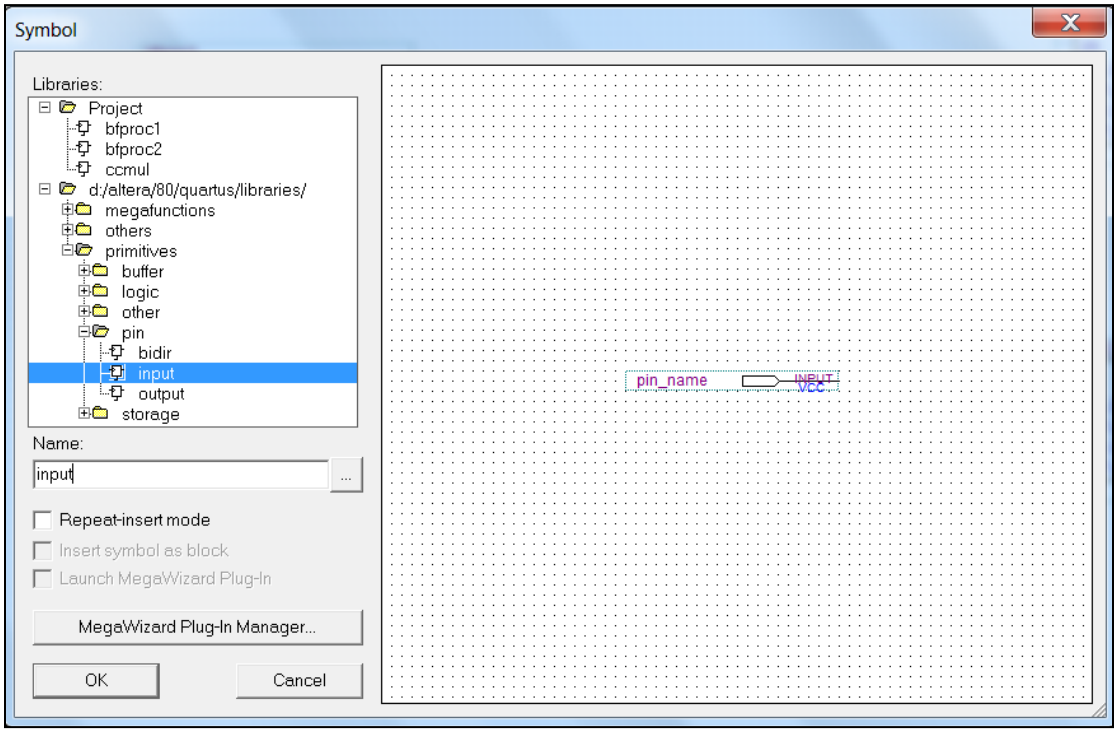


Figure 6.20

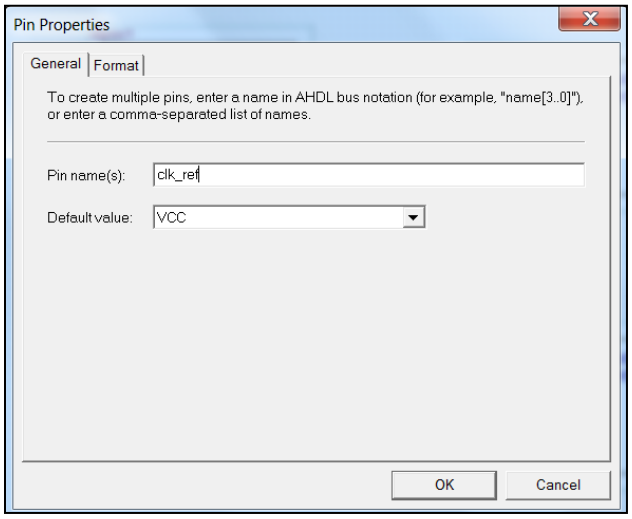


Figure 6.21

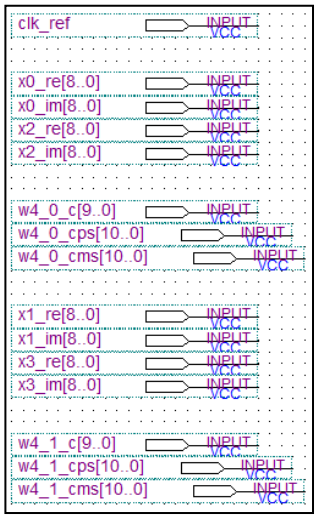


Figure 6.22

11. Repeat the previous step 14 times. We need 15 inputs in the entire system. The "Pin name(s)" of other inputs should be x0_re[8..0], x0_im[8..0], x2_re[8..0], x2_im[8..0], w4_0_c[9..0], w4_0_cps[10..0], w4_0_cms[10..0], x1_re[8..0], x1_im[8..0], x3_re[8..0], x3_im[8..0], w4_1_c[9..0], w4_1_cps[10..0], and w4_1_cms[10..0]. All the inputs we need are shown in Fig. 6.22.

12. Next, we will place outputs of the whole system. Double click on any blank area of file "Block1.bdf." The "Symbol" window will be opened. Type "output" in "Name" field. An "output" symbol should appear in the list (See Fig. 6.23). Place an "output" symbol on file "Block1.bdf". Double click this "output" symbol and rename the "Pin name(s)" to "y0_re[10..0]" (See Fig. 6.24).

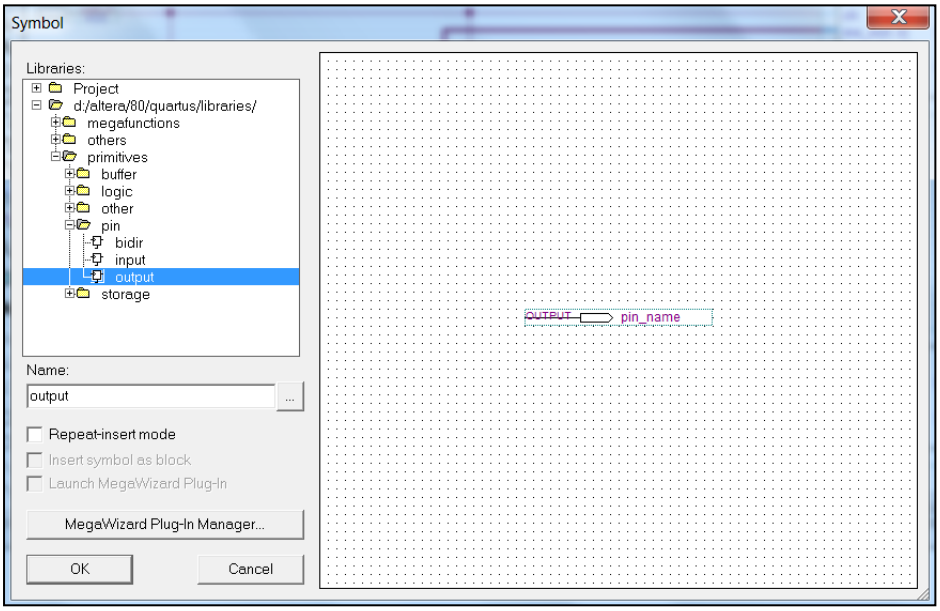


Figure 6.23

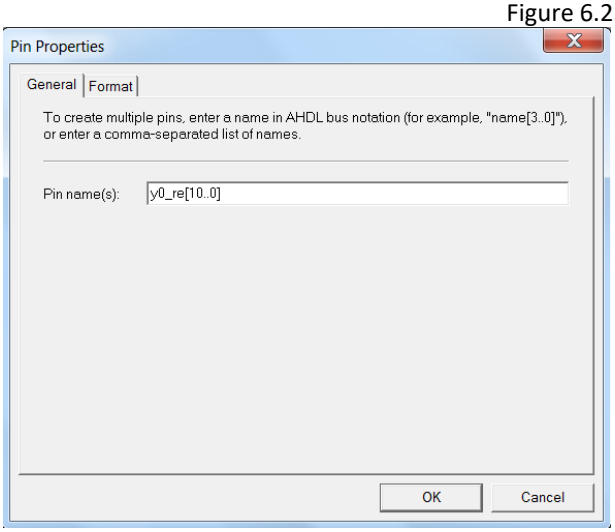


Figure 6.24

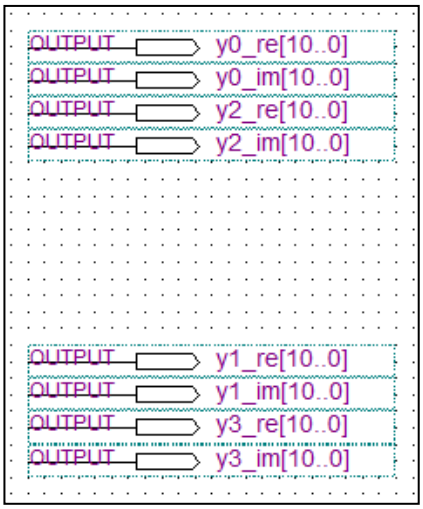


Figure 6.25

13. Repeat the previous step 7 times. We need 8 outputs in the entire system. The "Pin name(s)" of other outputs should be y0_im[10..0], y2_re[10..0], y2_im[10..0], y1_re[10..0], y1_im[10..0], y3_re[10..0], and y3_im[10..0]. All the outputs we need are shown in Fig. 6.25.

14. Now we will use "wires/buses" to connect these blocks together. Put your mouse pointer at output "Ere_out[9..0]" of symbol "bfproc1" then click and drag it to input

"x_in[9..0]." A bus will be placed between the two components. See Fig. 6.26.

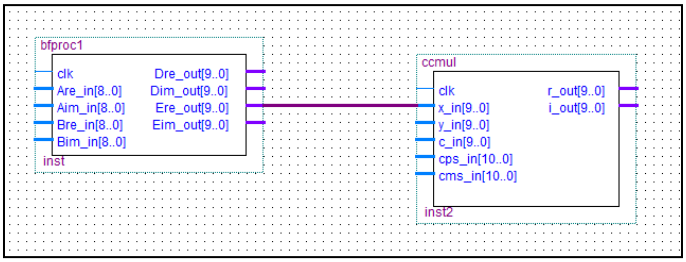


Figure 6.26

15. Repeat step 14 so that all the components are connected to each other and the entire system given in Fig. 6.3 is created. A completed system is shown in Fig. 6.27. Be very careful to not to make a mistake.

16. Click on "Save" under "File" menu to save the block diagram file. Name the file as "fft4_block.bdf." Make sure that "Add file to current project" is selected.

17. Right click on file "fft4_block.bdf" in "Project Navigator." Select "Set as Top-Level Entity" (See Fig 6.28.) Perform an Analysis and Synthesis.

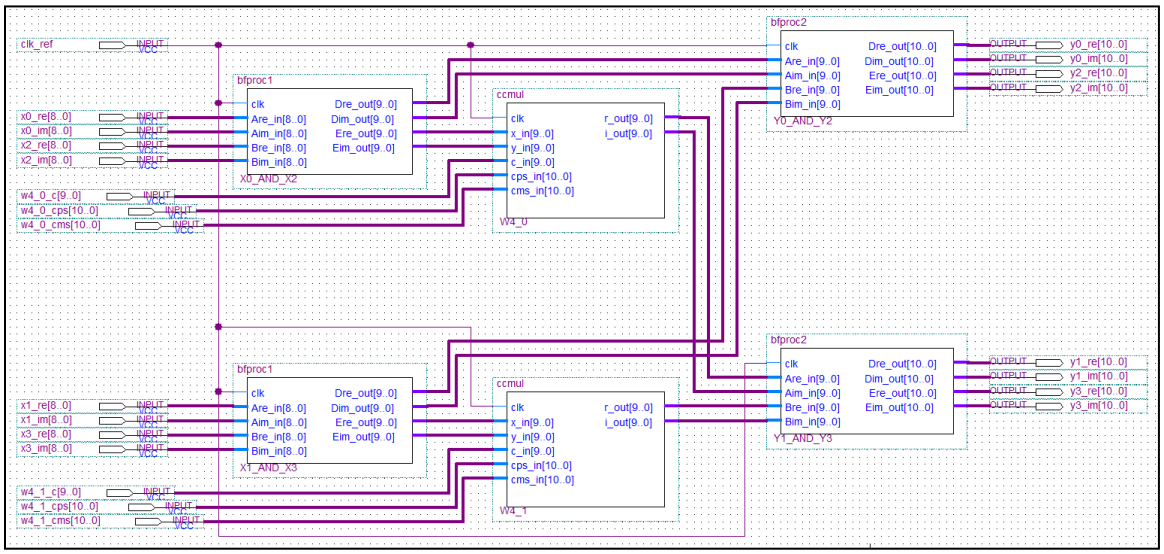


Figure 6.27

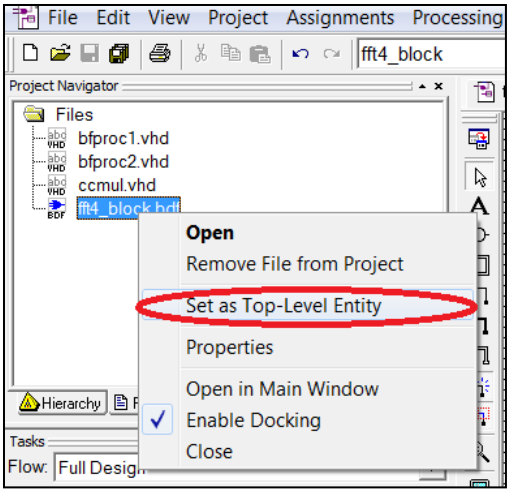


Figure 6.28

18. Create a new waveform file "fft4_block"to use it for functional simulation. Use the data given in Table 6.1 to check if the result is same as the VHDL Method given in section E. Save the result to include it in your report.

G. QUESTIONS AND REPORT

Your group report must be maximum **three pages long** including the figures in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment. Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. What are the differences between codes of "bfproc1.vhd" and "bfproc2.vhd"? Why do we need to use both symbols "bfproc1" and "bfproc2" in project "fft4_block.qpf"?

2. Estimate the number of total logic elements if we implement a length-8 FFT process by the same algorithm we used in this experiment. Give your reasoning. Do not try to write any code. Just think about the construction of length-8 FFT and try to use projects "bfproc.qpf" and "ccmul.qpf" to get the parameters you need.

3. Apply an impulse function with a magnitude of 1 as an input in the waveform file and observe the output via simulation. What do you expect to see? What is the simulation result?

   Hint: Calculate length-4 FFT of an impulse function with a magnitude of 1 using Matlab to find out the expected result. Refer to step 10 of section E. Equivalent value of input vector "x" representing an impulse function with a magnitude of 1 is

[1, 0, 0, 0].

4. (Optional) Do necessary modifications to add a frequency divider block to **divide the frequency by a half** between port "clk_ref" and signal "clk" in project "fft4_vhdl.qpf" and "fft4_block.qpf". Make a functional simulation to verify your design.

   Hint: You can refer to Experiment 5 for help.

H. FAQ FOR EXPERIMENT 6

1. How can I set input values to nodes in the waveform file for simulations?

Answer: Right click on the node you want to set in the waveform file for simulation. A menu will appear on the screen. Left click to the menu item "Value". Then a submenu will appear. Left click to the menu item "Arbitrary Value". You can set input values from the opened window.

FPGA EXPERIMENT 7
Modulation Using CORDIC Algorithm

TABLE OF CONTENTS

A. Introduction
B. CORDIC Algorithm
C. Universal Modulator Using CORDIC
D. Questions and Report
E. References

A. INTRODUCTION

In this experiment we will study an implementation of a universal modulator utilizing Coordinate Rotation Digital Computer (CORDIC) algorithm. We will simulate a VHDL file for AM modulation as a special case. You will be asked to improve the performance of the provided implementation of the CORDIC algorithm.

B. CORDIC ALGORITHM

Some parts of this section are taken from Wikipedia[2]. You are encouraged to study the concept in detail and not limit yourself only to this section. Refer to Section 2.8 of your textbook for details in implementing the CORDIC algorithm on FPGAs. CORDIC is a simple and efficient algorithm to calculate hyperbolic and trigonometric functions. It is commonly used when no hardware multiplier is available (e.g., simple microcontrollers and FPGAs) as the only operations it requires are addition, subtraction, shifting, and table lookup.

Here we will focus on only one of the six modes of the CORDIC algorithm which is calculating sine and cosine. The problem is to calculate (without multiplication)

$$\begin{aligned} x &= \cos\beta \\ y &= \sin\beta \end{aligned} \tag{7.1}$$

where the angle β is known. CORDIC starts with the vector

$$\mathbf{v}_0 = \begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 \end{bmatrix}^{\mathrm{T}} \tag{7.2}$$

where T is the transpose operator. The algorithm rotates this vector iteratively, and at every iteration step the rotation angle gets finer and finer. In the first iteration, the

[2] http://en.wikipedia.org/wiki/CORDIC

vector is rotated 45° to get the vector $\mathbf{v}_1$. Successive iterations will rotate the vector in the right direction by half the amount of the previous iteration result until the desired angular value is achieved (See Fig. 7.1.)

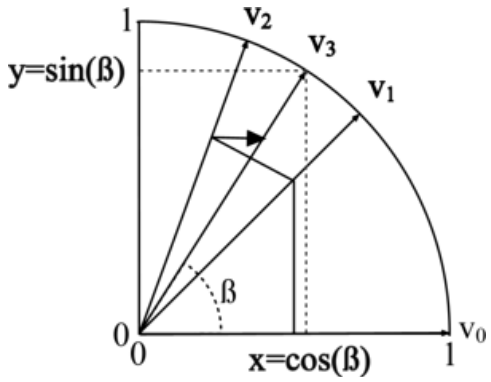


Figure 7.1: An illustration of the CORDIC algorithm in progress.

More formally, at every iteration the new vector is calculated using the following equation

$$\mathbf{v}_i = K_i \begin{bmatrix} x_{i-1} - \sigma_i 2^{-i} y_{i-1} \\ y_{i-1} + \sigma_i 2^{-i} x_{i-1} \end{bmatrix} \tag{7.3}$$

where the coefficient K_i is defined as

$$K_i = \frac{1}{\sqrt{1+2^{-2i}}} \tag{7.4}$$

and in the limit it converges to a real number which is around 0.6. The σ_i in Eq. (7.3) is equal to +1 or -1 depending on the relative position of the vector to the desired vector. It is determined by keeping track of how much we rotate at each iteration step and subtracting it from the desired angle.

CORDIC algorithm has an inherent error since it has to be stopped after a certain number of iterations. For example if we approximate 30 degrees with 3 iterations, the closest we can get is 45 – 26 + 14 = 33 degrees. (Note that $\tan^{-1}(2^{-0}) = 45°$, $\tan^{-2}(2^{-1}) = 26°$, and $\tan^{-1}(2^{-2}) = 14°$) The error is 3 degrees. Formally the angle error, ε, is equal to

$$\varepsilon = \beta - \hat{\beta} \tag{7.5}$$

where β is the desired angle and $\hat{\beta}$ is the approximate angle obtained via CORDIC.

C. UNIVERSAL MODULATOR USING CORDIC

Refer to Example 7.14 on page 455 in the textbook. We will AM modulate a DC signal which has a magnitude 100 for a while and later the magnitude changes to 25.

1. Open the project named "ammod.qpf" located under "VHDL/lab_7" in Quartus II. Carefully analyze the associated VHDL file. Note that on lines 27 and 29, K, coefficient defined in Eq. (7.4) is taken into account as 0.6 = 3/5. Since the idea behind the CORDIC is not using multipliers, this final step might seem a little odd. In practice, instead of multiplying the result with K, a lookup table is used.

2. Open the associated waveform file named "ammod.vwf." In the waveform file `r_in` is the signal to be AM modulated, `phi_in` is the angle of the carrier signal, and `eps` is the angle error defined in Eq. (7.5). What we are trying to accomplish is to calculate

$$\begin{aligned} x &= r\cos(\phi) \\ y &= r\sin(\phi) \end{aligned} \tag{7.6}$$

3. Let's run the following code in MATLAB to obtain the expected result

```
r = 100;
phi = [0,  30,  60,  90,  120,  150,...
    -180,  -150,  -120,  -90,  -60,  -30];
x = round(r * cosd(phi))
y = round(r * sind(phi))
```

And the result is

```
x =
  Columns 1 through 11
   100    87    50     0   -50   -87  -100   -87   -50     0
50
  Column 12
    87
y =
  Columns 1 through 11
     0    50    87   100    87    50     0   -50   -87  -100   -
87
  Column 12
   -50
```

4. Perform a functional simulation. Do you get the desired output? Is there a difference between MATLAB calculations and your simulation results? Why? (The output should look like the one given in Fig. 7.2.)

5. Compile the project, and record the total number of logic elements, Fmax, and worst case Tco. Include these results in your report. Refer to previous experiments if you have trouble with obtaining these results.

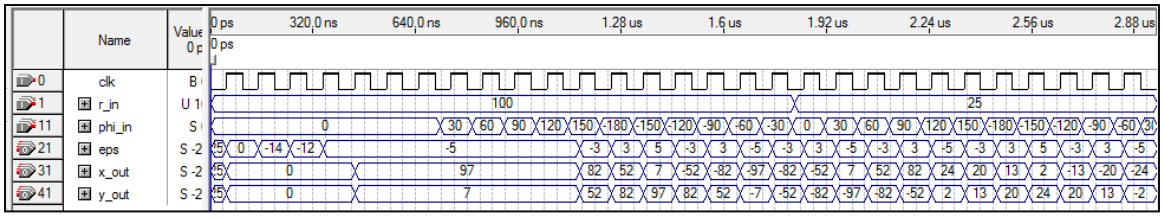


Figure 7.2

D. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures **excluding the codes** in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should answer to the **questions given in step 4 of section C** as well as to the following question:

1. Make necessary changes and/or additions to the VHDL file in order to reduce the angle error (eps.) Explain your steps. Perform a functional simulation to show that the error is reduced with your modifications. Include the code in the report.

   Hint: In the CORDIC algorithm, at every iteration step, the rotation angle gets finer and finer. Thus, the angle error is decreased.

2. Include the results obtained in step 5 of section C in your report.

E. REFERENCES

[1] http://en.wikipedia.org/wiki/CORDIC

FPGA EXPERIMENT 8
Memory Blocks, Megafunctions, Flash Memory, and Building a Simple Function Generator

TABLE OF CONTENTS

A. Introduction
B. Building a Function Generator Using Memory Blocks
C. Using the Flash Memory
D. Writing Data to the Flash Memory
E. Reading Data from the Flash Memory
F. Some Important Details
G. Questions and Report
H. References

A. INTRODUCTION

In this experiment, we will learn how to store one period of a periodic signal available in a file on our computer into one of the dedicated memories in the FPGA chip. To accomplish that, we will implement a Random Access Memory (RAM) using "Mega functions" available in ALTERA IP library. Then, we will learn how to modify the output of the memory and make the signal periodic. Next, we will utilize the Analog Devices ADV7123 High Speed VGA Digital/Analog Converter (DAC) available on the DE2-70 board to create an analog representation of the signal. Finally, we will learn how to write to and read from the FLASH chip available on DE2-70 board. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

The Cyclone II 2C70 FPGA located on the DE2-70 board has **dedicated memory resources** of size 4096 bits. They are called M4K blocks. Although the space is very limited, they are very easy to use. Moreover, there is a number of 250 blocks of M4K shipped with Cyclone II 2C70. There are also **external memory resources** consisting of two SDRAM modules of size 32MB, an SSRAM module of size 2MB, and a FLASH module of size 8MB located on the DE2-70 board. They are all connected to the Cyclone II FPGA via pins. The corresponding pin locations are available in the "DE2 70 pin assignments.csv" file. In this experiment, we are only interested in dedicated memory resources, namely M4K blocks. For further discussion on the topic please see the references section.

B. BUILDING A FUNCTION GENERATOR USING MEMORY BLOCKS

IMPORTANT: You will need a 10X oscilloscope probe. Please obtain one from the stock room before the lab starts.

1. Open Quartus II 9.1. Create a new project named “fun_gen” (Without the quotation marks!) Use the VHDL file named “fun_gen.vhd” provided to you as the top level entity. **It is very important that the correct device is selected while creating the project.**

2. Now we will create the “Memory Initialization File” (MIF) which will contain the initial values of our RAM implementation. Note that the values we enter in this file are actually written into the M4K memory during upload process. Therefore, it is not only good for simulation purposes, but also it is very handy for our real-world applications. Go to “File” menu and select “New.” Choose “Memory Initialization File” from the menu. Click on OK button. Set “Number of words” to 32 and set “Word Size” to 8. Click on OK button.

3. Open “samples.csv” file in Excel or in your favorite spreadsheet editor. Copy the 4 by 8 block of the cells (all of them) and paste them into the MIF in Quartus II (See Fig. 8.1.) Can you guess the *shape* of the periodic signal from the samples? Is it a random signal? Is it a sinusoidal, triangular, or square-wave signal? Plot the function in MATLAB, Excel, or in your favorite plotter to see. Save the MIF file and name it as “samples.mif.”

fun_gen.vhd

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	128	152	176	198	218	234	245	253
8	255	253	245	234	218	198	176	152
16	128	103	79	57	37	21	10	2
24	0	2	10	21	37	57	79	103

Figure 8.1: Memory Initialization File

4. We will implement the RAM structure given in Fig. 8.2 [1] Note that characteristics of this implementation are follows: There are 32 memory locations. Each memory location is of size 8 bits. The “input” and “address” buses and “write enable” line are registered (They operate *at the same time* at each rising edge of the clock.) However, the “output” bus is not registered (Its value is independent of clock.) Why do you think the registers, each connected to the same clock, are important to have in a RAM implementation?

Click on “Tools” top level menu and select “MegaWizard Plug-In Manager.” Select “Create a new custom megafunction variation” and click on “Next.” Select “RAM: 1-PORT” under “Memory Compiler” category. On the right, specify the full path of the project folder and give a name to the implementation file as “ram_for_samples.vhd.” Make sure the device is selected as “Cyclone II.” See Fig. 8.3.

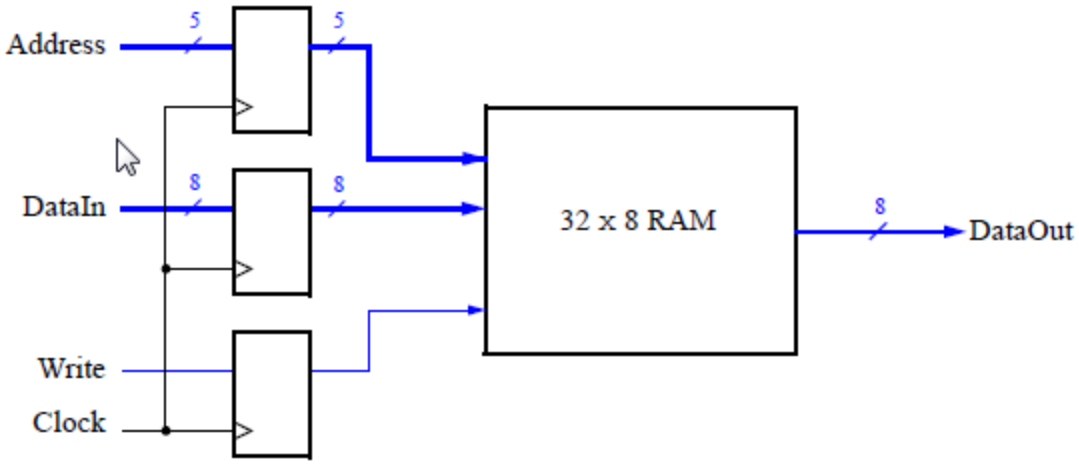


Figure 8.2: RAM Implementation [1]

5. Make sure the output is 8 bit wide and memory has 32 words. Select "memory block type" as "M4K." Use single clock (See Fig. 8.4.) On the next screen make sure that the 'q' output is NOT registered. Leave the other options default. Click on "Next" button. On the next screen select "Yes" and locate the "samples.mif" file you created in the previous steps. Leave other options on the further dialogs default and click on "Finish." Observe that "ram_for_samples.qip" and "ram_for_samples.vhd" files are created automatically for us. Make sure these files are in the project folder.

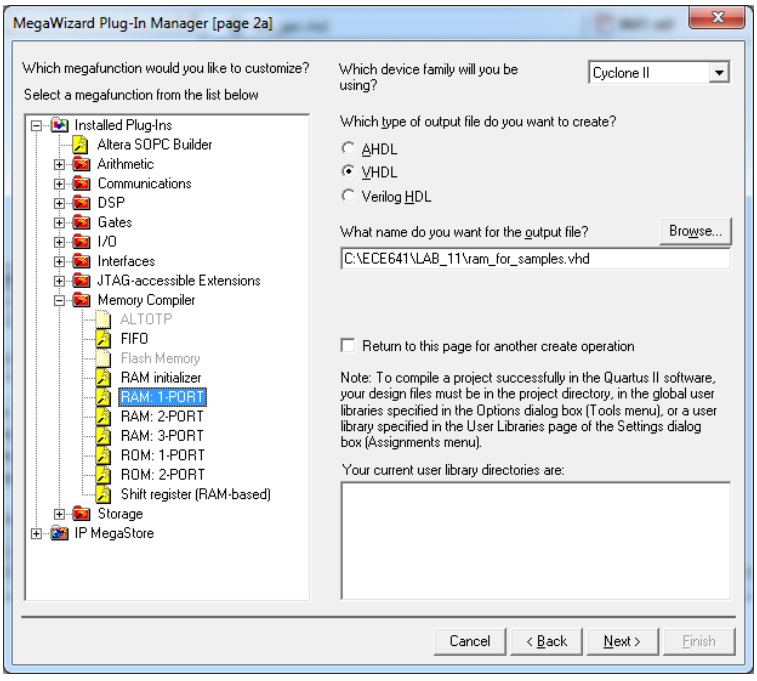


Figure 8.3

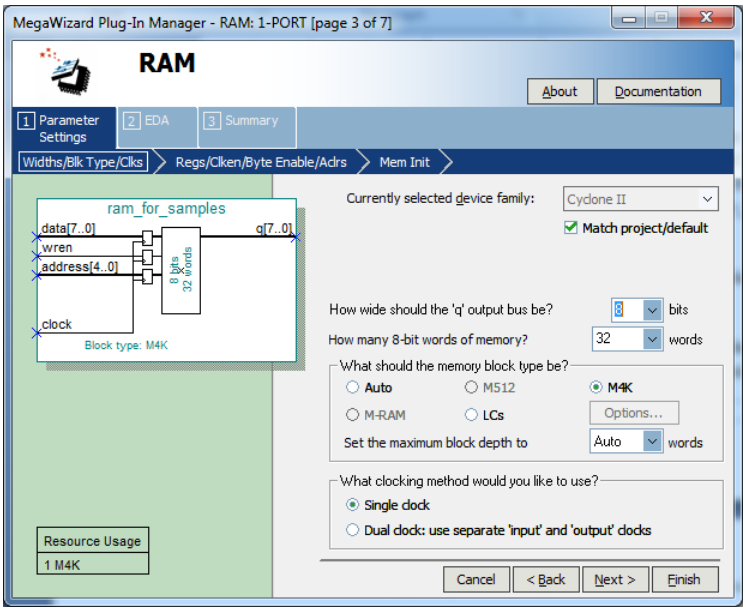


Figure 8.4

6. Now, we will use the automatically generated ram_for_samples entity as a "component" in our project. For your convenience the top level entity fun_gen is pre-coded. Open the "fun_gen.vhd" and read the code/comments and try to get a sense out of it. Here is how we introduced the ram_for_samples component in our design. The lines right after "ARCHITECTURE" keyword follows

```
COMPONENT ram_for_samples
      PORT (
            address     : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
            clock       : IN STD_LOGIC;
            data        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            wren        : IN STD_LOGIC;
            q           : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END COMPONENT;
```

Algorithm 8.1

What is the relationship between Alg. 8.1 and Fig. 8.2? Open the file named "ram_for_samples.vhd" and locate the keyword "ENTITY." Look at the lines right after it. Is there any connection between those lines and the ones given in Alg. 8.1? If yes, what?

Return to "fun_gen.vhd" and locate the following line

```
samples : ram_for_samples PORT MAP(address, slower_clk, memory_input,
wren, memory_output);
```

Algorithm 8.2

Perform a quick Internet research on your favorite search engine to find out why the keyword "PORT MAP" is used. Is Alg. 8.1 enough to *use an entity as a component in*

another entity? Provide a table showing the relation between the ports of the "ram_for_samples" and "fun_gen" entities.

7. Perform an analysis and synthesis. Make sure there is no error. Warnings are OK.

8. Open "Pin Planner" and refer to the "DE2 70 pin assignments.csv" file to assign locations to the design. Assign 50 MHz clock to "clk," five switches to "period_in_cycles," and a switch to "wren." Set "vga_sync," "vga_blank," and "vga_clk" to their corresponding locations (There is only one possible location for each!) For "vga_output" choose the oVGA_G[0] to oVGA_G[9]. Refer to Fig. 8.5 for the circuit of the DE2-70 board [3]. You may leave the "memory_input" blank. If you want, assign some switches to it in order to overwrite memory later.

9. Compile the project. Open the "Programmer" and download the image into your DE2-70 board. **Make sure the switch you assigned to "wren" port is in "off" position**. (We want to read data, not to overwrite it!) **Make sure at least one of the keys you assigned to "period_in_cycles" is in "on" position.**

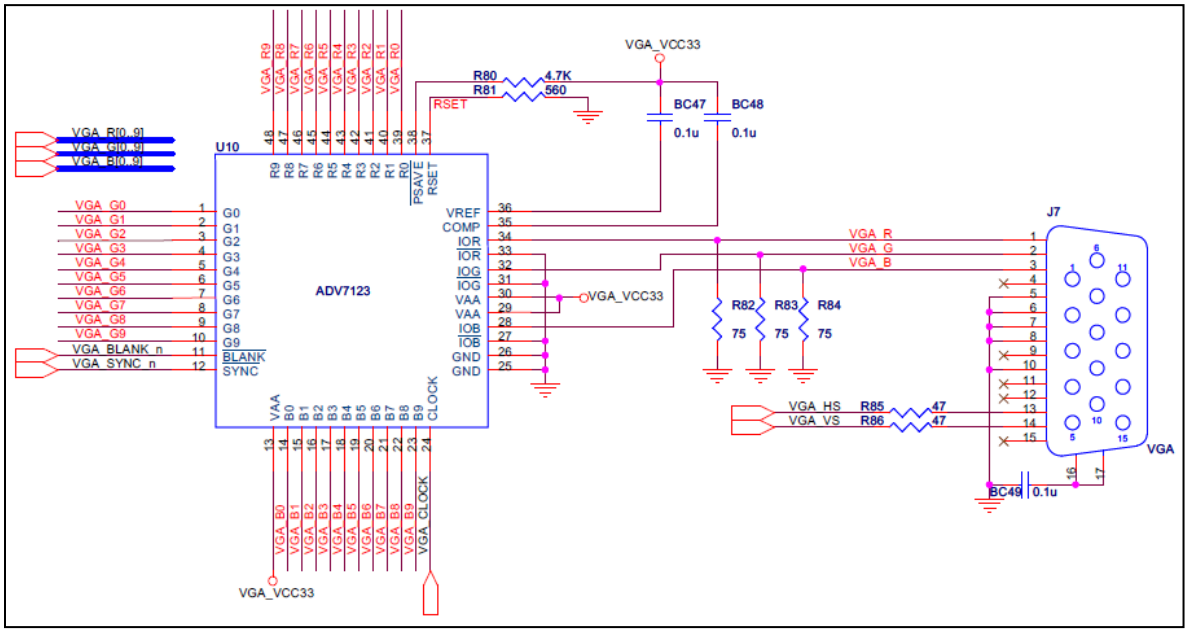


Figure 8.5: VGA DAC Circuit of DE2-70 Board [3]

10. Connect the oscilloscope probe to second pin of the VGA port. You can use fifth pin as ground for your measurement. Refer to Fig 8.6 for a sample connection setup. Observe the periodic analog signal in the scope.

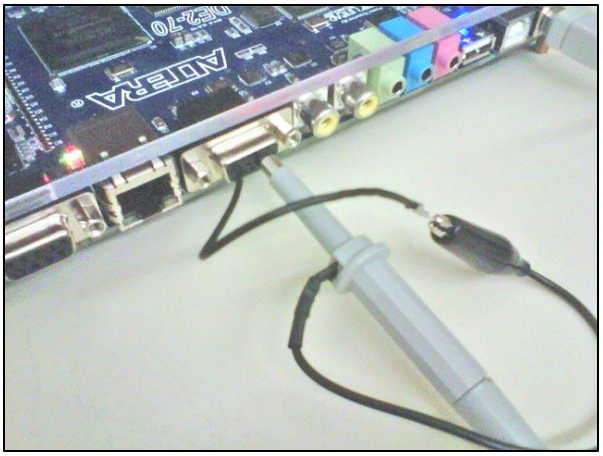


Figure 8.6: Probe and ground are connected to second and fifth pins of the VGA port, respectively.

11. On the laboratory PC connected to the scope, go to Start → All Programs → Keysight BenchVue. The program is shown in Figure 8.7. Then double click "Oscilloscope USBInstrument1" shown in a yellow rectangular in Figure 8.8. Then click "Get Current Screen" shown in a red rectangular in Figure 8.8. Save this image by clicking "Save Image" shown in a green rectangular in Figure 8.8. Include this snapshot to your report. See Fig 8.9 for an example snapshot.

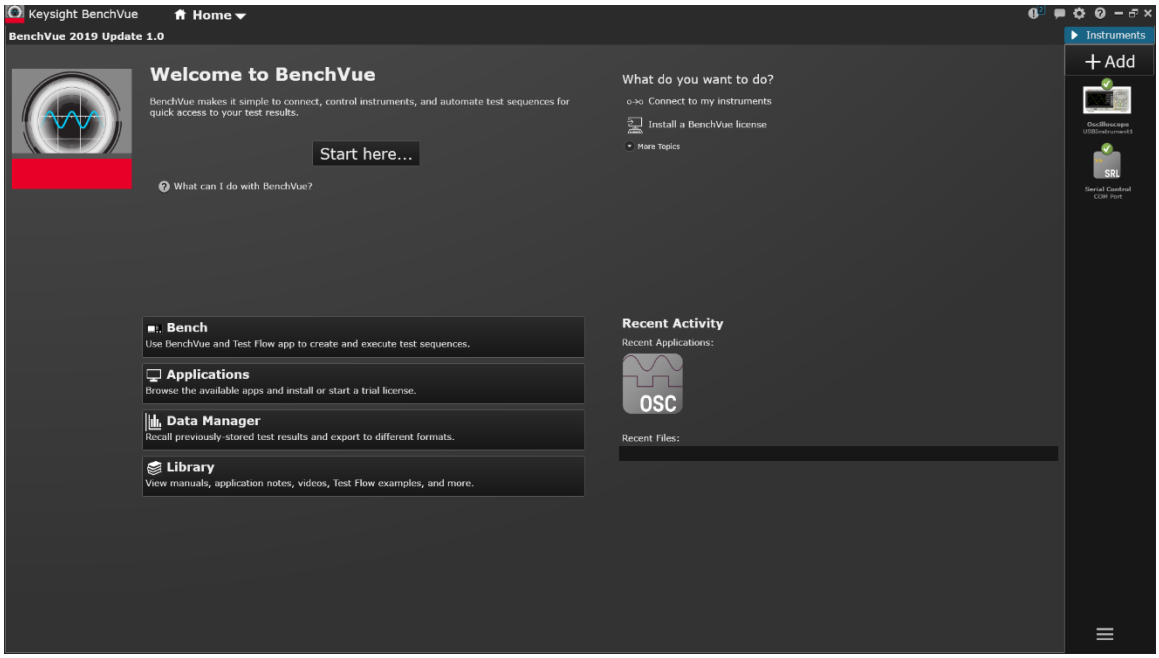


Figure 8.7: Keysight BenchVue.

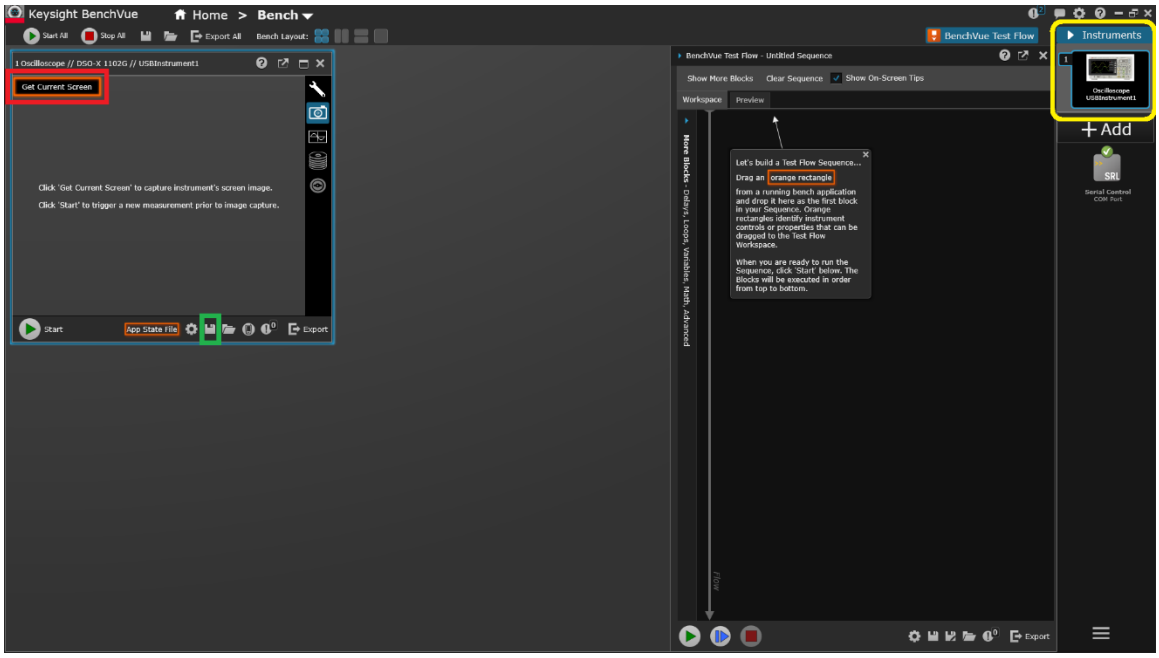


Figure 8.8: Steps for exporting the waveform of the selected oscilloscope using Keysight BenchVue.

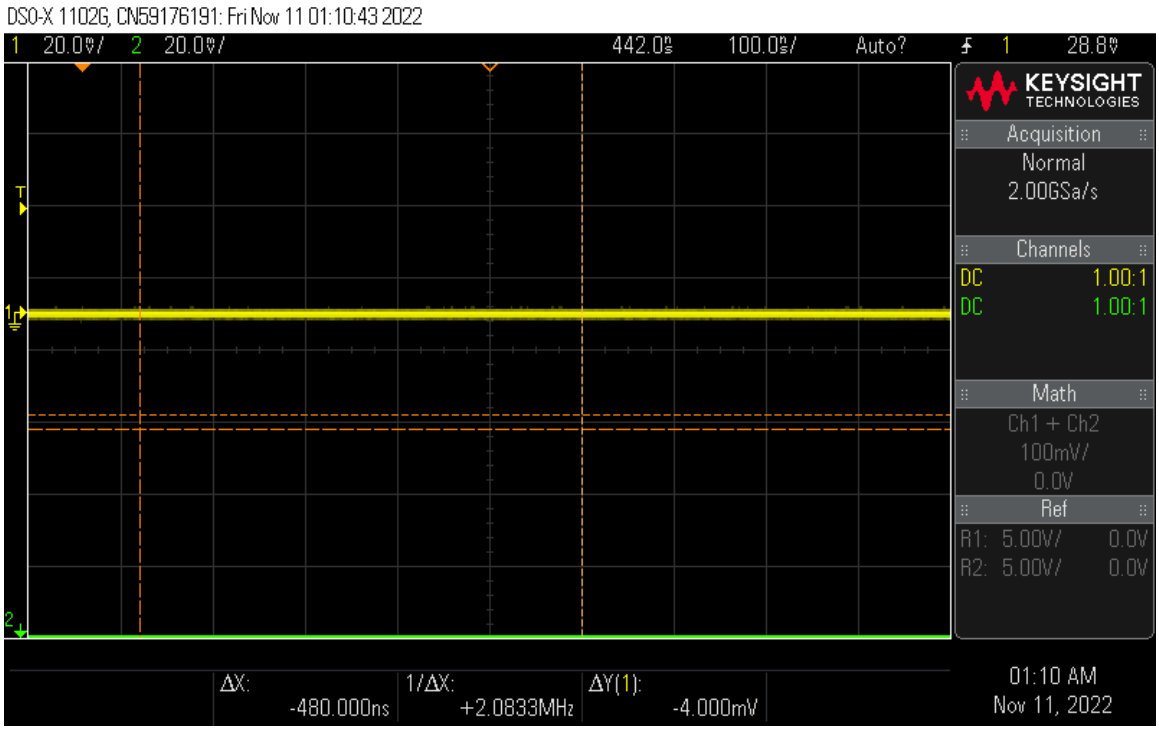


Figure 8.9: Snapshot of the CH1 scope screen. It is possible to get a good reading on 200 mV/div and 1 ms/div resolutions. Magenta and green colored sinusoids are generated using the same circuit with different period_in_cycles settings.

C. USING THE FLASH MEMORY

Let's start with erasing the contents of the flash memory. Note that the contents of the FLASH do not disappear even if you power down the DE2-70 board! Open DE2-70 control panel. Go to "Memory" tab. Select "FLASH" from the menu. Click on "Chip Erase" button. Wait for the chip to be erased. Note that when you erase the FLASH, all bits will be set as logic 1. If you want to write in logic 0, the FLASH will change the value but if you want to write 1 in, it just keeps the value. So remember that, you cannot change the value from 0 to 1 by write operation.

Important: Close the control panel after erasing is done.

D. WRITING DATA TO THE FLASH MEMORY

We will use project named "wt.qpf" to write data into the FLASH. Note that the address and data bus widths are 22 and 16 bits respectively. The most significant 18 bits of the address bus is hardwired to be 100000000000000000_2 and the least significant 4 bits are connected to SW17, SW16, SW15, and SW14. The least significant 8 bits of the data bus is hardwired to be 00000000_2 and the most significant 8 bits are connected to SW8, SW7, SW6, SW5, SW4, SW3, SW2, and SW1. Reset port is connected to SW0 and clock port is connected to KEY0.

Open the in project in Quartus II and compile it. Download it to the DE2-70 board. When the download is finished:

1. Set all the switches to "off" position. Set SW3 and SW2 to "on" position.

2. Set SW0 to "on" and back to "off" position. You should see that LEDG5 is "on."

3. Keep pressing on and releasing KEY0. The green LEDs will move from LEDG5 to LEDG0. When you see that LEDG0 is "on" you can stop pressing the KEY0. Note that, at this point, the binary data **00000110**00000000_2 (0600_{16}) is written to the FLASH memory space at address $100000000000000000\mathbf{0000}_2$ (200000_{16}.)

E. READING DATA FROM THE FLASH MEMORY

Again, open the control panel. Go to "Memory" tab. Select "FLASH" from the menu. In "Sequential Read" section, set the "address" field to 200000 and set the "length" field to 5 (Note that these numbers are hexadecimal.) Click on "Load Memory Content to a File" button. Set the file name as "content.hex" and click on "Save" button. Open the "content.hex" file with your favorite text (or hex) editor. You should read the following in the file:

0600FFFFFFFFFFFFFFFF

What we see here is that a number of 5 data blocks of length 16 bit stored in the FLASH chip starting from address 200000_{16}. As you may already noticed, every bit in the chip is wiped out and set to 1_2 (1111111111111111_2 = $FFFF_{16}$.) However, we overwrote the original $FFFF_{16}$ data in the memory space at address 200000_{16} with 0600_{16}.

F. SOME IMPORTANT DETAILS

a) The "first sector" cannot be written in FLASH. So **NEVER** set the first 18 bits of the address to something like 000000000000000000_2. How big the first sector of a FLASH chip is determined by the model. Refer to the datasheet located in your "ECE641 Materials" folder for further details.

b) In wt.vhd, we use a finite state machine. From state w1 to state w9 we implement the write operation. You are advised not to modify anything here. However you can modify the address and the data from states w10 to w12. Here is a more detailed explanation of the code:

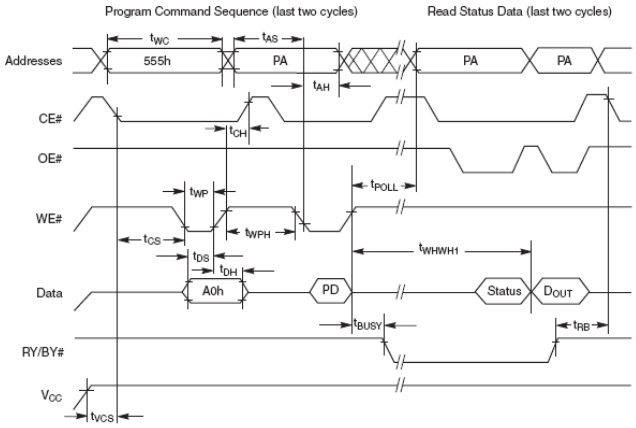
1. From line 5 to line 11: set input and output port.
2. From line 21 to line 23: set the signals for write.
3. From line 26 to line 33: change the state with every clock tick.
4. From line 45 to line 107: the first 3 cycles for write operation.
5. From line 110 to line 128: the last cycle of write operation.

See Figs. 8.10 and 8.11 and the datasheet of the FLASH chip for further details.

c) If you get the following error: "Error: Can't place multiple pins assigned to pin location Pin_AD25 (IOC_X95_Y2_N1)" be advised that this is due the fact that Pin_AD25 (SW7) on DE2-70 board is used for multiple purposes. To resolve the issue:

1. In Quartus-II select Assignments → Device...
2. Click on "Device and Pin Options..."
3. Select the tab "Dual-Purpose Pins."
4. In the list of "Dual-purpose pins:" change the "value" property of nCEO to "Use as regular I/O". Click "OK."

Figure 16.5 Program Operation Timings



Notes
1. *PA = program address, PD = program data, D_{OUT} is the true data at the program address.*
2. *Illustration shows device in word mode.*

Figure 8.10

Figure 10.2 Command Definitions (x16 Mode, BYTE# = V_{IH})

Command Sequence (Note 1)		Cycles	Bus Cycles (Notes 2–5)											
			First		Second		Third		Fourth		Fifth		Sixth	
Read (Note 5)		1	RA	RD										
Reset (Note 6)		1	XXX	F0										
Autoselect (Note 7)	Manufacturer ID	4	555	AA	2AA	55	555	90	X00	0001				
	Device ID (Note 8)	6	555	AA	2AA	55	555	90	X01	227E	X0E	(Note 19)	X0F	(Note 19)
	Device ID (Note 9)	4	555	AA	2AA	55	555	90	X01	(Note 18)				
	Secured Silicon Sector Factory Protect	4	555	AA	2AA	55	555	90	X03	(Note 10)				
	Sector Group Protect Verify (Note 11)	4	555	AA	2AA	55	555	90	(SA)X02	00/01				
Enter Secured Silicon Sector Region		3	555	AA	2AA	55	555	88						
Exit Secured Silicon Sector Region		4	555	AA	2AA	55	555	90	XXX	00				
Program		4	555	AA	2AA	55	555	A0	PA	PD				
Write to Buffer (Note 12)		3	555	AA	2AA	55	SA	25	SA	WC	PA	PD	WBL	PD
Program Buffer to Flash		1	SA	29										
Write to Buffer Abort Reset (Note 13)		3	555	AA	2AA	55	555	F0						
Unlock Bypass		3	555	AA	2AA	55	555	20						
Unlock Bypass Program (Note 14)		2	XXX	A0	PA	PD								
Unlock Bypass Reset (Note 15)		2	XXX	90	XXX	00								
Chip Erase		6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
Sector Erase		6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30
Program/Erase Suspend (Note 16)		1	XXX	B0										
Program/Erase Resume (Note 17)		1	XXX	30										
CFI Query (Note 18)		1	55	98										

Figure 8.11

G. QUESTIONS AND REPORT

There is **no limit** for number of sheets. Report is due in **one week**. "Bonus" parts are due in **two weeks.**

1. Provide your answers to questions raised in the manual. Clearly specify the questions.

2. What is the significance of generics M and N in "fun_gen.vhd" file? Decrease both of them to very small numbers, create a vector/waveform file and simulate the project. Provide your simulation results.

3. Explain what we do with "freq_divider", "repeater," and "main" processes? What are the importances and functionalities of the processes? Report the signals in the "sensitivity list" of each process. Do we really need the "main" process? Can we remove the main process, and place its contents somewhere else in the VHDL code? If yes, where?

4. (Optional) Create another RAM implementation. Create another MIF file that has one period of a periodic triangular function. Use it as a new component in your design. Modify the design so that the user can choose between the "sinusoid" and "triangle" for the output.

5. (Optional) Continue 4 by adding square and saw-tooth functions as well.

6. (Optional) Increase the number of bits of each memory location from 8 to 10.

7. (Bonus) Modify the main function such that the value at current address and value at the previous address are averaged and then written to another RAM implementation. The VGA input is connected to the latter.

8. (Bonus) Modify the code so that the resolution of the sinusoid is better. Compare the sinusoids generated by the original and modified codes on the scope and show that you improved the resolution. Note that you will need larger memory space for this modification.

9. (Bonus) Display the frequency of the signal being generated with 7-segment displays.

10. (Bonus) Design a digital "VU Meter" with LEDs. Connect it in parallel to the VGA digital input. See http://en.wikipedia.org/wiki/VU_meter for reference. Note that you may skip averaging, if you like.

H. REFERENCES

1. Laboratory Exercise 8, "Memory Blocks." Shipped with DE2-70 User Manual CD-ROM. (This document is available in the "ECE 641 Materials / DE2 additional lab exercises" folder.

2. ADV7123 datasheet in "ECE 641 Materials/Datasheets/VGA DAC" folder.

3. ALTERA DE2-70 User Manual Version 1.08

I. FAQ FOR EXPERIMENT 8

1. I downloaded the code but I cannot see anything on the oscilloscope. Why?

Answer: Please check your pin assignments carefully in the pin planner. Also, make sure that you connected the oscilloscope probe to second pin of the VGA port.

2. Where can I get oscilloscope probe?

Answer: From the stock room on the third floor.

FPGA EXPERIMENT 9
Fast Convolution Using Number Theoretic Transform

TABLE OF CONTENTS

A. Introduction
B. Implementation of Fast Convolution Using Fermat NTT on FPGA
C. Questions and Report

A. INTRODUCTION

In this experiment, we will perform a cyclic convolution using Number Theoretic Transform (NTT). It has been shown that, in some cases, NTTs can outperform FFT based systems.[3] You are strongly encouraged to read pages 401 – 407 of your textbook to get a "glimpse" of this rich and important concept. What we will do in this experiment is a direct result of Theorem 7.6 given on page 406 of the textbook.

As a side track we will continue improving our simulation skills in terms of creating a waveform file. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

B. IMPLEMENTATION OF FAST CONVOLUTION USING FERMAT NTT ON FGPA

1. Refer to Example 7.7 in the textbook on page 406 for this experiment. We are asked to use a Fermat NTT to perform a circular convolution on $x(n)$ and $h(n)$ given as

$$\begin{aligned} x(n) &= \{1, 1, 0, 0\} \\ h(n) &= \{1, 0, 0, 1\} \end{aligned} \tag{9.1}$$

In order to accomplish this, we need to perform following algebraic calculations

$$\begin{aligned} \mathbf{X} &= \mathbf{T}\mathbf{x} \\ \mathbf{H} &= \mathbf{T}\mathbf{h} \\ \mathbf{Y} &= \mathbf{X} \times \mathbf{H} \bmod 257 \\ \mathbf{y} &= \mathbf{T}^{-1}\mathbf{Y} / 4 \end{aligned} \tag{9.2}$$

where x is the term-by-term multiplication operator. Transform matrix **T** and inverse transform matrix $\mathbf{T}^{-1}$ are given in Eq. 7.12 on page 407 of the textbook. Pre-calculated results are also given on the same page. As you know, we can also perform circular

[3] See section 7.1 on page 401 of the textbook.

convolution via forward and inverse Fourier transform. Let's run the following MATLAB code to obtain the expected result:

```
x = [1, 1, 0, 0];
h = [1, 0, 0, 1];
y = ifft(fft(x).*fft(h))
```

The result is

```
y =
 2     1     0     1
```

which confirms the result given on Page 407 in the textbook.

3. Let's open the project "ntt.qpf" located under "VHDL/lab_9/ntt" folder in Quartus II and simulate it. The project contains the "ntt.vhd" code which performs the necessary tasks for this experiment. You are encouraged to carefully examine this file in order to improve your understanding of VHDL. Once the project is opened, perform an "Analysis and Synthesis."

4. Create a new waveform file named "ntt.vwf" to use it for simulation. Refer to step 9 of section E in the sixth experiment for details (how to set value to a signal in a waveform file). Here are the steps you need to perform to create this specific waveform file:

a) Set the `clk` signal period to 10 ns.

b) Set `reset` signal to 1 from 0 to 12 ns and set it to "0" for the rest.

c) Set `x_in` to 1 for the 2nd and the 3rd rising edges of the clock, and set it to 0 for the rest, as shown in Figure 9.1.

d) Set `h_in` to 1 for the 2nd rising edge of the clock, and set it to 0 for the 3rd and 4th rising edge of the clock, set it to 1 again for the 5th rising edge of the clock, and set it to 0 for the rest. Your waveform file should look like the one given in Fig. 9.1.

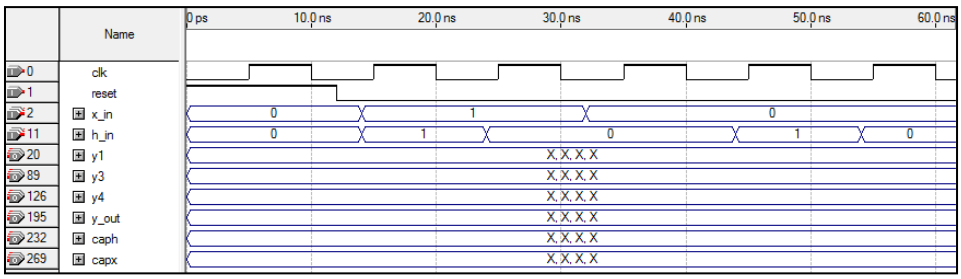


Figure 9.1

5. Perform a functional simulation using the waveform file you created. Keep the result in order to include it in your report. Your result should look like the one given in Fig 9.2. Note that we are able to obtain the expected output at `y_out`. If you haven't already noticed, the result is in reversed order. So are the results for `capx` and `caph`. If you click on the plus sign next to one of your signals, you will notice that everything is in its correct place. The problem is due to Quartus II showing the MSB on top instead of LSB.

6. Compile the project, and record the total number of logic elements, fmax, and worst case Tco. Include these results in your report. Refer to previous experiments if you have trouble with obtaining these results.

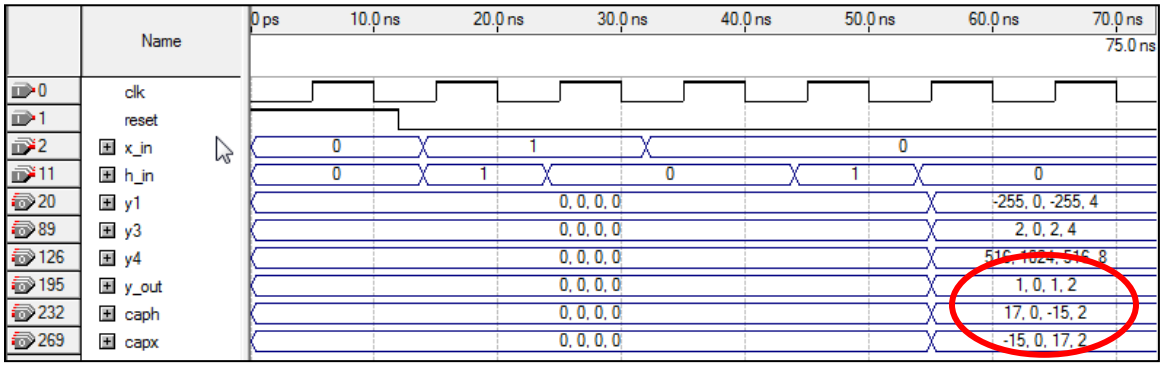


Figure 9.2

C. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. Deduce the results given on page 407 of textbook using Eqs. 9.1 and 9.2 given in Section B of this experiment, and using forward and inverse T matrices given in Eq. 7.12 on page 407 of the textbook. You may use MATLAB if you like.

2. Refer to Example 3.2 on page 179 of the textbook and the associated Quartus II project file named "fir_srg.qpf" located in your "VHDL/lab_9/fir_srg" folder. Compile the project and record the total number of logic elements, fmax, and worst case Tco. Compare those to the ones you obtained for "ntt.qpf." How do you compare these two designs in terms of speed and resource used?

3. Provide an alternative solution to calculate the cyclic convolution of two length-4 signals in the form of a block diagram which utilizes FFT (and possibly inverse FFT) blocks. **Important:** Don't try to code a VHDL or try to create a schematics file for that! You are asked to *draw* a block diagram such as the one given in Fig. 6.3. Based on the speed and resource usage of the "fft.qpf" you obtained in experiment 6, estimate the speed and resource usage of your design and compare it to the ones you obtained for "ntt.qpf."

   Hint: Note that your design will need only two stages in FFT and IFFT blocks. Assume that the relation between number of stages and speed is linear. However you need to be careful when calculating the speed estimation keeping in mind that many processes can be run in parallel on an FPGA.

   You have to be very careful in relation between number of stages and resources. Check how many butterfly processors are used in 8-length and 4-length FFT blocks. Do you think this relation is linear?

Make sure that you include the question numbers to your answers.

D. FAQ FOR EXPERIMENT 9

1. Even though I followed the steps exactly as in the lab manual, my results do not look like the one given in Fig. 9.2. Why?

Answer: The nodes "y1, y3, y4, y_out, caph, capx" in the waveform simulation file are displayed in the vector form. The waveform may not show the correct results due to a little bug if the displayed node is in vector form. When you add each element of nodes "y1, y3, y4, y_out, caph, capx" separately rather than as a vector in the waveform file, you will get the correct results. For example, you should add the y1[0], y1[1], y1[2] and y1[3] as a separate node in the waveform simulation file.

2. How can I set input values to nodes in the waveform file for simulations?

Answer: Right click on the node you want to set in the waveform file for simulation. A menu will appear on the screen. Left click to the menu item "Value". Then a submenu will appear. Left click to the menu item "Arbitrary Value". You can set input values from the opened window.

FPGA EXPERIMENT 10
Error Control and Cryptography

TABLE OF CONTENTS

A. Introduction
B. Linear Feedback Shift Register (LFSR)
C. Cryptography Algorithms
D. Questions and Report

1. INTRODUCTION

In this experiment we will implement a simple encryption/decryption system on FPGA. For that purpose we will utilize a linear feedback shift register (LFSR) to generate a pseudo-random key and use this key to cipher and decipher a sequence. The block diagram of the system is given in Fig. 10.1. You are referred to Section 7.2.4 of the textbook for further details. Interested students are strongly encouraged to read the complete section for more complicated algorithms.

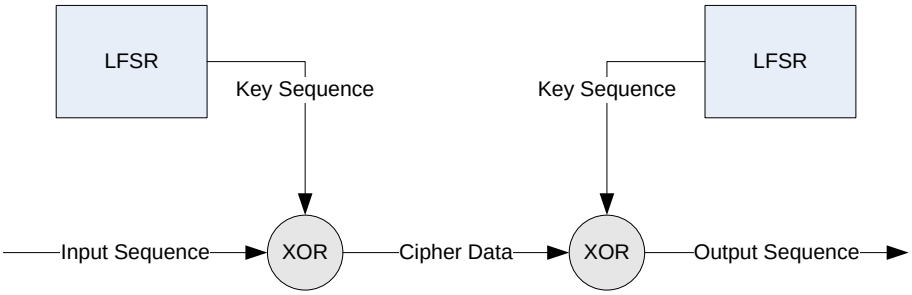


Figure 10.1

Note that in Fig. 10.1 the goal is to have the "output sequence" completely identical to the "input sequence." Also note that the "cipher data" bit stream can only be decrypted by a receiver equipped with the same LFSR starting at the same state. Although an attacker might discover the generating polynomial, the starting state has to be known as well. The brute-force search is a practically impossible problem. Though the algorithm is simple, it is yet powerful.

2. LINEAR FEEDBACK SHIFT REGISTER (LFSR)

Refer to Example 7.12 on page 437 of the textbook. In this example a key sequence is generated using a length-6 LFSR. The polynomial used in the example is

$$y = 1 + x + x^6 \tag{10.1}$$

A pre-coded VHDL file named "lfsr.vhd" is available under your "VHDL/lab_10" folder. We need to modify this code in order to make it compatible with higher-level VHDL codes we will use in the next sections.

1. First, add a generic constant named as `sdata` and set its value to 1. This will be the zero-indexed value of our key sequence. Moreover, add an extra input port named as `reset`. This will help us later to store the constant into the LFSR. Here is the corresponding original and modified part of the VHDL code:

Original:

```
ENTITY lfsr IS                          ------> Interface
  PORT ( clk : IN  STD_LOGIC;
```

Modified:

```
ENTITY lfsr IS                          ------> Interface
  GENERIC (sdata: integer:=1);
  PORT ( reset: in std_logic;
         clk : IN  STD_LOGIC;
```

Make sure that you completely understand the modifications we have done in this step. It is a nice example of how to introduce a new constant and a new port in a VHDL file.

2. Next, modify the first process in the code to utilize the reset functionality. We need to wait for `reset` to go from 1 to 0 to start the process. Whenever it is equal to 1, we store the constant `sdata` into the `ff` variable which is the 6-bit register for the values of the pseudo-random key itself. Here are the corresponding original and modified parts of the VHDL code:

Original:

```
PROCESS
BEGIN
WAIT UNTIL clk = '1';
    ff(1) <= NOT (ff(5) XOR ff(6));
    FOR I IN 6 DOWNTO 2 LOOP
      ff(I) <= ff(I-1);
    END LOOP;
END PROCESS;
```

Modified:

```
PROCESS(reset, clk)
BEGIN
IF reset='1' then
       ff<=conv_std_logic_vector(sdata, 6);
ELSIF clk'event and clk='1' then
       ff(1) <= NOT (ff(5) XOR ff(6));
       FOR I IN 6 DOWNTO 2 LOOP
           ff(I) <= ff(I-1);
       END LOOP;
```

```
    END IF;
    END PROCESS;
```

Carefully analyze the modifications we have done in this step. See how we utilized IF, ELSIF block, how we passed the constant into the ff register etc. This also a nice example to understand how to utilize reset and clock-rising edge functionalities in a VHDL file. Save the modified program when you are done.

C. CRYPTOGRAPHY ALGORITHMS

Now the LFSR is ready. Nest, we will need two more VHDL files: first one is the "encryption.vhd" which is nothing else but an XOR operator, and the second one is the "cryptproc.vhd" which is the top-level VHDL file, which performs the ciphering and deciphering process. These files are located in "VHDL/lab_10" folder.

1. Create a new project named "cryptproc.qpf." Include "lfsr.vhd," "encryption.vhd," and "cryptproc.vhd" in the project. The top-level entity of the project should be "cryptproc.vhd." Refer to the fifth experiment if you need help whit this step.

2. Perform and Analysis & Synthesis. In order to understand the relation between the design files expand everything in "Project Navigator" window (See Fig. 10.2) Note that "cryptproc.vhd" uses two components named "cipher" and "decipher" and these components are implemented by "encryption.vhd." The "lfsr.vhd" is used as a component in "encryption.vhd" and the name of the component is "LFSR1."

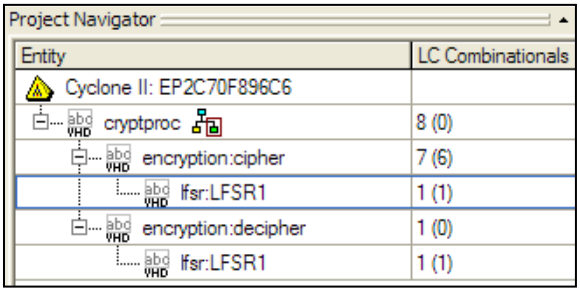


Figure 10.2

Take a look at line 26 of "cryptproc.vhd" The reset and clk signals are fed into the component. More importantly, `x` is fed as the input and the output is fed back into `cipherdata`:

```
    PORT map( reset, clk, x, cipherdata);
```

Similarly, on line 29 `cipherdata` is fed as the input and the output is fed back into `y`:

```
    PORT map( reset, clk, cipherdata, y);
```

It is very beneficial for you to completely understand the modular structure of this project. Very large-scale projects are better managed when everything is modular.

3. Create a waveform file named "cryptproc.vwf." Refer to step 9 of section E in the sixth experiment for details (how to set value to a signal in a waveform file). Here are the steps you need to perform to create this specific waveform file:

a) Add signals named `clk`, `reset`, `x`, `y`, and `cipherdata`.

b) Define the period of clock (`clk`) as 20 ns.

c) Set `reset` signal to 1 from 0 ns to 25 ns and 0 for the rest

d) Set `x` signal to 1 from 0 ns to 30ns, 2 from 30 ns to 50 ns, 3 from 50 ns to 70 ns, and 0 for the rest.

Your waveform file should look like the one given in Fig. 10.3.

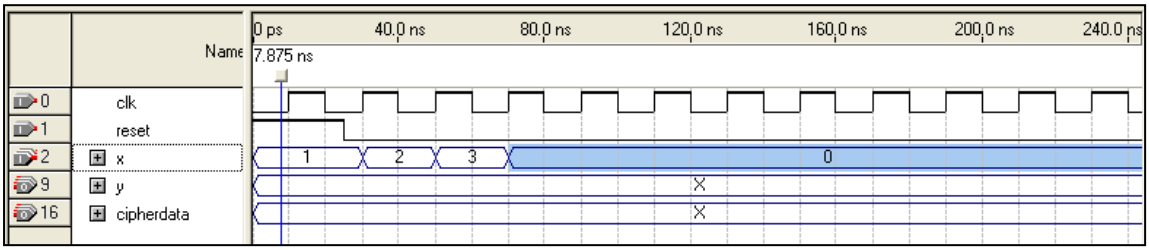


Figure 10.3

4. Perform a functional simulation using the waveform file you created. Keep the result in order to include it in your report. Do you get an identical input and output?

5. Compile the project, and record the total number of logic elements, Fmax, and worst case Tco. Include these results in your report. Refer to previous experiments if you have trouble with obtaining these results.

D. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. Though the title of the experiment is “Error Control and Cryptography,” we did not mention anything about “Error Control.” Describe how this design can be used as an “Error Control” system.

2. Take a second look at our modification on “lsfr.vhd” file. Find out what the function named `conv_std_logic_vector()` specifically does. What happens if we don’t use it?

3. What happens when you change the zero-indexed value of the key sequence, i.e. `sdata`? What do you expect to see? Check via functional simulation. (Hint: Changing line 6 in lfsr.vhd or line 16 of encryptor.vhd will not help you. Locate in which file and on what line we “map” the lfsr.vhd as well as its `sdata` value.)

4. (Bonus Question) In “lfsr.vhd” we defined the zero-indexed value of the key sequence as a constant. Introduce a new port in the VHDL file to be used instead of that constant. Modify your waveform file accordingly. Run the simulations again. (Your report might be three pages long if you decide to answer to this question.)

Make sure that you include the question numbers to your answers.

FPGA EXPERIMENT 11
Adaptive Filters and LMS Algorithm

TABLE OF CONTENTS

A. Introduction
B. Adaptive Filters and LMS Algorithm
C. Direct Implementation of LMS Algorithm
D. Pipeline Implementation of LMS Algorithm
E. Questions and Report
F. References

A. INTRODUCTION

In this experiment, we will study FPGA implementation of the Least Mean Squares (LMS) algorithm and its use in adaptive filters. First we will revisit the adaptive filters and LMS algorithm. Then, we will analyze and simulate a VHDL file which directly implements the LMS algorithm on a two-tap adaptive filter. Next, we will simulate another implementation method of the algorithm called "pipeline implementation" and compare its performance and resource cost to those of the direct method.

B. ADAPTIVE FILTERS AND LMS ALGORITHM

Some parts of this section are taken from Wikipedia[4]. You are encouraged to study the concept in detail and not limit yourself only to this section. Refer to Chapter 8 of your textbook for details of the concept. Most linear adaptive filtering problems can be formulated using the block diagram given in Fig. 11.1.

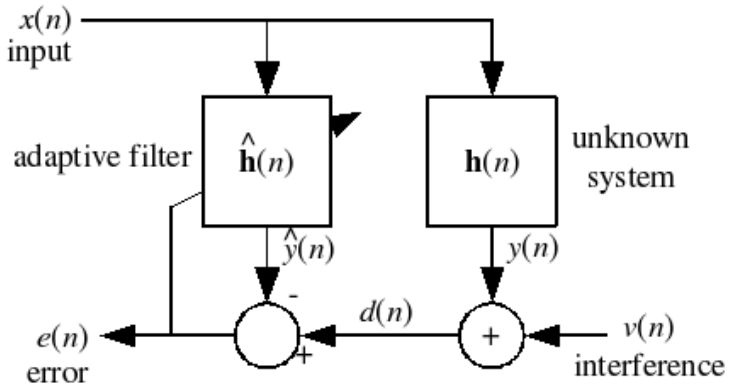


Figure 11.1

[4] http://en.wikipedia.org/wiki/Least_mean_squares_filter

In Fig. 11.1, an unknown system with its unit-sample response $h(n)$ is to be identified. The adaptive filter $\hat{h}(n)$ adapts itself to mimic $h(n)$ as close as possible. The adaptive filter coefficients can be found by solving the Wiener-Hopf equations. However, that solution requires the input and output statistics as well as it involves matrix inversion. The idea behind LMS filters is to use the method of "steepest descent" to find a unit-sample response coefficient vector $h(n)$ that minimizes a cost function defined as

$$C(n) = E\left\{|\, e(n)\,|^2\right\} \tag{11.1}$$

where $E\{\cdot\}$ is the expectation operator and $e(n)$ is the error signal defined as

$$e(n) = d(n) - y(n) \tag{11.2}$$

where $d(n)$ is the reference (desired) output and $y(n)$ is the adaptive filter output. Refer to Widrow-Hoff LMS Algorithm given on page 488 in the textbook for details of the algorithm.

C. DIRECT IMPLEMENTATION OF LMS ALGORITHM

1. Read and understand the example given in Example 8.1 on page 485 in the textbook. Summarize with a couple of sentences what we are trying to accomplish in the example. What are the optimum coefficients of the filter taps in the example?

2. Now we will examine the pre-coded VHDL file named "fir_lms.vhd" available in your "VHDL/lab_11/fir_lms" folder. It is the FPGA implementation of Example 8.1 (on page 485) via Widrow-Hoff LMS Algorithm (on page 488.) We will go over the steps of Algorithm 8.2 on page 488 in the textbook, and highlight the parts of the VHDL code that corresponds to each step.

The Widrow-Hoff LMS algorithm to adjust the *L* coefficients of an adaptive filter uses the following steps:

a. *Initialize the (L x 1) vectors* $\mathbf{f} = \mathbf{x} = \mathbf{0} = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}$. This step is bypassed in the code since the default register values are 0 on the FPGA.

b. *Accept a new pair of input samples* $\{x(n), d(n)\}$ *and shift* $x(n)$ *in the reference signal vector.* As we see from the process named `dsxt` and the process named `Store` between lines 40 – 56, `x_in` is shifted to `x(0)`, `x(0)` is shifted to `x(1)`, `d_in` is shifted to `d` at each clock event. At the same time, 8-bit signal `d` is expanded to 16 bits as 2's complement in the `dsxt` process. This is required for line 78 which is `e <= sxtd - sxty`. Note that `sxty` is of 16 bits due to 8-bits by 8-bits multiplication.

c. *Compute the output signal of the FIR filter via* $y(n)=\mathbf{f}^{\mathrm{T}}(n)\mathbf{x}(n)$. In order to calculate the output $y(n)=f(0)x(0)+f(1)x(1)$, two FPGA embedded multipliers were instantiated and then their product is added up and divided by 128 since `x_in` is an input number reference to 128. Next, `y` is expanded to 16-bits 2's complement number (See lines 58 – 76.)

d. *Compute the error function with* $e(n)=d(n)-y(n)$. `e` is computed on line 78 and turned into 2's complement number named `emu` via dividing it by 2. Then, it is multiplied by `x(0)` and `x(1)`, and the outcomes are assigned to `xemu(0)` and `xemu(1)` respectively by using the two embedded multipliers given on lines between 79 and 89.

e. *Update the filter coefficients according to* $\mathbf{f}(n+1)=\mathbf{f}(n)+\mu e(n)\mathbf{x}(n)$ *and continue with step b.* Coefficients `f(0)` and `f(1)` are updated in the process named `Store` at each clock (See lines 54 and 55.) Note that in this step, `xemu` is divided by 2 implicitly.

3. Let's rewrite the equations of the algorithm specific to this example. We have

$$\begin{aligned} f_0(n+1) &= f_0(n)+\mu e(n)x_0(n)/128 \\ f_1(n+1) &= f_1(n)+\mu e(n)x_1(n)/128 \\ y(n) &= \left[\, f_0(n)x_0(n)+f_1(n)x_1(n)\,\right]/128 \\ e(n) &= d(n)-y(n) \end{aligned} \tag{11.3}$$

where division by 128 is there since the input is referenced to 128 (quantization.) Open the waveform file named "fir_lms.vwf." In the waveform file `x_in` is the filter input and `d_in` is the reference (desired) output. If we apply Eq. (11.3) to these signals via MATLAB or spreadsheet software we get the results listed in Table 11.1.

4. Perform a functional simulation. Do you get the expected output? The output should look like the one given in Fig. 11.2.

5. Compile the project, and record the total number of logic elements, Fmax, and worst case Tco. Include these results in your report.

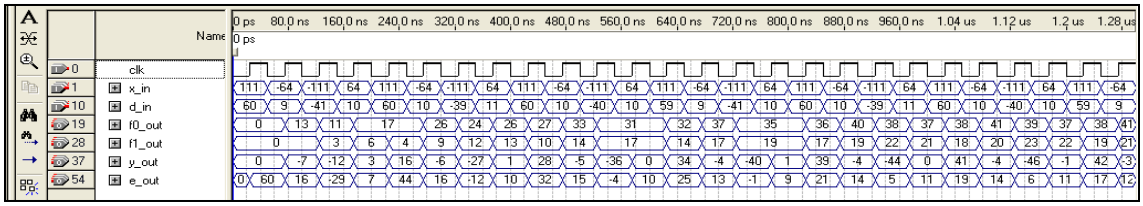


Figure 11.2

x(n)	d(n)	$f_0(n)$	$f_1(n)$	e(n)	y(n)
0	0	0.00	0.00	0.00	0.00
111	60	0.00	0.00	60.00	0.00
-64	9	13.01	0.00	15.50	-6.50
-111	-41	11.07	3.36	-29.72	-11.28
64	10	17.51	7.08	7.38	2.62
111	60	18.44	5.48	41.27	18.73
-64	10	27.38	10.64	14.47	-4.47
-111	-39	25.58	13.77	-9.94	-29.06
64	11	27.73	15.01	10.16	0.84
111	60	29.00	12.81	28.45	31.55
-64	10	35.17	16.37	13.39	-3.39
-111	-40	33.49	19.27	-1.32	-38.68
64	10	33.78	19.44	9.97	0.03
111	59	35.02	17.28	19.99	39.01
-64	9	39.36	19.77	11.53	-2.53
-111	-41	37.92	22.27	3.02	-44.02
64	10	37.26	21.90	10.36	-0.36
111	60	38.56	19.65	16.74	43.26
-64	10	42.19	21.74	12.24	-2.24
-111	-39	40.66	24.40	8.45	-47.45
64	11	38.82	23.34	11.83	-0.83
111	60	40.30	20.78	14.66	45.34
-64	10	43.48	22.61	12.13	-2.13
-111	-40	41.96	25.24	9.01	-49.01
64	10	40.01	24.11	10.91	-0.91
111	59	41.37	21.75	12.25	46.75
-64	9	44.03	23.28	10.83	-1.83

Table 11.1: Expected result

D. PIPELINE IMPLEMENTATION OF LMS ALGORITHM

By just changing the multiplier in the VHDL Program for the direct implementation with a “pipeline multiplier”, we are able to increase the speed of the register at the cost of more resources. Refer to Example 8.6 on page 511 in the textbook for details.

1. Open the file "fir6dlms.qpf" located under "VHDL/lab_11/fir6dlms" folder. Perform a functional simulation. Do you get the expected output? The output should look like the one given in Fig. 11.3.

2. Compile the project, and record the total number of logic elements, Fmax, and worst case Tco. Include these results in your report. Compare these with the ones you obtained in the previous section.

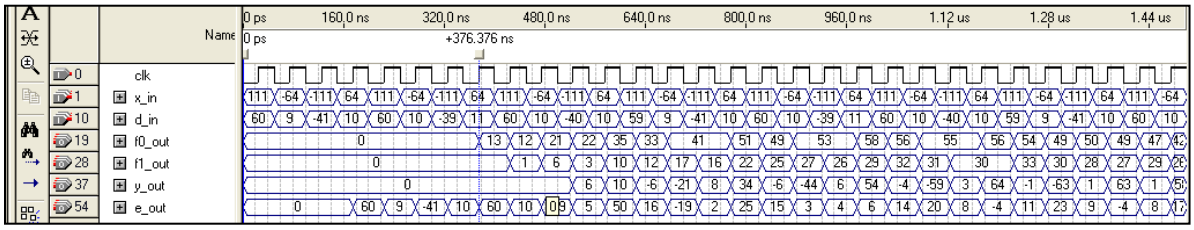


Figure 11.3

E. QUESTIONS AND REPORT

Your group report must be maximum **two pages long** including the figures **excluding the codes** in pdf format. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your report should have answers to the questions raised in the previous sections as well as answers to the following:

1. Why do you think the output named `f0_out` given in Fig 11.3 is delayed compared to the one given in Fig 11.2?

2. How do you compare the two design methods we covered in the experiment in terms of speed and resource?

3. As you noticed, the step size, μ, of the LMS algorithm is chosen to be 1/4 in this experiment. What is the effect of step size on the LMS algorithm? Why don't we pick a huge step size for faster convergence? (Hint: See pages 490 and 491 in the textbook.)

4. Make the necessary modification(s) to the "fir_lms.vhd" file so that the step size is 1/2. What difference you expect to see in the simulation? Perform a functional analysis to test your idea. Using your favorite plotting software (e.g. MATLAB) plot the `e_out` versus time for $\mu = 1/2$ and $\mu = 1/4$ on the same figure. Based on the figure, comment on the convergence time and noise floor.

Make sure that you include the question numbers to your answers.

F. REFERENCES

[1] http://en.wikipedia.org/wiki/Least_mean_squares_filter

GPU EXPERIMENT 1
GPU Computing: Lab Orientation, Software Installation, and Vector Operations

TABLE OF CONTENTS

A. Introduction
B. Software Installation for Local Operation
C. Software Installation for Remote Operation
D. Connecting to the Remote Server
E. Compiling and Running CUDA Computing SDK Examples
F. Vector addition on CPU and GPU
G. Questions and Report

A. INTRODUCTION

Before starting to this experiment, it is strongly recommended that you read the Chapters 1, 2, 3, and 4 of the textbook, "CUDA by Example" by Sanders and Kandrot. Also click on "CUDA Zone" link on the course web page and surf through the "NVIDIA CUDA Developer Zone" web site as much as you can in order to get an exposure of the state-of-the art GPU computing technology. In this experiment, we will install the software and drivers needed for the laboratory. Next, we will compile the several GPU examples provided by NVIDIA. Then, we will analyze two pre-written codes that perform vector addition, one running on CPU, and one running on GPU. We will emphasize on the difference between two codes in order to understand and appreciate how the parallelization is realized on GPU implementation. **Please read FAQ at the end of the experiment in the manual if you face any issues**.

Note that GPU and its memory come in one package, and it is introduced into a system over PCI I/O interface. In other words, we have a PCI extension card installed on the motherboard of the system. It is operated by the operating system (OS) through its driver software. We write codes in C using the libraries provided by the vendor, in this case by NVIDIA, and compile them with a special compiler, also provided by the vendor of the GPU device.

Throughout the experiment, we use the terms "Host" and "Device" to describe the CPU and the system memory, and to describe GPU and its memory, respectively, as displayed in Fig. 12.1

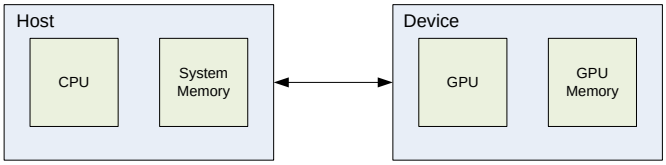


Figure 12.1: Block diagram of a system equipped with GPU device.

B. SOFTWARE INSTALLATION FOR LOCAL OPERATION

If your computer is equipped with an NVIDIA enabled graphics chip, you can go to CUDA Zone at NVIDIA website, and download the required drivers, toolkits, and SDKs, and read the corresponding help files to create a CUDA development environment on your local computer. You can find a link to CUDA Zone at the course web page.

If you do not have an NVIDA brand graphics chip in your computer or the chip you have is not CUDA enabled, or if you want to jump-start your journey in GPU computing, you can access and use the server available in High Performance DSP and Network Computing Research (HPDNCR) laboratory at NJIT. The server has an NVIDIA GeForce GTX 580 GPU device installed on it. The details of GeForce GTX 580 are found in [1]. Ubuntu Linux version 10.04.3 LTS is installed on the server as its O/S. In the next section, we explain how to install the necessary software in order to access the server remotely.

C. SOFTWARE INSTALLATION FOR REMOTE OPERATION

In this manual, it is assumed that you are using Microsoft Windows XP, Vista, 7, 10, or 11. If you are using a different O/S, then, you may skip this section. You may consult the instructor regarding the compatible software for the O/S you are using.

First, note that software we are going to use is an SSH client. We use it to create a secure connection to the server. If you already have an SSH client, you may skip this step. Go to course web page and right click on “Putty” link and choose “Save target as.” Save the executable named “putty.exe” into a convenient location on your computer. Putty does not require an installation and you can run it directly by double-clicking on it.

Second software that we use is called Xming. It is the leading X Window Server software for Microsoft Windows. We will use it to render graphical user interfaces (GUI) on our computer that are originally located on the server. Go to course web page and click on “Xming” link. When the download is completed, double click on the file to start the installation. The installation is straightforward. When the installation is completed, launch the Xming by double clicking on its icon. You should see its icon next to clock as shown in Fig. 12.2.

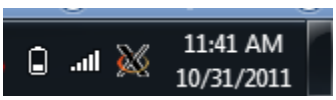


Figure 12.2: When Xming is running, its icon is displayed next to system clock.

D. CONNECTING TO THE REMOTE SERVER

Now we have all the tools required to create an SSL connection to the server. The block diagram of the setup is displayed in Fig. 12.3. Here, the terminal is your computer. You are remotely connecting to the server via SSL over TCP/IP protocol. Conveniently, the software we installed in the previous section will do the job for us.

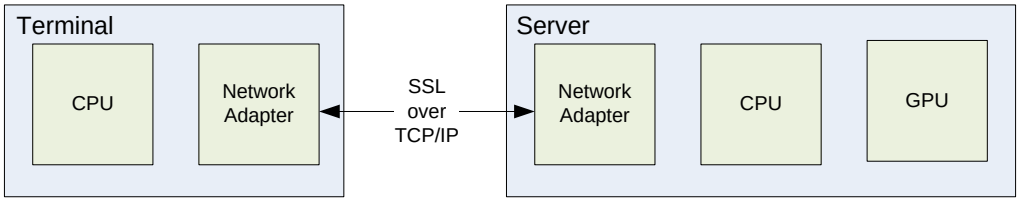


Figure 12.3: Block diagram of the setup used in the experiments with remote access to GPU server.

1. Make sure that you are on NJIT network. If you are currently at NJIT premises or connected to the Internet, you may skip this step. You can use "Cisco VPN Client" provided by NJIT in order to access NJIT Network remotely. Refer to [2] and [3] for more information on VPN Connections.

2. Double click on Putty icon. You should see a window as shown in Fig. 12.4.

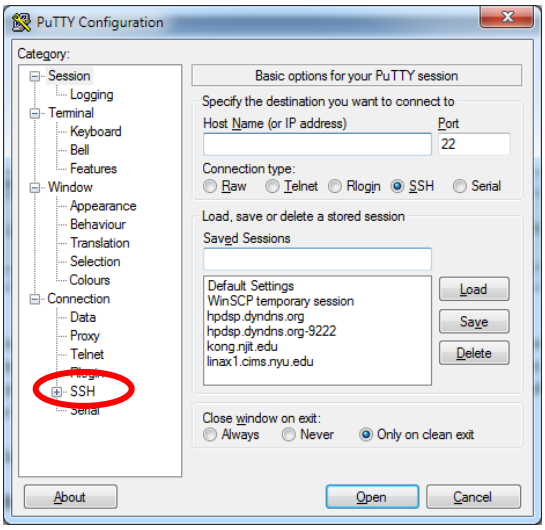


Figure 12.4: Putty's main screen.

3. Type "ECE641.resource.campus.njit.edu" into "Host Name (or IP address)" field (without quotation marks!). Change the "Port" field from 22 to 10022. Then, click on the small plus [+] sign next to "SSH" under the "Category" section. Click on "X11" in the list and click on "Enable X11 forwarding" on the right. See Fig. 12.5.

4. Go back to "Session" category by clicking on the top item, "Session," in the category list. Type "ECE641 SSL Session" into "Saved Sessions" field and click on "Save" button. "ECE641 SSL Connection" will be included in the list. Next time when you open Putty,

you can choose this item from the list and click on “Load” button. This way, you will not need to reconfigure the session each time.

5. Click on “Open” button at the bottom. You should see a black screen asking for login name. Type the login name and hit enter. Then, type the password and hit enter again. Note that the password you type will not be displayed on the screen. No special characters like ‘*’ will be printed as well. Obtain the username and password information from the instructor. Once you are connected to the server, you should a screen similar to the one displayed in Fig. 12.6.

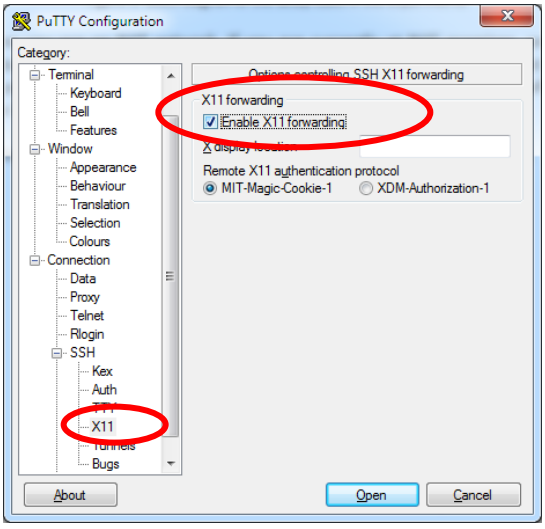


Figure 12.5: X11 Tunneling in Putty.

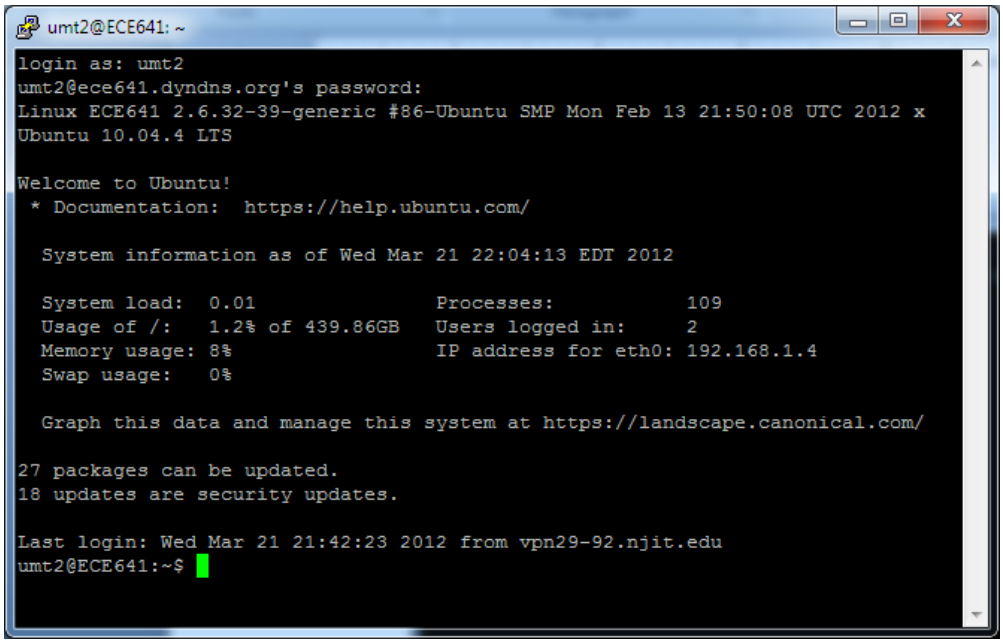


Figure 12.6: Successful login to the server via Putty software.

6. Congratulations! Now you are connected to the server. Type “passwd” and reset your password to a more secure one. When you login next time, don’t forget to use your new password. You can start typing commands and they will be executed on the server. If you are unfamiliar with UNIX commands, click on “Learn UNIX in 10 Minutes” link at course webpage for a quick tutorial. In order to give Xming software a shot, type “firefox &” (without the quotation marks) and press enter. Wait for a while. A Firefox browser window should be opened shortly. Notice that, the Firefox software is running **on the server** although its GUI is rendered **on your computer**. You can close Firefox whenever you like. Why do you think we need an “&” at the end?

E. COMPILING AND RUNNING CUDA COMPUTING SDK EXAMPLES

In this section, we will compile and run pre-coded CUDA coding examples provided by the GPU vendor NVIDIA. We will also learn how to use a text editor to edit “.cu” files.

1. Connect to the server via Putty software.

2. Type “gedit .bashrc” and press enter. Wait for the GUI to appear. Go to the end of the file, and add the following three lines to the end.

```
# CUDA PATH AND LIBRARY
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Save the file, and close “gedit” software. Then, exit “Putty” software. Launch it again, and connect to the server. This step ensures that the command prompt specific to your account is aware of the CUDA installation on the server. You only need to do this step once. Now your account is ready to develop CUDA applications!

3. Now type “cp ../../student/gpucomputingsdk_4.1.28_linux.run .” and press enter (Note that there is “space” and there is a “dot character” at the end after “.run”). This will copy the setup file for the GPU Computing Software Development Kit into your own directory. Then, type “./gpucomputingsdk_4.1.28_linux.run” and press enter. This will start the installation.

When you see the question “Enter install path (default ~/NVIDIA_GPU_Computing_SDK):” press enter.

When you see the question “Enter CUDA install path (default /usr/local/cuda):” don’t type anything, just press enter. When the installation is finished you should see the “Installation Complete” message. Type “ls” and press enter. You should see something like the following:

```
umt2@ECE641:~$ ls
```

```
gpucomputingsdk_4.1.28_linux.run  NVIDIA_GPU_Computing_SDK
umt2@ECE641:~$
```

4. Great! The installation of the SDK to your folder is completed. Change the directory to "NVIDIA_GPU_Computing_SDK" by typing "cd NVIDIA_GPU_Computing_SDK" and pressing enter. Type "ls" command again to see the files in this folder.

5. Type "make" and press enter to run the make process. Make process reads the "MakeFile" file in the current folder and builds the source code to generate "executables". Note that this process might take a considerable time (Up to 5 minutes!). Be patient. You will see "Finished building all" message and the bash prompt will wait for your next command.

6. When the compilation is finished, type "cd C/bin/linux/release/" and press enter to change the directory to the folder where the compiled executable examples are located. Type "ls" and press enter to see a list of available executables.

7. Type "./deviceQuery" to run the executable named "deviceQuery." A list of all available GPU's installed on the server and their properties will be listed as follows.

```
umt2@ECE641:~/NVIDIA_GPU_Computing_SDK/C/bin/linux/release$
./deviceQuery
[deviceQuery] starting...

./deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Found 1 CUDA Capable device(s)

Device 0: "GeForce GTX 580"
  CUDA Driver Version / Runtime Version          4.1 / 4.1
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 1536 MBytes
  (16) Multiprocessors x (32) CUDA Cores/MP:     512 CUDA Cores
...
```

Analyze the list. How many GPUs are there installed on the system? What is the "memory clock rate of the GPU installed on the system? What are its "maximum sizes of each dimension of a block"? Include your answers to these questions in the lab report.

8. Let's take a look at the source code of this executable. Type "cd ../../../src/" and press enter. Type "ls" and hit enter to see the list of available files and folders.

9. Type "cd deviceQuery" hit enter, and then type "nano deviceQuery.cpp" and hit enter to open the source file "deviceQuery.cpp" by using the nano editor. If you are not happy

with a text-based editor, you can exit "nano" by pressing CTRL+X. Type "gedit deviceQuery.cpp" and hit enter to view/edit the source code in gedit GUI. After analyzing the code, you can close it.

10. The code we analyzed in the previous step is provided by the vendor and it is not intuitive to understand easily. Therefore, we will go through and analyze the code provided in the textbook. Read section 3.3 of the textbook on pages 27 - 33. Type "cd" to return to your home folder. Then, type "cp -r ../../student/cuda_by_example/ ." to copy the examples of the textbook into your home directory. Type "cd cuda_by_example/chapter03" and hit enter, then type "nano enum_gpu.cu" and hit enter. Make sure the code is the same as the one given on page 32 of the textbook. Press CTRL+X to exit "nano". You can also use "gedit" for this step.

11. Now, we will compile the source code using NVIDIA CUDA compiler driver (nvcc.). In order to compile the source code, type "nvcc -o enum_gpu_compiled enum_gpu.cu" and press enter. nvcc will read the source code "enum_gpu.cu," compile it, and name the compiled executable as "enum_gpu_compiled." Type "ls" and hit enter to make sure the file is compiled. Now, let's run the compiled code. Type "./enum_gpu_compiled" and press enter. You should see the following.

```
umt2@ECE641:~/cuda_by_example/chapter03$ ./enum_gpu_compiled
   --- General Information for device 0 ---
Name:  GeForce GTX 580
Compute capability:  2.0
Clock rate:  1544000
Device copy overlap:  Enabled
...
```

Does this information match with the one given in Step 7? Include your answer in your lab report.

F. VECTOR ADDITION OPERATION ON CPU AND GPU

Read Section 4.2.1 in the textbook before you start this part of the experiment. Let $\mathbf{a}$ and $\mathbf{b}$ be two vectors of size $N \times 1$, and let a_i and b_i be the elements of $\mathbf{a}$ and $\mathbf{b}$ vectors at the i*th* row, respectively. Then, the element of the resulting summation vector $\mathbf{c}$ at the i*th* row is calculated as

$$c_i = a_i + b_i \tag{12.1}$$

Now, we implement this mathematical operation both on CPU and GPU as follows.

1. Type "cd ~/cuda_by_example/chapter04" and hit enter. Type "nano add_loop_cpu.cu" and analyze the source code. Press CTRL+X to exit "nano".

2. Compile the source code using nvcc, name the compiled binary "add_loop_cpu_compiled," and run the compiled binary. See explanations in the previous sections if you are unclear how to follow these instructions.

3. Repeat steps 1 and 2 for the GPU implementation. The source code is in the file "add_loop_gpu.cu."

G. QUESTIONS AND LAB REPORT

Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

Your lab report must have the answers to the questions raised in the previous sections as well as the answers to the following questions.

1. What are the UNIX commands for the following operations? Give an example for each one.
    a. Changing a directory
    b. Listing the files in a folder
    c. Running an executable file
    d. Copy a file to a folder
2. What is the version of the NVIDIA CUDA compiler driver installed on the server? Explain how you obtained that information.
3. How do we get the ID of a block in CUDA C?
4. Modify the GPU vector addition source code such that in addition to its current operation, it also multiplies the second vector with 2 and subtracts it from the first vector as expressed

$$d_i = a_i - 2b_i \tag{12.2}$$

    Provide the modifications to the code you made in the lab report. Avoid including the entire source code in the report. Just focus on the modifications.
5. **Bonus Point**: Modify the parallel vector addition code such that it adds two matrices of sizes *N-by-M*.

H. REFERENCES

[1] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580
[2] http://ist.njit.edu/vpn/
[3] http://en.wikipedia.org/wiki/Virtual_private_network

I. FAQ FOR GPU EXPERIMENT 1

1. Can I connect to the server from outside the NJIT for GPU experiments?

Answer: Yes, you can. First, you need to connect to NJIT's network using VPN client software. Then, you will be able to connect to our servers for your experiments. You can find all kinds of information such as VPN setup and connecting to NJIT's network from following link: http://ist.njit.edu/vpn/

2. I followed the steps as instructed in the manual for compiling and running the CUDA Computing SDK but when I try to run the executable "deviceQuery" using "./deviceQuery" or compile file "enum_gpu.cu" using "nvcc -o enum_gpu_compiled enum_gpu.cu", I get a library link error. Why?

Answer: Even though you installed the CUDA Computing SDK successfully, you might get the library link error. The reason is usually due to wrong environment variables that you set in second step of Section E. Please redo that step and make sure that you set the environment variables correctly.

GPU EXPERIMENT 2
Memory, Blocks, Threads, Synchronization, and Matrix Multiplication

TABLE OF CONTENTS

A. Introduction
B. Memory Allocation and Copying Data between Host and Device Memories
C. Matrix Multiplication on a Computer
D. Blocks, Threads, Shared Memory, and Thread Synchronization
E. Questions and Report

A. INTRODUCTION

Before starting to this experiment, it is highly recommended that you read the Chapter 5 of the textbook and understand the examples provided in that chapter.

In this experiment we first focus on the memory allocation mechanism of the GPU. We return to the example given in the previous experiment to get an insight on it. We also revisit the concept of moving data between the host and the device. Then, we revisit the matrix multiplication problem and implement the multiplication algorithm on CUDA C. We provide an analysis of the source code in order to highlight the concepts such as blocks, threads, shared memory and thread synchronization.

Students are advised to follow and understand each step. The questions to be answered in the report are provided between the steps. **Please read FAQ at the end of the experiment in the manual if you face any issues.**

B. MEMORY ALLOCATION AND COPYING DATA BETWEEN HOST AND DEVICE MEMORIES

In this section, we review how the memory allocation both in host and device is done and how data is transferred between the host and device memories, i.e. the system memory and GPU memory. As you know, in standard C, memory allocation is done via malloc() function. The following code will allocate a memory space on the system to store 10 numbers of type float:

```
int N = 10;
float *a;
a = (float*)malloc(N*sizeof(float));
```

In order to do the same allocation on GPU memory we use cudaMalloc() function as given

```
int N = 10;
float *dev_a;
cudaMalloc((void**)&dev_a, N*sizeof(float));
```

Let's fill in the values of array a in system memory with a ramp function with the following code:

```
for (int i=0; i<N; i++) {
    a[i] = i;
}
```

Now our task is to copy the contents of a into the dev_a, i.e. contents located in the system memory into the GPU memory. We use cudaMemcpy() function to accomplish this as given

```
cudaMemcpy(dev_a, a, N*sizeof(float), cudaMemcpyHostToDevice);
```

Q1: What would be the code to copy the contents of dev_a into a, i.e. contents located in the GPU memory into the system memory? Provide the answer in your report.

Now we will go back to the vector addition code in GPU Experiment 1 and modify it to get a better understanding on how GPU memory allocation works:

1. Connect to the server.

2. Locate the "cuda_by_example" folder and change directory to "chapter04" folder.

3. Open the "add_loop_gpu.cu" source code in your favorite editor. As you remember from the previous experiment, this code adds two vectors of size 10 by 1 together and writes the result to another 10 by 1 vector. Compile the code and run it at least once. Refer to the previous experiment if you need help with this step.

4. Reopen the source code and edit the 24th line as follows:

Original:

```
c[tid] = a[tid] + b[tid];
```

Modified:

```
c[tid] = a[tid] - b[tid];
```

5. Edit the 48th line as follows:

Original:

```
add<<<N,1>>>( dev_a, dev_b, dev_c );
```

Modified:

```
add<<<5,1>>>( dev_a, dev_b, dev_c );
```

6. Edit the 56th line as follows:

Original:

```
printf( "%d + %d = %d\n", a[i], b[i], c[i] );
```

Modified:

```
printf( "%d - %d = %d\n", a[i], b[i], c[i] );
```

7. Compile the code and observe the result.

Q2: Include output into your report. Explain what we do with the above modifications, what the expected outcome is. More importantly, focus on the values of the elements of c with indexes larger than five. What is your opinion on the memory allocation mechanism of the GPU?

C. MATRIX MULTIPLICATION ON A COMPUTER

In this section, our goal is to multiply two square matrices and write the result into another matrix. Let **A** and **B** be the input matrices of size *N* by *N* and **C** be the output matrix of size *N* by *N*. Matrix multiplication is given as

$$\mathbf{C} = \mathbf{AB} \tag{13.1}$$

Assuming *N* = 3 we have

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \tag{13.2}$$

where c_{ij} is the element of the matrix **C** located at the i*th* row and j*th* column. From linear algebra we know that

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \tag{13.3}$$

Similarly we have

$$c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \tag{13.4}$$

and so on. In general we have

$$c_{ij} = \sum_{k=1}^{N} a_{ki} b_{jk} \tag{13.5}$$

where *k* is the dummy variable used for the summation. In order to store a matrix in a computer memory, one can use special structures that has inherent two dimensional indexing (as in a two dimensional array in C). However, in order to have a full understanding of what we are doing, and for the sake of ease of memory allocation, we convert the two dimensional matrices into one dimensional arrays as given

$$\begin{aligned} \overline{\mathbf{a}} &= \begin{bmatrix} \overline{a}_0 & \overline{a}_1 & \cdots & \overline{a}_{N^2-1} \end{bmatrix} \\ \overline{\mathbf{b}} &= \begin{bmatrix} \overline{b}_0 & \overline{b}_1 & \cdots & \overline{b}_{N^2-1} \end{bmatrix} \\ \overline{\mathbf{c}} &= \begin{bmatrix} \overline{c}_0 & \overline{c}_1 & \cdots & \overline{c}_{N^2-1} \end{bmatrix} \end{aligned} \tag{13.6}$$

We also define a special indexing as given

$$\begin{aligned} x &= (i-1)N + (k-1) \\ y &= (k-1)N + (j-1) \\ z &= (i-1)N + (j-1) \end{aligned} \tag{13.7}$$

where *i* and *j* are the row and column indexes of the original matrix, respectively, *k* is the dummy variable of (13.5), and *N* is the number of rows (as well as number of columns since it is a square matrix) of the original matrix. Note that since $1 \le i, j, k \le N$ and we have $0 \le x, y, z \le N^2 - 1$. With the new indexing we can write

$$c_{ij} = \overline{c}_z = \sum_{k=1}^{N} \overline{a}_x \overline{b}_y \tag{13.8}$$

Note that in (13.6) and (13.7) new indexes run from 0 to *N* - 1, not from 1 to *N*. This is due to the fact that in C language (and in CUDA C language) the indexing of arrays start from 0, not 1.

Q3: Give a numeric example to **A** and **B** for *N* = 3 and analytically calculate the element of **C** located at *i* = 1 and *j* = 1 using (13.5). Then, write the arrays given in (13.6) and recalculate the same element using (13.8). Show that both results are same. Repeat for *i* = 2 and *j* = 3.

D. BLOCKS, THREADS, SHARED MEMORY, AND THREAD SYNCHRONIZATION

In this section we implement the matrix multiplication introduced in the previous section in CUDA C. Specifically, we create a *kernel* in the GPU with a *grid* of *N* x *N blocks*.

Each block calculates an element of the matrix **C**. Each block in the grid has *N threads* and each thread multiplies an element of matrix **A** with an element of matrix **B**. The first thread in each block sums the multiplication done by itself and by the other threads in the same block. We use *shared memory* in the blocks to store the multiplication results, and use *thread synchronization* to make sure every thread finishes the multiplication before the first thread performs the summation. You can find the corresponding source code under /home/student/mtx_mul in mtx_mul.cu file. Copy this file into your directory before working on it. Open the source file in your favorite text editor and analyze it. Let's go step by step starting with the main() function:

1. First we define the matrices **A**, **B**, and **C** on both host and device and allocate a memory space to them:

```
float *a, *b, *c;
float *dev_a, *dev_b, *dev_c;

// allocate memory on the CPU side
a = (float*)malloc(N*N*sizeof(float));
b = (float*)malloc(N*N*sizeof(float));
c = (float*)malloc(N*N*sizeof(float));

// allocate the memory on the GPU
cudaMalloc((void**)&dev_a,N*N*sizeof(float));
cudaMalloc((void**)&dev_b,N*N*sizeof(float));
cudaMalloc((void**)&dev_c,N*N*sizeof(float));
```

2. We fill the matrices A and B on host with some values and copy them to their counterparts on the device:

```
// fill in the host memory with data
for (int ctr=0; ctr<N*N; ctr++) {
    a[ctr] = ctr+1;
    b[ctr] = -ctr-1;
}

// copy the arrays 'a' and 'b' to the GPU
cudaMemcpy(dev_a, a, N*N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b, N*N*sizeof(float), cudaMemcpyHostToDevice);
```

3. Then, we create a kernel on the host with a grid which has *N* x *N* blocks where each block has *N* threads:

```
dim3 blockDim(N, N);
multiply<<<blockDim,N>>>( dev_a, dev_b, dev_c );
```

Note that, we create a structure of type dim3 for the block. The kernel runs the global function multiply(). See Fig. 13.1 for an example grid for *N* = 3. We have a block of size 3

by 3 and each block has 3 threads. The *x* and *y* dimensions of the blocks and *x* dimension of the threads can be easily observed in the figure.

Q4: What is the limitation of CUDA C in terms of dimensions? What is the maximum number of dimensions for a grid to create blocks? What is the maximum number of dimensions for a block to create threads? Is there a limitation on total number of blocks or total number of threads that can be created on a GPU? Is this brand and model dependent? If so, what are these limitations in GeForce GTX 580 (The GPU installed on the server)?

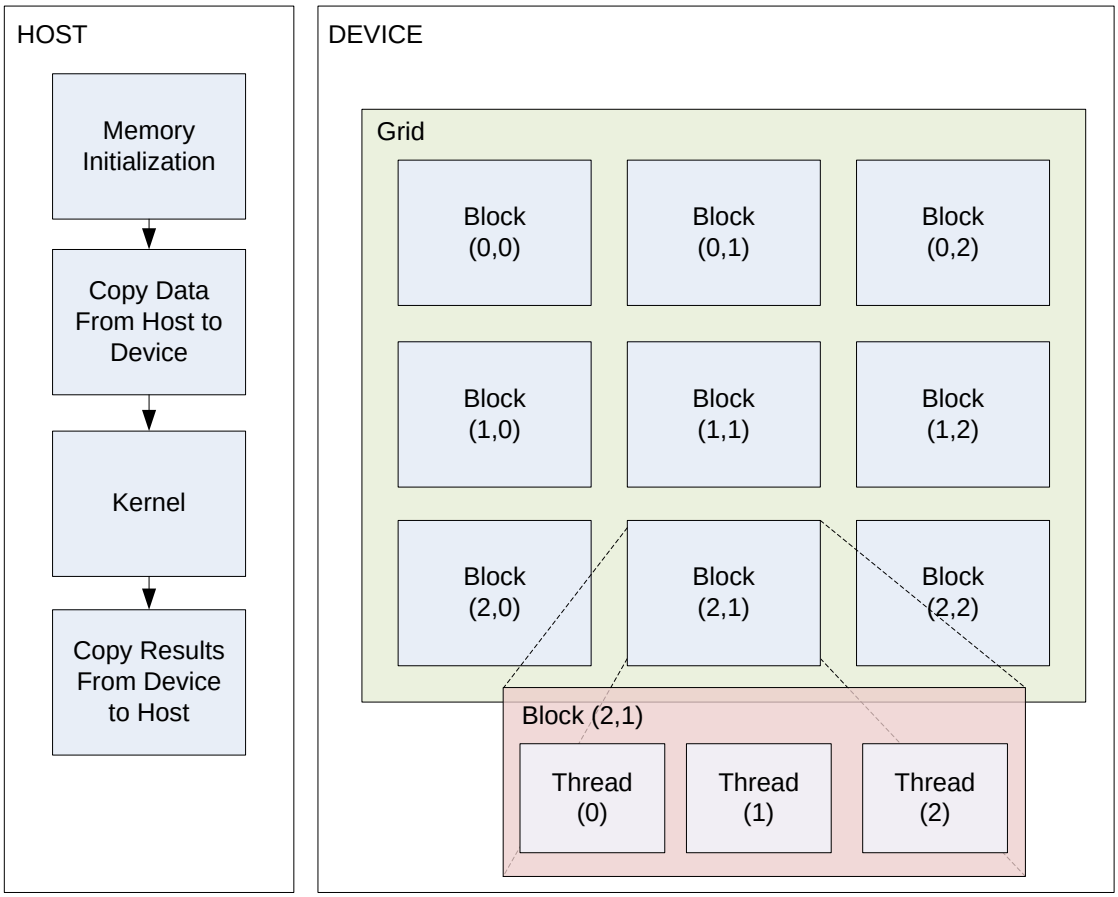


Figure 13.1: An example grid with 3 x 3 blocks. Each block has 3 threads.

4. Finally we print the results on the screen and free the memory both on host and device:

```
// print matrices A, B, and C on screen
printf("A=\n");
print_mtx(a);

printf("B=\n");
print_mtx(b);
```

```
printf("C=\n");
print_mtx(c);

// free memory on the GPU side
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);

// free memory on the CPU side
free(a);
free(b);
free(c);
```

Now let's focus on the multiply() kernel:

1. We start with defining an array of size *N* called cache. Note that this is a shared array, i.e. it is visible to all the threads running on a block. This is achieved by using "__shared__" keyword:

```
// define cache in the shared memory
__shared__ float cache[N];
```

2. Then we set the indexes *i*, *j*, *k*, *x*, *y*, and *z* according to (13.7). Note that with this setup the block that is located at the i*th* row and j*th* column is associated with the element of matrix **C** located at the (i+1)*th* row and (j+1)*th* column. See previous section for further details on this indexing.

```
// get the x and y indexes of the block and x index of the thread
int i = blockIdx.x;
int j = blockIdx.y;
int k = threadIdx.x;

// calculate the indexes from one dimensional arrays
// we need this calculations for matrix operations
int x = i * N + k;
int y = k * N + j;
int z = i * N + j;
```

3. Next we implement (13.8):

```
// set the cache values
cache[k] = a[x] * b[y];
```

4. Once we have all the threads finish the multiplication given in step 3, we have the first thread in that block (with threadIdx.x = 0) sum all the values in the cache. Note that no thread can move further until every thread in the block reaches to the __syncthreads(); function. This way, we are able to synchronize all the threads:

```
// synchronize threads in this block
__syncthreads();

 // first thread sums the values in the cache
 // and writes the result to c array
if (k == 0){
      int ctr = 0;
      float temp = 0;
      while (ctr<N){
            temp += cache[ctr++];
      }
      c[z] = temp;
}
```

Q5: Compile the source code and run it:

a) Check at least two elements of the matrix **C** to see if the results match with the theory. Provide the results in your report.

b) Change the matrix size to *N* = 8, *N* = 24, *N* = 1,024, and *N* = 100,000 (You may want to comment out the printing in the main() function at this point). Recompile it every time and run the code. Does the code compile for every case? If not, why?

Bonus: Modify the code so that you don't need to recompile the code every time you change the value of *N*. Provide the modifications on your report.

Bonus: Modify the code so that it can handle any value for *N*. No matter how large the *N* is, your code should be able to execute. Refer to Chapter 5 of the textbook.

E. BONUS QUESTIONS AND REPORT

Make sure you provide answers in your report to Q1, Q2, Q3, Q4, and Q5 given in the previous sections. There is no page limitation. Students are encouraged to compile and run the example source codes provided in the Chapter 5 of the textbook. These codes can be found under cuda_by_example/chapter05 folder. Students are also encouraged to challenge themselves in solving the following bonus questions. Bonus questions are due in two weeks. Do not prepare a cover page. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

1. Remove the printing function at the end. Write a CPU implementation of the multiplication function and compare the GPU results with CPU results to see if they match.

2. Modify the source code so that it can multiply two rectangular matrices. If the size of the matrices **A** and **B** are N by M then the code should be able to provide the matrix **C** of size N by N.

3. Further modify the code to handle both **C** = **AB** and **C** = **BA** multiplications for the rectangular matrices **A** and **B**. Note that the resultant matrix is of size M by M in the latter.

4. Apply the reduction technique provided in Section 5.2 of the textbook in your code. See Fig. 5.4 of the textbook.

Provide the source code and output for the bonus questions you answered.

F. FAQ FOR GPU EXPERIMENT 2

1. mtx_mul.cu file is read only. I cannot modify it using gedit.

Answer: Type the following command to add read, write, and execute permissions for the file.

chmod +rwx mtx_mul.cu

GPU EXPERIMENT 3
Discrete Cosine Transform on GPU

TABLE OF CONTENTS

A. Introduction
B. Discrete Cosine Transform
C. Preparing the Quantized DCT Matrix and Data
D. Discrete Cosine Transform on GPU
E. Time Measurement with CUDA Events
F. Questions and Report

A. INTRODUCTION

In this experiment we apply our knowledge in CUDA C into discrete cosine transform (DCT). We begin with introducing the DCT which is a common tool in image and video processing applications. Then we use a MATLAB code to prepare data for our CUDA C code. We discuss the details of the algorithm in the next section. Finally, we introduce the CUDA events as a way to measure the processing speed of our GPU codes.

B. DISCRETE COSINE TRANSFORM

Discrete cosine transform (DCT) of an *N* by *N* matrix $\mathbf{B}$ is defined as

$$\mathbf{\Phi} = \mathbf{A}_{DCT}^{T}\mathbf{B}\mathbf{A}_{DCT} \tag{14.1}$$

where

$$\mathbf{A}_{DCT} = \left[A(k,n)\right] = \frac{1}{c_k}\cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{14.2}$$

where $0 \le k, n \le N-1$ and

$$c_k = \begin{cases} \sqrt{N} & k = 0 \\ \sqrt{N/2} & k \neq 0 \end{cases} \tag{14.3}$$

However, since the absolute value of the elements of the $\mathbf{A}_{DCT}$ are less than 1, we scale-up its elements as given

$$\mathbf{A}_{DCT}^{Q} = Q\mathbf{A}_{DCT} \tag{14.4}$$

where Q is the scalar quantization coefficient. We modify (14.1) as

$$\mathbf{\Phi} = \left[\mathbf{A}_{DCT}^{Q} \right]^{T} \mathbf{B} \mathbf{A}_{DCT}^{Q} / Q^{2} \tag{14.5}$$

which is equivalent to (14.1).

C. PREPARING THE QUANTIZED DCT MATRIX AND DATA

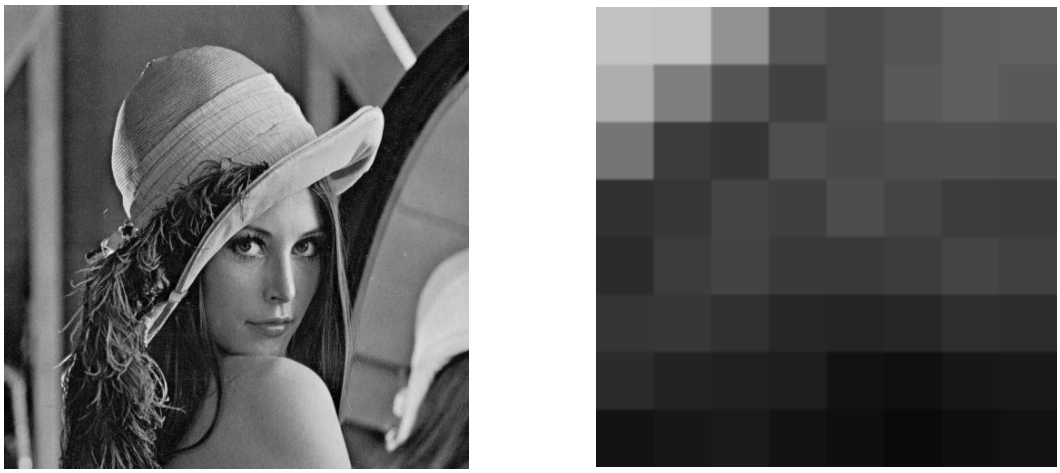
1. Download "dct.zip" from the course web page and unzip it into course materials folder. Open the file "generate_dct_matrix.m" in MATLAB and analyze the code.

Q1: Note that the MATLAB code has five sections. Explain clearly the operations done in each section of the code.

2. Run the MATLAB code. Observe that it generates three text files, i.e. "dct.txt," "dct_transpose.txt," and "sub_image.txt" which contain the row-major coefficients of the transform matrix, transpose of the transform matrix, and the *N* by *N* sub-image, respectively, in each line.

Note that the code reads an image file named "lena.jpg" and it only transfers a middle block of it of size *N* by *N* into the "sub_image.txt" file. The image file and the magnified version of the sub-image are given in Fig. 14.1. Note that these 8-bit coded gray colors are represented with integer numbers. The numbers 0 and 255 represent black and white, respectively, and everything in between are the tones of gray. The number representation of the sub-image is given in Table 14.1. Make sure that these numbers confirms with the elements in the text file "sub_image.txt." Also note that the MATLAB code not only prepares the data but it also provides the expected output. You should see the following in the "Command Window" of MATLAB when you run the code:

```
Therotical Result
  729.2368   -9.6953  165.2654  -23.4958   91.6099   19.7154   60.8633   40.4326
  122.4807   78.3133    1.1055   -5.7550   -2.9000    1.8442   -2.9100    6.6953
  133.2608    8.9482  -41.8019  -48.1487   -9.0634  -15.6713   -7.4661   -5.9354
   29.0202  -41.3986  -52.9816  -20.0031   -5.7922    2.4324    0.6491    0.1461
   56.8547   -8.3817  -16.7897   24.9657   16.5809    5.7814    0.9052    9.9045
   42.0635   -2.0454    1.1394   17.0512   14.6140   -5.8286   -7.2513    5.4120
   56.3506   -7.3933    1.3281    9.6004   16.8015   -4.1217   -4.1891    5.4726
   47.2290   -5.3825   -1.9240    6.9931    2.6257    3.7168    3.2242    3.6918
----------------------
Expected Result
  729.7594   -9.7681  165.8096  -22.9302   91.7414   19.6016   60.8387   40.6005
  122.5216   78.3225    1.1111   -5.7408   -2.9162    1.8521   -2.9313    6.7630
  133.7952    8.8898  -41.6311  -48.1275   -9.0031  -15.7098   -7.4709   -5.8915
   29.5455  -41.4865  -52.9255  -20.0624   -5.7285    2.3983    0.6661    0.1436
   56.9864   -8.4232  -16.7610   25.0280   16.6359    5.7664    0.9036    9.9470
   42.0403   -1.9938    1.1102   17.0876   14.6067   -5.8541   -7.2821    5.4325
   56.3424   -7.3781    1.3195    9.6756   16.8222   -4.1507   -4.2184    5.4969
   47.5055   -5.3705   -1.8827    7.0076    2.6534    3.7170    3.2346    3.7201
```



(a) (b)

Figure 14.1: (a) Lena image, (b) 8 by 8 block in the middle of the Lena image we use as the data source in the experiment.

186	194	195	178	156	96	82	78
183	193	190	144	87	77	85	94
198	172	126	86	67	77	89	95
171	116	62	56	79	74	77	78
98	52	57	70	65	78	70	62
50	47	62	69	60	59	62	70
45	56	58	51	42	41	42	50
46	47	40	37	35	25	23	29

Table 14.1: Number representation of the sub-image given in Fig 14.1.b

Q2: Pick three elements of the DCT matrix, calculate its elements using (14.2), multiply them by quantization factor, Q, and compare them with their corresponding entries in "dct.txt" and "dct_transpose.txt." Show that they are equal (with a reasonable quantization error!) to each other.

Q3: Why the elements in Q2 are not exactly same? Are we doing just multiplication, or are we doing something more?

3. Login to server. Create a folder named "dct" by typing "mkdir dct" and hitting enter. Change directory to this folder.

4. Copy the file "dct_transform.cu" into the current folder. The file is located at "/home/student/dct/"dct_transform.cu"

5. Make sure the XMing server is running. Type "gedit dct.txt &" and hit enter to open an empty text file. Copy the contents of "dct.txt" on your local computer into this remote text file and save it. Repeat this step for "dct_transpose.txt" and "sub_image.txt" files. If you would like to use a secure file transfer software such as WinSCP you can do so.

D. DISCRETE COSINE TRANSFORM ON GPU

1. Open the "dct_transform.cu" file in your favorite editor and analyze the code. Note that the code is a modified version of "mtx_mul.cu" we studied in the previous experiment. Some highlights of the code include:

a) Pointers to `dev_at` and `dev_x` arrays are introduced. We use the former to store the transpose of the DCT matrix and latter to store the interim matrix $\mathbf{X}$ as given

$$\begin{aligned} \mathbf{X} &= \left[\mathbf{A}_{DCT}^{Q} \right]^{T} \mathbf{B} \\ \mathbf{\Phi} &= \mathbf{X}\mathbf{A}_{DCT}^{Q} \end{aligned} \tag{14.6}$$

Note that we require an interim matrix since the original matrix multiplication algorithm only supports multiplication of *two* matrices.

Q4: Which variables do represent the matrices $\mathbf{B}$ and $\mathbf{\Phi}$ in the code? Provide your reasoning with your answer.

b) Note that the operations given in (14.6) are performed by calling two kernels on lines 90 and 91 as given

```
multiply<<<blockDim,N>>>( dev_at, dev_b, dev_x );
multiply<<<blockDim,N>>>( dev_x, dev_a, dev_c );
```

Since the codes runs sequentially on host, second kernel is not run before the first kernel is finished. This method is called as *host synchronization*.

c) In the kernel instead of assigning the sum to c[z] element, we first divide the sum by Q. See line 53 in the code.

Q5: Show that this is equivalent to (14.5).

d) The text files are read using the C++ libraries with read_file() function as given

```
void read_file(float *m, char* file_name, int N){
 string line;
 ifstream textfile (file_name);
 if (textfile.is_open()) {
      int ctr=0;
      while (textfile.good() && ctr<N*N) {
           getline (textfile,line);
           m[ctr++] = atoi(line.c_str());
      }
```

```
    textfile.close();
} else {
    cout << "Unable to open file " << file_name;
}
}
```

Q6: How do we make sure that we do not exceed the pre-allocated memory space for the array m of type float? What happens if the data file contains more than N^2 lines?

Q7: Compile the code and run it. Show that the results match with the expected result. How can you improve the expected result and make it closer to the theoretical result? State the modification, do it, compile the code, and show that you actually reduced the error.

E. TIME MEASUREMENT WITH CUDA EVENTS

Although CUDA events are not solely created for this purpose we can use them to measure the time it takes for our code to be run on the GPU. This practice is very useful in measuring the performance increase (or decrease) in performing operations on GPU rather than CPU. Students are referred to Section 5.3 of the textbook for a detailed discussion.

In order to measure the time it takes to perform DCT on the server add the following lines to your code:

On line 60:

```
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start, 0);
```

On line 96:

```
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
```

On line 112:

```
    float   elapsedTime;
    cudaEventElapsedTime(&elapsedTime, start, stop);
    printf("\nElapsed Time: %f ms\n", elapsedTime);

    cudaEventDestroy(start);
    cudaEventDestroy(stop);
```

Q8: Compile the code. Run it. Include the measured time in your report. Run it five more times. Report the measured times. Do they differ? What might be the reason?

E. BONUS QUESTIONS AND REPORT

Make sure you provide answers in your report to all the questions from Q1 to Q8 given in the previous sections. There is no page limitation. Students are encouraged to challenge themselves in solving the following bonus questions. Bonus questions are due in two weeks. Adding your names and student IDs to the top of the first page is enough. Depending on the preference of the instructor, submit your report via Canvas or email it to the instructor by the beginning of the next experiment.

1. Instead of just calculating the DCT transform of one 8 by 8 block in the middle of the image, divide the original 512 by 512 image into 64 x 64 = 4096 blocks of size 8 by 8 and perform DCT on all of them.

2. Challenge yourself to do everything in the previous question in one kernel call.

3. Instead of reading DCT coefficients from a file, calculate them in the GPU.

4. Write a code that calculates the inverse DCT.