

Revision: Date:

1.0 15/10/2021

Title:

**LAB MANUAL – FIR Filter on FPGA - Teacher**

**Author(s):**

**Thomas Overmeire (22255577)**

**Zicong Yan (22940392)**

**Jonathan Zhang (22941998)**

**Jonathan Louw (21712457)**

**Chang Li (22669058)**

**Bo Wang (22942279)**

**Note: This is the teacher’s version of the Lab Manual, with answers included. To convert to student version, with no answers, simply remove the contents of each action item.**

Document History:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Comments | Completed | Checked |
| 1.0 | 15/10/2021 | Document Completed | T.O. |  |
|  |  |  |  |  |

## Table of Contents

[Table of Contents 2](#_Toc85227756)

[Laboratory 1 – FIR Filter 3](#_Toc85227757)

[Introduction 3](#_Toc85227758)

[Other Materials 4](#_Toc85227759)

[Explanation of provided files 4](#_Toc85227760)

[Assessments 4](#_Toc85227761)

[System to be Developed 5](#_Toc85227762)

[Functionality 6](#_Toc85227763)

[Hardwired Logic Implementation Hierarchy 6](#_Toc85227764)

[Preparation 7](#_Toc85227765)

[Synthesis 12](#_Toc85227766)

[Analysis and Synthesis 13](#_Toc85227767)

[RTL View 14](#_Toc85227768)

[Fitter 16](#_Toc85227769)

[Assembler 17](#_Toc85227770)

[Timing Analysis 18](#_Toc85227771)

[Device Configuration 18](#_Toc85227772)

[Hardware Testing 19](#_Toc85227773)

[References 20](#_Toc85227774)

[Appendix A 21](#_Toc85227775)

[Appendix B 22](#_Toc85227776)

# Laboratory 1 – FIR Filter

## Introduction

In this laboratory, a FIR filter is implemented using a hardware description language (HDL) known as VHDL, on an Intel FPGA, the INTEL DE10\_LITE MAX 10 10M50DAF484C7G. In this lab, you will complete a few missing pieces of code, run analysis and simulations on said code, and test the code on a real FPGA alongside a laboratory wave generator and oscilloscope.

A close-up of a computer chip

Description automatically generated with low confidence

Figure 1: Labelled diagram of FPGA

A picture containing diagram

Description automatically generated

Figure 2: Exploded view of FPGA

## Other Materials

A user manual has been provided and can be found in the docs folder in the zip file of resources described in the next section. Apart from that, the manuals supplied by the manufacturers of the hardware used in this laboratory can be found here.

The manual for the DE10-Lite Intel FPGA can be found at the following reference: [1]

The manual for the InfiniiVision DSO-X 2002A oscilloscope and wave generator can be found at the following link: [2]

The manual for the PmodDA2 Digital to Analog converter (DAC) can be found at the following link: [3]

## Explanation of provided files

A zip file containing the files used in this lab is provided. Unzip this zip file to see the following directory structure:

<my\_dir>

|  |  |  |  |
| --- | --- | --- | --- |
| GitHub Repository |  | | |
|  | /doc |  | Contains documentation for the laboratory and the design. |
|  | /src |  | Contains VHDL code for the FPGA. |
|  | /synth |  | Contains files used for synthesis. |
|  | /sim |  | Contains files used to simulate the design. |

Table 1: Directory structure

You should copy and paste the contents of this directory to have an unedited copy to refer to should something go wrong during this laboratory.

You will be expected to submit this fine structure by the due date specified by your laboratory supervisor.

## Assessments

This laboratory has three major components.

1. Report (80% of final mark)  
   A Laboratory report should be completed. The marked components that will need to be completed in this report will be explained later. This will include completing some code in the /src/ folder and copy pasting changes into the report, providing screenshots of RTL views of the completed system, providing images of the oscilloscope as evidence that the system is working, and providing outputs of timing analysis done using Quartus’ timing analysis tools. The report should have a UWA coversheet attached. The activities in this manual amount to 16 marks.
2. Zip file (10% of final mark)  
   The directory structure containing the documents and code for this lab should be zipped after you have completed the tasks.
3. Interview (10% of final mark)  
   You will attend a one-on-one interview with a member of the teaching staff of this unit. The purpose of this interview will be to demonstrate understanding of the completed system, so you should ensure you understand each exercise thoroughly.

# System to be Developed

A FIR filter is implemented using a hardware description language (HDL) known as VHDL, on an Intel FPGA, the INTEL DE10\_LITE MAX 10 10M50DAF484C7G.

Buttons, switches, and a bank of seven segment displays serve as built-in user interface for the user to interface with the programmed functionality of the FPGA. In the case of this laboratory, the switches should be set to on or off to form a binary number, where the right most switch is the least significant bit. This binary number between 0 and 3 correspond to high-pass, low-pass, band-pass and band stop respectively with changes to the seven-segment display and filter coefficients being changed in near real time, the FIR filter coefficients are swapped to the coefficients at that address (the addresses being controlled via a counter).

The filter coefficients are the predetermined numbers the FIR filter convolves incoming data with. The four sets of coefficients are implementations of four different FIR filters: a high pass filter, which suppresses low frequencies, a low pass filter, which suppresses high frequencies, as well as a band pass filter, and a band stop filter, which either suppress all information inside or surrounding a band of frequencies respectively.

The FIR filter processes an incoming signal from a wave generator through ADC and applies the FIR filter to it. The FIR filter has the effect of suppressing the incoming signal if it falls within the stop band of the filter. The output is extracted through a DAC and is displayed on an oscilloscope. An example of the output is shown below.

|  |  |
| --- | --- |
|  |  |
| Figure 3: Here, the Low pass filter correctly allows a low frequency input to pass | Figure 4: Here, the low pass filter correctly reduces the amplitude of a high frequency signal. |

## Functionality

List of functionalities

1. The user can select the desired FIR filter coefficients using the filter switch button and the bank of switches. The bank of switches is used to write a binary number, which determines the
2. The ADC samples an analog input signal.
3. The FIR Filter module filters specific frequencies of the sampled signal.
4. The DAC outputs the digital filtered signal as an analog output.
5. The reset button restarts the code.

## Hardwired Logic Implementation Hierarchy

The overall hierarchy of the system is shown below. The Lab2\_top top file. The top file controls the separate modules, from left to right: the ADC, the FIR filter, the DAC, and the coefficient generator.

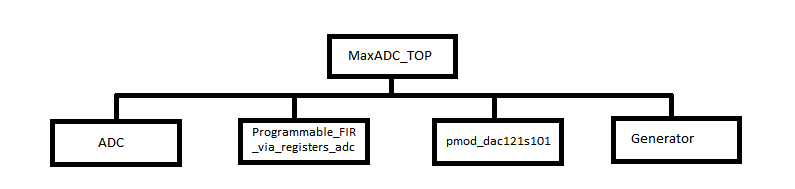


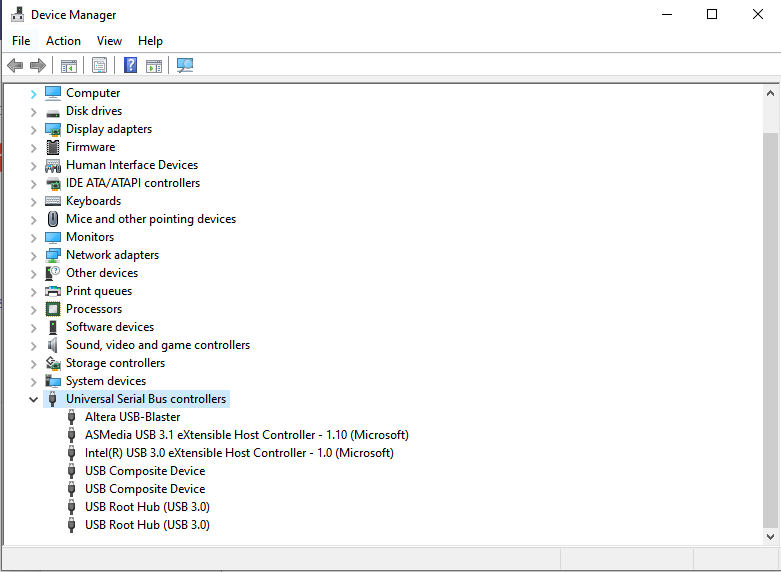
Figure 3: Design hierarchy for all modules included in the top level design

# Preparation

This section contains all actions and exercises required to complete the code and get the system ready for simulation and analysis. Questions should be copied into your report document and answered by either attaching code, a photo, a screenshot, or code output, as directed.

**USB Blaster Setup**

To ensure that your Altera USB-Blaster driver is working as intended check for the USB-Blaster in your computers device manager once the device has been connected. If connected and all relevant drivers are correctly installed, then you should see it listed in Universal Serial Bus controllers as Altera USB-Blaster.



If this is not the case check under other devices and you should see the USB-Blaster under Other devices, if you see the following you need to update the drivers of your device with the Alter USB-Blaster drivers.



To do this right click on the device and go to update driver, making sure to update driver software from your computer. From here you need to find the USB-Blaster driver which should be in your installation path for intel (example C:\intelFPGA\_lite\18.1\quartus\drivers\usb-blaster). Once this has been selected the driver should be installed and hardware recognized by the Quartus device programmer.

**Task 1:**

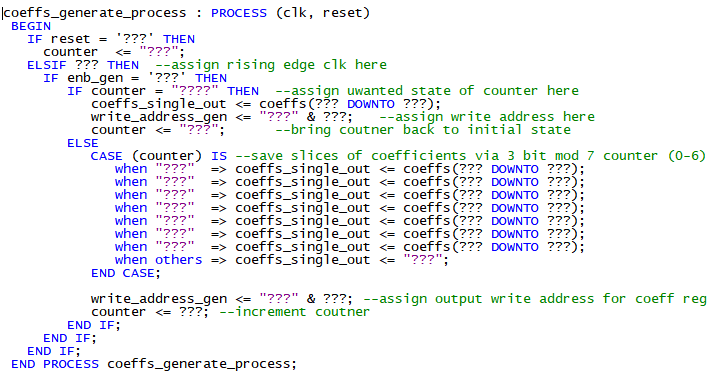
For the first task, go to the generator.vhd file, and go to line 26 to the process coeffs\_assign\_process. In this process we want to use a CASE statement to monitor changes in the input firselect, and assign the appropriate coefficients for each case. For this we want “00” to represent the high-pass filter coefficients, “01” to represent low-pass, “10” to represent band-pass and “11” to represent band-stop. You will need to convert the hexidecimal coefficients given to binary making sure that the number of bits is compatible with the coefficient width (same width as the word found in the coefficient register vhdl file, coeffs\_registers\_adc.vhd). As such you must also assign the width of the coeffs signal which must hold seven concatenated coefficients (for a seven tap filter). Hence you must fill in the widths of the following signal and assignment process

’

|  |
| --- |
| **Action: Completed code. 1 mark** |
| signal coeffs :std\_logic\_vector(97 downto 0); |

**Task 2:**

For the next task go to line 45 in the generator.vhd file to the process coeffs\_generate\_process. The functionality of this process should slice the coeffs signal and saving each slice in a corresponding location in the the coefficient register by assignging a write address. This process is handled by a 3 bit MOD 7 counter, where the coefficient slices are stored in the coeffs\_single\_out output port. For the counter we need to ensure that all transactions are clocked on the rising edge. The reset needs initialize the counter to its “000” state (initial state). We need to make sure that generation of coeffcients is handled with an enable signal, when such a signal is high the process should state assigning coefficients and addresses. As we have a 3 bit MOD 7 counter there is an unwanted state, hence we need to save the first slice of filter coefficients into this unwanted case incase this unwanted state is reached at any point of the process. You need to fill the functionality of the process as follows



|  |
| --- |
| **Action: Completed code. 1 mark** |
| coeffs\_generate\_process : PROCESS (clk, reset)  BEGIN  IF reset = '1' THEN  counter <= "000";  ELSIF clk'EVENT AND clk = '1' THEN  IF enb\_gen = '1' THEN  IF counter = "111" THEN  coeffs\_single\_out <= coeffs(13 DOWNTO 0);  write\_address\_gen <= "00000" & counter;  -- firselect<=firselect\_signal;  counter <= "000";  ELSE  CASE (counter) IS  when "000" => coeffs\_single\_out <= coeffs(97 DOWNTO 84);  when "001" => coeffs\_single\_out <= coeffs(83 DOWNTO 70);  when "010" => coeffs\_single\_out <= coeffs(69 DOWNTO 56);  when "011" => coeffs\_single\_out <= coeffs(55 DOWNTO 42);  when "100" => coeffs\_single\_out <= coeffs(41 DOWNTO 28);  when "101" => coeffs\_single\_out <= coeffs(27 DOWNTO 14);  when "110" => coeffs\_single\_out <= coeffs(13 DOWNTO 0);  when others => coeffs\_single\_out <= "00000000000000";  END CASE;    write\_address\_gen <= "00000" & counter;  counter <= std\_logic\_vector( unsigned(counter) + 1 );  END IF;  END IF;  END IF;  END PROCESS coeffs\_generate\_process; |

**Task 3:**

The FIR coefficients were generated using the MATLAB Filter designer program. There are four filters: a high pass filter, a low pass filter, a band pass filter, and a band stop filter. The coefficients can be replaced by using the steps in Appendix A.

|  |  |
| --- | --- |
| Filter mode | Coefficient |
| High Pass | 0167 3dee 36fa 1310 36fa 3dee 0167 |
| Low Pass | 3ec2 038c 1200 1b12 1200 038c 3ec2 |
| Band Pass | 386d 33eb 0759 1852 0759 33eb 386d |
| Band Stop | 058f 0a3d 3a7a 1cc5 3a7a 0a3d 058f |

Table 2: Coefficients of the FIR Filter

Convert

|  |
| --- |
| **Action: Convert coefficients and add them to the code. 1 mark** |
| when "00" => coeffs <= "00000101100111111101111011101101101111101001001100010000110110111110101111011110111000000101100111";  when "01" => coeffs <= "11111011000010000011100011000100100000000001101100010010010010000000000000111000110011111011000010";  when "10" => coeffs <= "11100001101101110011111010110001110101100101100001010010000111010110011100111110101111100001101101";  when "11" => coeffs <= "00010110001111001010001111011110100111101001110011000101111010011110100010100011110100010110001111";  when others => coeffs <= "00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"; |

**Task 4: Finish convolution code:**

The code to convolve the FIR coefficients with the incoming data from the ADC is incomplete. You will need to complete the code and copy paste it 7 times, slightly changing some of the variables each time.

In the Discrete\_FIR\_Filter\_adc.vhd file due to the multiplication and accumulation of the ADC sampled data with the FIR coefficients, we need to ensure that we don’t have data overflow. Hence we must create a limiter for the overflow condition. If the positive accumulated result is greater than the maximum positive number that could be represented by the bit width (14 bits) we would limit the result by the maximum allowed positive number. This is the same for the minimum allowable negative number. Hence for the “productx” assignment you need to assign the maximum allowable positive number for a bit width of 14, along with a minimum allowable negative number for the same bit width (for 14 bits the maximum is defined as 01111111111111111111111111111, take the 2’s compliment to find the negative value). For the first WHEN condition corresponding to the maximum positive value. we need to check the sign bit of the multiplier\_mul\_temp signal (for 14 bit coefficient data width and 14 bit input data width). Then we need to check that the next most significant bits are non-zero, this is the condition for overflow hence when it is reached we must assign the limited value (maximum postivie number for 14 bit data). The same must be done with the 2’s compliment for the minimum negative value allowable by the data width of 14 bits, hence we check if the sign bit is 1, and the next two most significant bits are zero (for overflow condition of a negative number). Else we simply assign the result of the multiplication as multiplier\_mul\_temp to the product.

After this product is assigned, we need to assign the delay for the filter. This is obtained for the Delay\_Pipeline\_process and must correspond to the appropriate input data for the discrete FIR filter.

Discrete\_FIR\_Filter\_coeff\_0 <= signed(Discrete\_FIR\_Filter\_coeff(**?????**));​

multiplier\_mul\_temp <=**???** \* **????**;

 product1 <= "**?????**" WHEN (multiplier\_mul\_temp(**???**) = '0') AND (multiplier\_mul\_temp(**???**DOWNTO **????**) /= "00") ELSE​  
      "**?????**" WHEN (multiplier\_mul\_temp(**????**) = '1') AND (multiplier\_mul\_temp(**???** DOWNTO **???**) /= "11") ELSE​  
      multiplier\_mul\_temp(25 DOWNTO 0) & '0' & '0' & '0';​

 ​

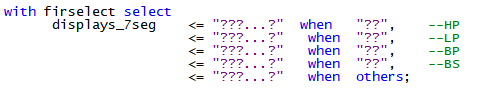
  delay\_pipeline\_0 <= delay\_pipeline\_1(**????**);​

 Complete this code: You may look at the other modules in the same section for hints.

|  |
| --- |
| **Action: Convolutional Code. 1 mark** |
| Discrete\_FIR\_Filter\_coeff\_0 <= signed(Discrete\_FIR\_Filter\_coeff(0));  multiplier\_mul\_temp <= Discrete\_FIR\_Filter\_in\_signed \* Discrete\_FIR\_Filter\_coeff\_0;  product1 <= "01111111111111111111111111111" WHEN (multiplier\_mul\_temp(27) = '0') AND (multiplier\_mul\_temp(26 DOWNTO 25) /= "00") ELSE  "10000000000000000000000000000" WHEN (multiplier\_mul\_temp(27) = '1') AND (multiplier\_mul\_temp(26 DOWNTO 25) /= "11") ELSE  multiplier\_mul\_temp(25 DOWNTO 0) & '0' & '0' & '0';  delay\_pipeline\_0 <= delay\_pipeline\_1(0); |

**Task 5: Finish 7 segment code:**

The functionality of the seven segment is to respond to the firselect signal which signifies the selection of filter coefficients for the function of FIR filter with the choices being high-pass, low-pass, band-pass and band-stop which correspond to numbers 0-3 respectively. To signify what filter has been selected we wish to display the corresponding filter to the fir select switch selection as H.P, L.P, B.P and B.S respectively. To do this you must fill in the segment code (for two 7 bit displays for a total of 14 bits) for two seven segment displays to display the desired filter type. Fill in the selection of active bits in the seven-segment display for each corresponding filter as directed by the code comments.

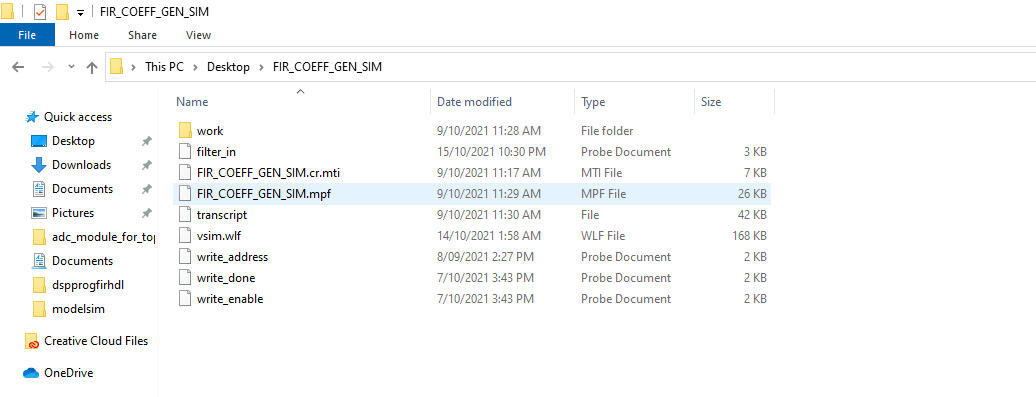


|  |
| --- |
| **Action: Seven Segment Code. 1 mark** |
| with firselect\_1 select  displays\_7seg <= "10010000011000" when "00", --HP  "11100010011000" when "01", --LP  "11000000011000" when "10", --BP  "11000001101100" when "11", --BS  "11111111111111" when others; |

# Simulation

Using modelsim, simulate the design using the testbench files located in the sim folder.

Ask the lab demonstrator if you do not know how to use modelsim. For simulation of the FIR filter please use the following files provided in the sim folder of the lab; MaxADC\_TOP, MaxADC\_TOP\_tb, Programmable\_FIR\_via\_Registers\_adc, reg\_adc, generator, Discrete\_FIR\_Filter\_adc and coeffs\_registers\_adc. You need to ensure that you have the relevant .dat files for use in the simulation. To simulate bring these files into a modelsim project, ensure that you have the relevant .dat files (filter\_in.dat, write\_address.dat, write\_done.dat and write\_enable.dat) and have them copied into the working directory of your simulation as so



After this is complete, make sure you include all relevant signals as per the image contained in the simulation folder that details the finished simulation. Make sure the relevant signals (filter\_in, Filter\_out\_1 and the signal mapped to coefficients) are displayed in an analogue format to observe the filtering process and confirm correct function of all filters.

|  |
| --- |
| **Action: Paste Outputs for DAC Simulation. 2 marks** |
|  |

# Synthesis

The completed and verified system now requires synthesis before it can be realized on the FPGA. Quartus Prime is used to complete this process.

## Analysis and Synthesis

The steps to complete this section are as follows:

1. Launch Quartus Prime
2. Select the src directory as the current working directory.
3. Select the top-level entity as MaxADC\_Top
4. Select Empty project, then Next
5. Add the project from the src directory.
   1. Add all of the files to the project.
6. Click Assignments, Device, Board, Family, Max 10, then Board, Max10 De 10 Lite. This will set the target device to the correct model.
7. Go to the Tasks window, click Analysis and Synthesis, Start. Errors will need to be corrected.

## RTL View

A top level view can be used to follow an implementation’s logic visually. To verify that the code is working properly, RTL views should be screenshot and pasted into the respective action items.

1. Click Tools, Netlist Viewers, RTL Viewer.
2. Check RTL View.
3. Take a screenshot of the diagram that appears. This is the RTL view of the top file.
4. Click on he ADC, FIR, and DAC components to expand them and show the RTL views of them as well.

|  |
| --- |
| **Action: RTL View: MaxADC\_TOP.vhd. 0.5 marks** |
|  |

|  |
| --- |
| **Action: RTL View: Programmable\_FIR\_via\_Registers\_adc. 0.5 marks** |
|  |

|  |
| --- |
| **Action: RTL View: ADC.vhd. 0.5 marks** |
|  |

|  |
| --- |
| **Action: RTL View: pmod\_dac121s101.vhd. 0.5 marks** |
|  |

## Fitter

Go to Tasks, then click Fitter, Start. Any errors will need to be corrected.

In the Table of Contents window in the Fitter, double click Messages. Select the text displayed and then paste it into the following action item field.

|  |
| --- |
| **Action: Fitter Output. 1 mark** |
| Info: Running Quartus Prime Fitter Info (171003): Fitter is performing an Auto Fit compilation, which may decrease Fitter effort to reduce compilation time Info (169124): Fitter converted 8 user pins into dedicated programming pins Info (169124): Fitter converted 16 user pins into dedicated programming pins Critical Warning (169085): No exact pin location assignment(s) for 94 pins of 130 total pins. For the list of pins please refer to the I/O Assignment Warnings table in the fitter report. Info (171121): Fitter preparation operations ending: elapsed time is 00:00:26 Info (14896): Fitter has disabled Advanced Physical Optimization because it is not supported for the current family. Info (170189): Fitter placement preparation operations beginning Info (170190): Fitter placement preparation operations ending: elapsed time is 00:00:02 Info (170191): Fitter placement operations beginning Info (170137): Fitter placement was successful Info (170192): Fitter placement operations ending: elapsed time is 00:01:10 Info (170193): Fitter routing operations beginning Info (170199): The Fitter performed an Auto Fit compilation. Optimizations were skipped to reduce compilation time. Info (170194): Fitter routing operations ending: elapsed time is 00:00:20 Info (11888): Total time spent on timing analysis during the Fitter is 12.95 seconds. Info (11218): Fitter post-fit operations ending: elapsed time is 00:00:28 Info: Quartus Prime Fitter was successful. 0 errors, 19 warnings |

## Assembler

Go to Tasks, then click Assembler, Start. Any errors will need to be corrected.

In the Table of Contents window in the Assembler, double click Messages. Select the text displayed and then paste it into the following action item field.

|  |
| --- |
| **Action: Assembler Output. 1 mark** |
| Info: Running Quartus Prime Assembler Info: Version 20.1.1 Build 720 11/11/2020 SJ Lite Edition Info: Processing started: Fri Oct 15 20:41:13 2021 Info: Command: quartus\_asm --read\_settings\_files=off --write\_settings\_files=off lab2\_top -c lab2\_top Warning (18236): Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM\_PARALLEL\_PROCESSORS in your QSF to an appropriate value for best performance. Info (115031): Writing out detailed assembly data for power analysis Info (115030): Assembler is generating device programming files Warning (12914): The file, C:/Users/jonlo/OneDrive/Desktop/Lecturer-20211011T092710Z-001/Lecturer/ELEC4406\_LAB2/synth/lab2\_top/adc\_qsys.sopcinfo, is not embedded into sof file as expected. Some tools, such as System Console, may not function fully. Info: Quartus Prime Assembler was successful. 0 errors, 2 warnings Info: Peak virtual memory: 4693 megabytes Info: Processing ended: Fri Oct 15 20:41:53 2021 Info: Elapsed time: 00:00:40 Info: Total CPU time (on all processors): 00:00:13 Info (293026): Skipped module Power Analyzer due to the assignment FLOW\_ENABLE\_POWER\_ANALYZER |

## Timing Analysis

Find lab\_top.sdc in the src folder and ensure this code is written in it:

create\_clock -period "50.0 MHz" [get\_ports clk\_in]

Go to tasks window, click Timing Analyzer, then Start.

Using the output from the timing analyzer to explain why   
Use the class lecture material if you need help understanding which outputs mean what.

Hint: Failure to meet timing constraints results in the section being highlighted with red text.

|  |
| --- |
| **Action: Explain timing analysis. 3 marks** |
| Marks up to teacher’s discretion. |

## Device Configuration

1. Configure the Max 10 device following these steps:
2. Launch Quartus Prime Programmer
3. Check that the Hardware Setup shows the USB port the board is connected to.
4. Check that the mode is set to JTAG.
5. Click Add File, select the lab2\_top.sof file.
6. Verify that the device defined near Add Device is 10M50DAF484.
7. Click start to begin the process.

## Hardware Testing

Test the hardware using the setup instructions in the User manual and test the different filters. For each filter, take a screenshot or photo of the display of the oscilloscope.

On the oscilloscope, press the pause button or adjust the trigger so the waveform stays in place.

Switch between the filters using the switches and the buttons as discussed earlier. Test the low pass filter by introducing a low frequency signal, and then a high frequency signal. The signals should be selected using the wave generator by increasing the frequency until the signal begins to be dampened by the stop band.

Take a screenshot or photo of the oscilloscope showing the filter correctly filtering a high frequency signal and allowing a low frequency signal to pass.

|  |
| --- |
| **Action: Testing Low pass with a high frequency signal. 1 mark** |
|  |

|  |
| --- |
| **Action: Testing Low Pass with a low frequency signal. 1 mark** |
|  |

# References

|  |  |
| --- | --- |
| [1] | terasIC, "De10-Lite User Manual," 5 June 2020. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive\_download.pl?Language=English&No=1021&FID=a13a2782811152b477e60203d34b1baa. |
| [2] | Keysight Technologies, "InfiniiVision 2000 X-Series Oscilloscopes Data Sheet," 2021 May 4. [Online]. Available: https://www.keysight.com/au/en/assets/7018-02733/data-sheets/5990-6618.pdf. |
| [3] | Digilent, "PmodDA2 Reference Manual," 24 May 2016. [Online]. Available: https://digilent.com/reference/\_media/reference/pmod/pmodda2/pmodda2\_rm.pdf. |

# Appendix A

Open the MATLAB filter designer using this command on the MATLAB command line:

filterDesigner

2. The filter designer should have opened on the main screen. Design a lowpass filter using the options given on the main page.

Graphical user interface

Description automatically generated

3. Click ‘Design Filter’ at the bottom of the GUI.

4. Go to Filter Arithmetic under the 3rd tab from the top on the left panel.

5. Set the Filter Arithmetic field to Fixed-Point

6. Go to File, then Export, then select these options in the popup, and click the Export button.

Graphical user interface, application

Description automatically generated