SİVAS UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

DIGITAL SIGNAL PROCESSING

Experiments Manual

Supervisor: Asst. Prof. Nurhan Güneş
Assoc. Prof. Nurbanu Güzey
Assoc. Prof. Yusuf Doğan

Prepared by: Res. Asst. Berke Can Turan
Res. Asst. Şekip Dalgaç

SİVAS

TABLE OF CONTENTS

Experiment 1: Generation of Discrete Time Signals in MATLAB and on FPGA

Objective

The objectives of this experiment are

- to generate digital signals by programming an FPGA development board, and then to produce analog signals by driving a ladder circuit with the digital signals.
- to write a "MATLAB" Program to generate discrete-time signals like unit impulse, unit step, unit ramp, exponential signal, and sawtooth signals.

Required Equipment

- PC
- MATLAB (A programming and numeric computing platform used to analyze data, develop algorithms, and create models.)
- WaveForms (A virtual instrument suite for Digilent Test and Measurement devices.)
- Analog Discovery Studio (A portable circuits laboratory for students)
- The Basys 3 board (A complete, ready-to-use digital circuit development platform based on the latest AMD Artix™ 7 Field Programmable Gate Array (FPGA) from AMD.)
- Vivado (An application used to develop projects to run on Digilent FPGA Development Boards)

Theoretical Background

The analysis and synthesis of discrete-time signals are fundamental to system modelling, filter design, and communication applications. The first experiment emphasises generating and visualising basic discrete-time signals, which serve as essential building blocks in signal theory.

- **Unit Impulse Function (δ[n]):** The unit impulse, also known as the Dirac delta function in discrete-time, is defined as:

$$\delta[n] = \begin{cases} 1, \text{if } n = 0 \\ 0, \text{otherwise} \end{cases}$$

  This signal is widely used for system identification and serves as the identity element in convolution operations. It plays a crucial role in characterizing the response of linear time-invariant (LTI) systems.
- **Unit Step Signals (u[n]):** The discrete-time unit step function is defined as:

$$u[n] = \begin{cases} 1, \text{if } n \geq 0 \\ 0, \text{if } n < 0 \end{cases}$$

  The unit step signal models systems that switch on at a certain point and remain active thereafter. It is frequently used in constructing piecewise-defined signals and analyzing system causality.
- **Unit Ramp Signal (r[n]):** The discrete ramp signal increases linearly for non-negative nnn:

$$r[n] = \begin{cases} n, \text{if } n \geq 0 \\ 0, \text{otherwise} \end{cases}$$

  This signal is often employed to simulate linearly increasing inputs in system testing and to examine the stability and response characteristics of DSP systems.
- **Exponential Signal:** The discrete-time exponential signal is expressed as:

$$x[n] = e^{an}$$

  Where a determines the growth or decay rate. For a <0 the signal decays exponentially; for a>0 it grows. This signal type models many physical systems including charge/discharge phenomena and system responses with exponential transients

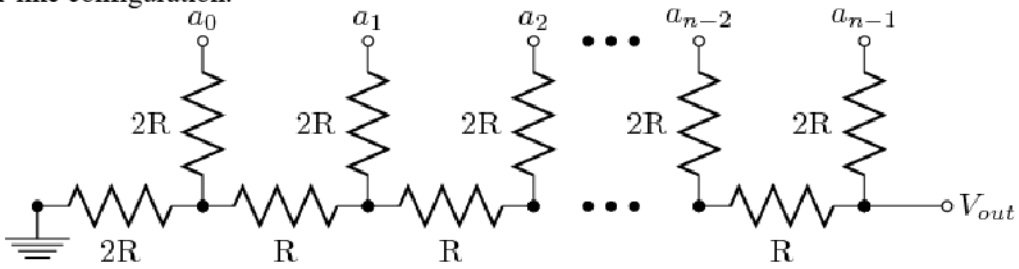
- **Sawtooth Signal:** The sawtooth waveform is a non-sinusoidal periodic signal characterized by a linear rise and a sudden drop. In discrete form, it can be defined using periodic trigonometric functions such as

  $$x[n] = \text{sawtooth}(\omega n)$$

  This waveform is used in waveform generation, modulation techniques, and control systems where a linearly ramping reference is required.

R-2R DACs

One of the most common DAC building-block structures is the R-2R resistor ladder network shown in the figure. It uses resistors of only two different values, and their ratio is 2:1. An N-bit. DAC requires 2N resistors, and they are quite easily trimmed. An R–2R ladder configuration is a simple and inexpensive way to perform digital-to-analog conversion (DAC), using repetitive arrangements of precise resistor networks in a ladder-like configuration.



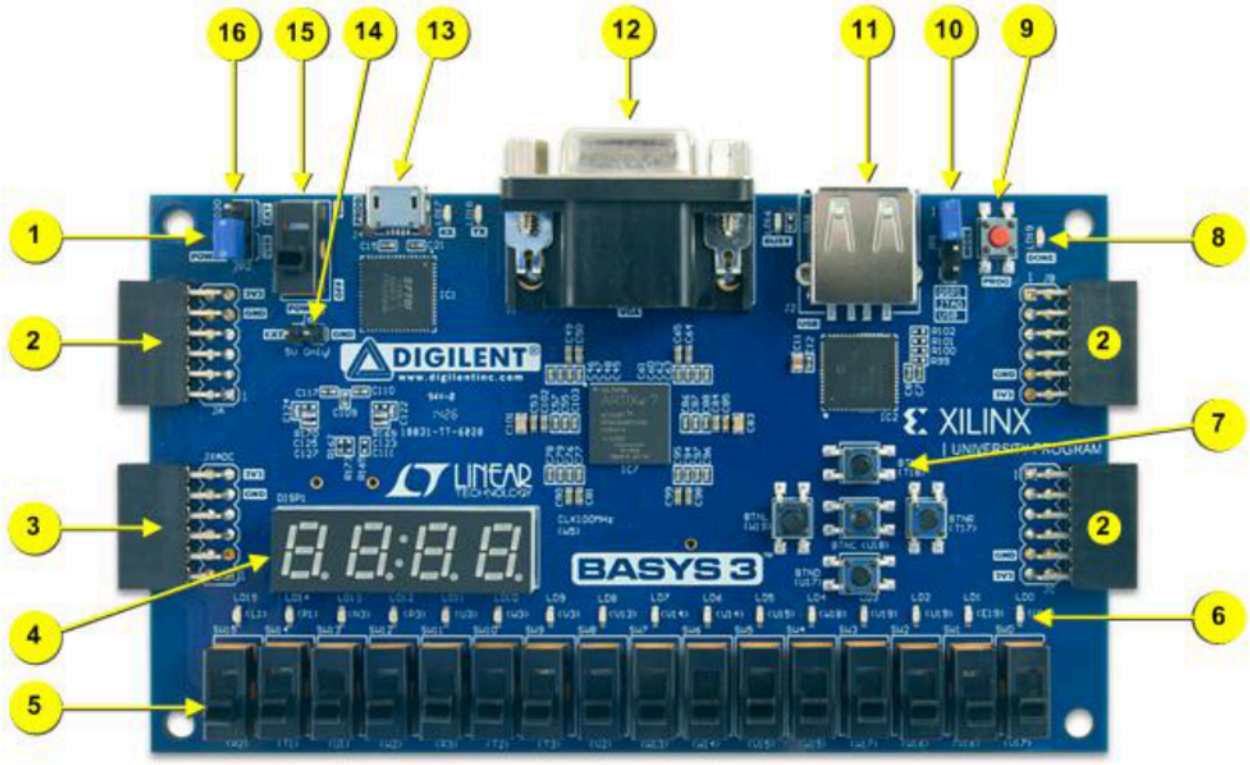
A general-purpose input/output (GPIO) is an uncommitted digital signal pin on an integrated circuit or electronic circuit (e.g. MCUs/MPUs) board that can be used as an input or output, or both, and is controllable by software.

General-purpose input/output

GPIOs have no predefined purpose and are unused by default.[1][2] If used, the purpose and behavior of a GPIO is defined and implemented by the designer of higher assembly-level circuitry: the circuit board designer in the case of integrated circuit GPIOs, or system integrator in the case of board-level GPIOs.

Basys 3

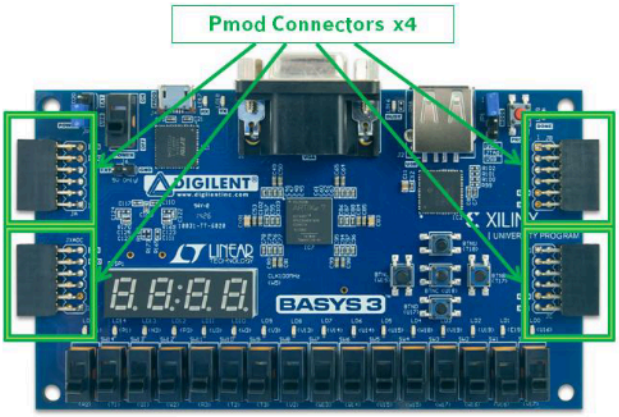
Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits.

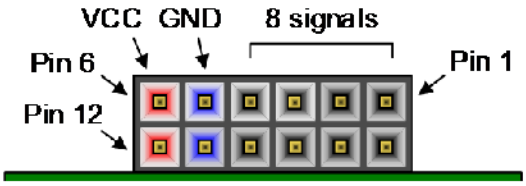


Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Digilent produces a large collection of Pmod (Peripheral Module) accessory boards that can attach to the expansion ports to add ready-made functions such as A/D's, D/A's, motor drivers, sensors, displays, and many other functions. These ports can be used as simple expansion ports, since all of the pin-outs correspond to pins on the FPGA.

The Pmod ports are arranged in a 2×6 right-angle, and are 100-mil female connectors that mate with standard 2×6 pin headers. Each 12-pin Pmod ports provides two 3.3V VCC signals (pins 6 and 12), two Ground signals (pins 5 and 11), and eight logic signals. The VCC and Ground pins can deliver up to 1A of current. Pmod data signals are not matched pairs, and they are routed using best-available tracks without impedance control or delay matching.





Procedure (MATLAB)

- Open MATLAB
- Open a new M-file
- Type the code block given below
- Save in current directory
- Compile and Run the program
- For the output, see the command window\ Figure window

Experiment (MATLAB)

```
%unit impulse function%
%Discrete%
n=-10:10;
Xn=(n==0);
subplot(3,2,1);
stem(n,Xn);
axis([-11 11 -0.5 1.5]);
xlabel('Samples');
ylabel('amplitude');
title(' unit impulse function');
%unit step function%
%Discrete%
n=-10:10;
Xn=(n>=0);
subplot(3,2,2);
stem(n,Xn);
axis([-11 11 -0.5 1.5]);
xlabel('Samples');
ylabel('amplitude');
title(' discrete unit step function');
%unit ramp function%
%Discrete%
n=-10:10;
Xn=(n>=0).*n;
subplot(3,2,3);
stem(n,Xn);
axis([-11 11 -1 11]);
```

```
xlabel(' Samples');
ylabel('amplitude');
title(' discrete unit ramp function'); DIGITAL SIGNAL PROCESSING LABORATORY
(20A04502P) % exponential signal:
%Discrete%
n2=input('enter the length of the exponential sequence');
t=0:n2;
a=input('enter the a value');
y2=exp(a*t);
subplot(3,2,4);
stem(t,y2);
ylabel('amplitude');
xlabel('time period');
title('exponential sequence')
%sawtooth signal
%Discrete%
n=0:10;
Xn=sawtooth(pi*n/4);
subplot(3,2,5);
stem(n,Xn);
axis([-0.5 11 -1.5 1.5])
xlabel('time period');
ylabel('amplitude');
title('sawtooth sequence');
```

Evaluation of Results (MATLAB)

- Define impulse, unit step, ramp signals and write their expressions?
- Define exponential and sinusoidal signals and write their expressions?
- Express unit step signal in terms of unit impulse?
- Express ramp signal in terms of unit step signal?
- Represent the signal x[n]={1,2,-1,3,2}using impulse signal?

Procedure (FPGA)

- In Vivado's welcome screen, click the Create Project button to open a wizard used to begin creating a Vivado project from scratch.
- The first page of the New Project wizard summarizes the steps involved in creating a project. Click Next.
- The first step is to set the name for the project. Vivado will use this name when generating its folder structure. Checking the Create project subdirectory box will create a new folder in the chosen location to store the project's files. This is recommended. And then, Click Next to continue.
- At the Select Project Type screen, choose RTL Project and check the Do not specify sources at this time box. Click Next to continue.
- Next, a part or a board must be chosen for the project to target. The project will only be usable with the chosen device (though the selection can later be changed through the project's Settings). Search for your board and select it from the list. Click Next to continue.
- The last screen of the New Project wizard summarizes what was chosen in the previous screens. Click Finish to open your project.

The Flow Navigator is the most important pane of the main Vivado window to know. It is how a user navigates between different tools within Vivado. The Navigator is broken into seven sections:

- Project Manager: Access project settings, add new source files, view snippets of useful RTL code, and add components of a hardware design (IP) included within the tool to your own design.
- IP Integrator: Create a hardware design through the use of a Block Design by connecting IP blocks together.
- Simulation: Debug and verify a hardware design fully within software, without deploying it to hardware.
- RTL Analysis: See how the tools are interpreting a design by viewing the circuit that the design describes.
- Synthesis: Modify Synthesis settings and view post-synthesis reports - determine whether the design meets timing, and how much of the chip resources it will use.
- Implementation: Modify Implementation settings and view post-implementation reports - determine whether the design meets timing, and how much of the chip resources it will use.
- Program and Debug: Access to settings for bitstream generation and the Hardware Manager. Used to build a design and deploy it to a board.

The Project Manager

This tool is where most development will occur and is the initial tool open after creating a new project. The Project Manager consists of four panes, Sources, Properties, Results, and the Workspace. The Sources pane contains the project hierarchy and is used for opening up files. The folder structure is organized such that the HDL files are kept under the Design Sources folder, constraints are kept under the Constraints folder, and simulation files are kept under the Simulation Sources folder. Files can be opened in the Workspace by double-clicking on the corresponding entry in the Sources pane. Sources can also be added by either right-clicking the folder to add the file to and selecting Add Sources or by clicking the Add Sources button.

The most important pane in the Project Manager is the Workspace. The Workspace is where reports are opened for viewing and HDL/constraints files are opened for editing. Initially, the Workspace displays the Project Summary which show some basic information from some of the reports.

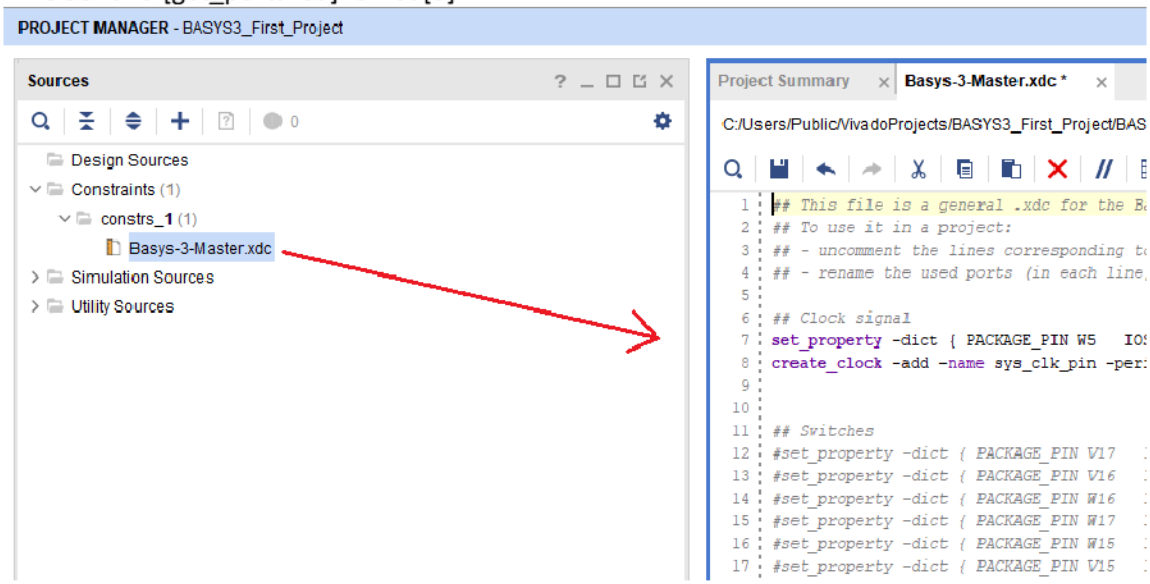
Adding a Constraint File

Constraint files describe the requirements put upon a design by the components surrounding an FPGA - including all peripherals, external clocks, and more.

In order to connect the ports found in an RTL top module (described later) with the physical pins of the FPGA, a constraint file needs to be added or created. Digilent provides Xilinx Design Constraint (XDC) files for each board. Download digilent-xdc-master.zip, the ZIP Archive containing each of these master XDC files, then extract it in a location you will remember.

- In the Project Manager section of the Flow Navigator, click the button. In the wizard that pops up, select Add or create constraints then click Next.
- At this stage, Vivado provides a list of all of the constraint files that will be added or created when we click Finish. Currently, this list is empty, this will change when files have been added or created. A constraint file will not be created from scratch in this guide, so click Add Files.
- Find the directory you extracted the digilent-xdc-master.zip archive into, then click on the file for your board. This should add the name of the file to the File Name field. Click OK to continue.
- Make sure that the selected XDC file has been added to the list of sources, and that the Copy constraint files into project is checked, then click Finish.

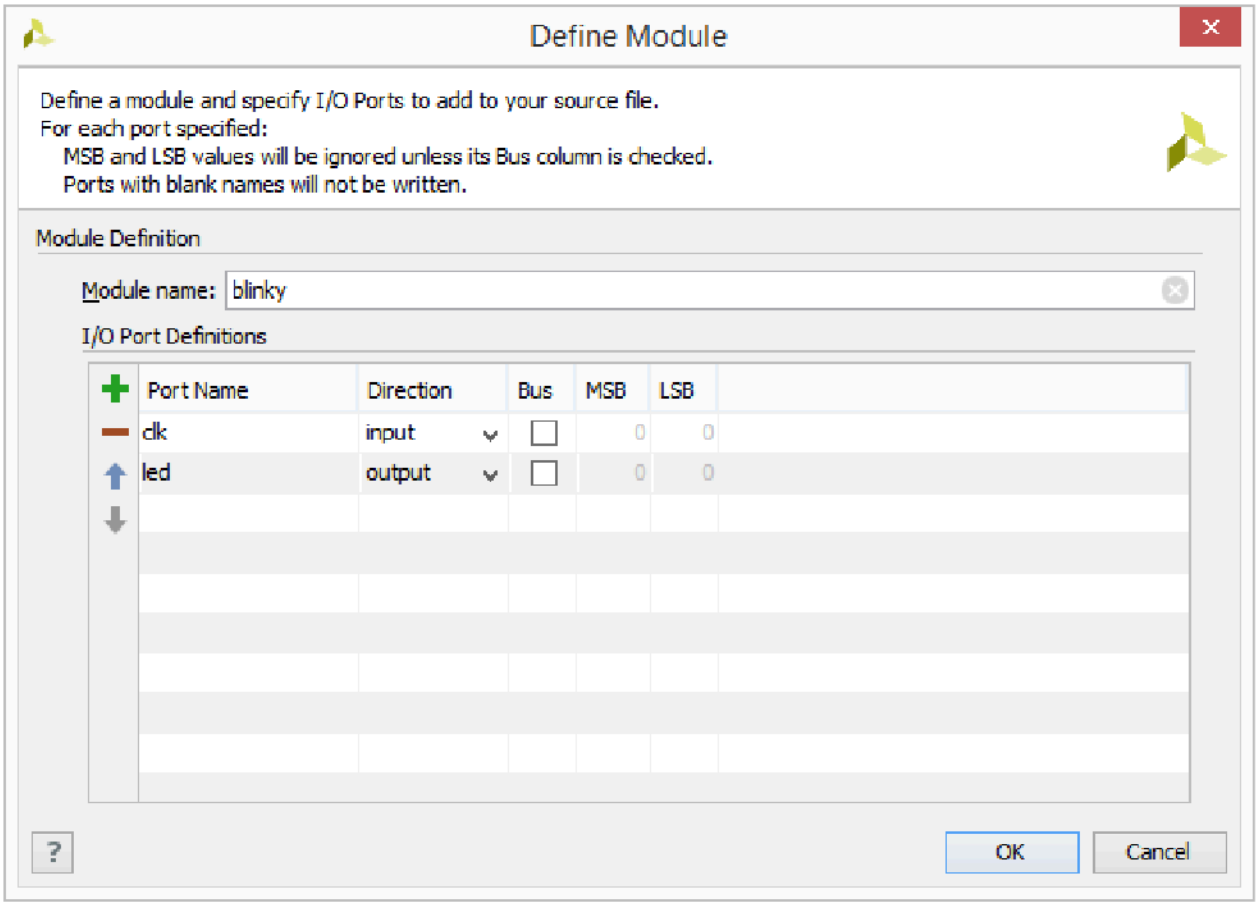
- Find the (.xdc) constraints file in the Sources section of the project. Double-click it to view the contents. This is the file that maps your design signal names to FPGA pins. In this example, we are going to use clk for the system clock, Pmod JA, and a led for the indicator led[0]. Uncomment these by removing the leading # character and make sure the get_ports name is [get_ports clk] for the clock and [get_ports led] for led[0].



Creating a Verilog Source File

Verilog is one of several hardware description languages (HDLs) that can be used within Vivado to describe a circuit to be implemented within an FPGA. This section describes how to create a Verilog file within Vivado, and create a simple circuit that will work on any Digilent FPGA development board.

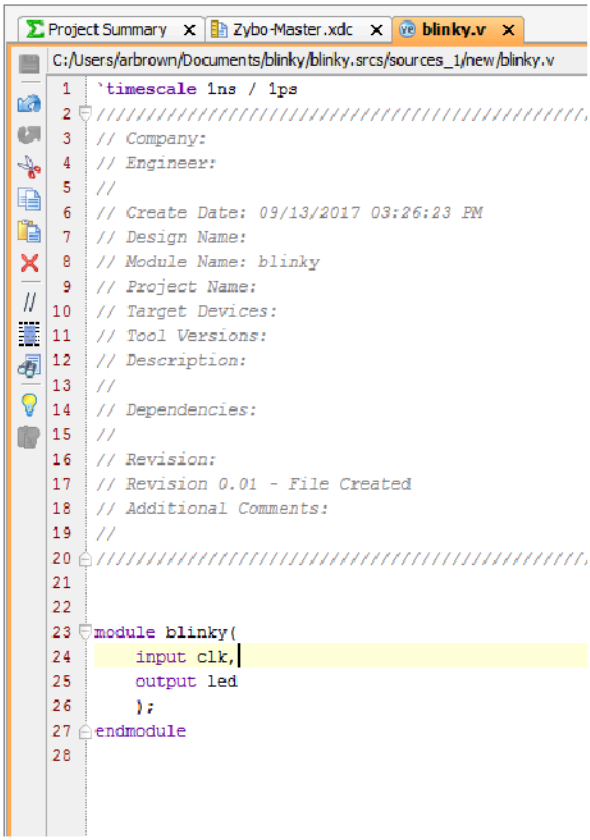
- In the Project Manager section of the Flow Navigator, click the Add Source button again. Select Add or create design sources then click Next.
- As before, at this stage, we will be provided a list of all of the source files that will be added or created when we click Finish. Instead of clicking Add Files, click Create File.
- You will be prompted to select a File type, File name, and File location. Make sure to pick Verilog and <Local to project> for the type and location. Give your file a name ending in '.v'. Click OK to continue. Make sure that the new Verilog source file has been added to the list of sources, then click Finish.
- Unlike when the constraint file was added, at this point, a Define Module dialog will pop up. You can rename your Verilog module using the Module name field, but this is unnecessary. The Verilog module's clock and led ports need to be defined. Clicking the Add (+) button will add an empty slot for a port to the I/O Port Definitions list.



- Add JA port as a bus, and enter 7 in MSB. Once these three ports have been added, click OK to continue.
- Now, everything is ready for the simulation process.

Simulation (FPGA)

At this point, the new source has been added to the Design Sources folder in the Sources pane of the Project Manager. Expand this folder and double-click on the file to open it. Next, some Verilog code needs to be written to define how the design will actually behave.



Between the ');' that comes after the module's port list and the 'endmodule' statement, add the following lines of code. (Just replace the module with the following one.)

```
module my_source(
    input clk,
    output [7:0] JA,
    output led
    );

reg [24:0] count = 0;
wire [7:0] myc = count[20:13];

assign led = count[24];

assign JA = myc;

always @ (posedge(clk)) count <=
count + 1;

endmodule
```

This snippet of code implements a 25-bit counter, which will increment by 1 every clock cycle, and roll back over to 0 after ~33 million cycles (about a third of a second, given a 100MHz input clock). The LED is directly driven by the top bit of the counter, which, of the bits, changes the least frequently. Given a 100MHz clock, the LED will blink (go through a full cycle of low-to-high-to-low transitions) approximately three times per second.

Generate a Bitstream

In order to create a file that can be used to program the target board, each stage of the “compilation pipeline” needs to be run.

This starts with Synthesis. Synthesis creates a description of the logic gates and connections between them required to perform the functionality described by the HDL files, given the constraints included in XDC files. To run Synthesis click Run Synthesis either in the toolbar or in the Flow Navigator. The output of Synthesis is then passed to Implementation.

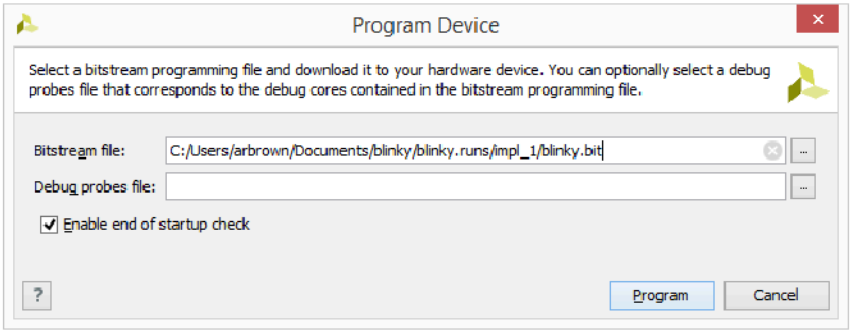
Implementation has several steps. The steps that are always run are Opt Design (Optimize the design to fit on the target FPGA), Place Design (Lay out the design in the target FPGA fabric), and Route Design (Route signals through the fabric). To run Implementation click Run Implementation either in the toolbar or in the Flow Navigator. This output is then passed on to the Bitstream Generator.

Before the bitstream part, go to settings for bitstream and put a tick to have ‘.bin’ file.

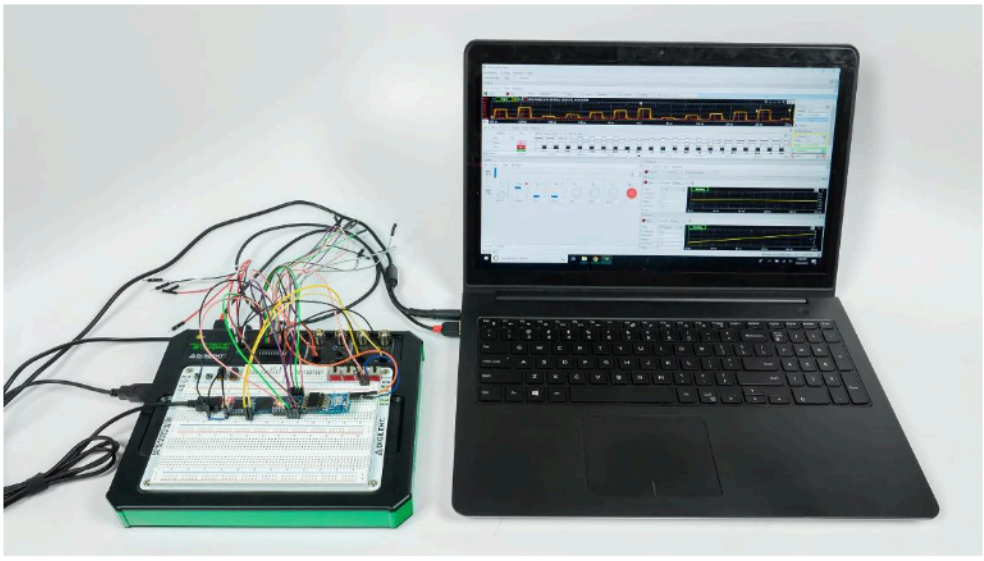
The Bitstream Generator generates the final output file needed for programming the FPGA. To run Bitstream Generation, click Generate Bitstream either in the toolbar or in the Flow Navigator. With no settings changed, the generator will create a '.bit' file.
Depending on the complexity of the design, the board used, and the strength of your computer, the process of building the project can take between 5 and 60 minutes. When complete, a pop-up dialog will appear, prompting you to select one of several options. None are relevant for the purposes of this guide, so click Cancel. The “write_bitstream complete” status message can be seen in the top right corner of the window, indicating that the demo is ready to be deployed to your board.

Vivado's Hardware Manager can be opened by clicking on Open Hardware Manager at the bottom of the Flow Navigator pane on the left side of the Vivado window.

To program the device with the bit file generated earlier, either click the Program device link in the green banner at the top of the window.



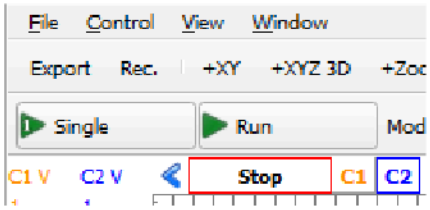
The Bitstream File field should be automatically filled in with the bit file generated earlier. Now click Program. This will connect to the board, clear the current configuration, and program it using the new bit file.
You should now see one of the LEDs on your board blinking!
Now, check if the JA ports output square waves. To do so, plug in the Test and Measurement Device (Analog Discovery Studio), then start WaveForms and make sure the device is connected.



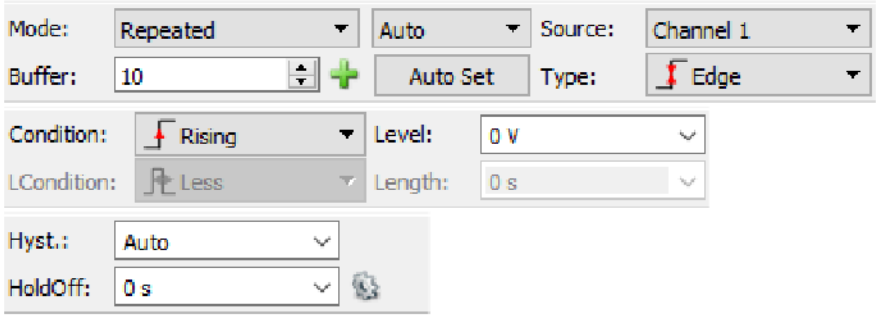
Once the Welcome page loads, in the instrument panel at the left side of the window, click on the Scope button to open the Oscilloscope instrument.

Once the Scope instrument opens, the window contains the data plot showing captured data, the configuration panel to the right of the plot, and the control toolbar at the top of the window.

Use jumper wires to connect the oscilloscope's port 1 + and – with the Pmod JA 1 and GND. Then, run the scope.



Click the Auto Set to configure the Scope settings and to see the square wave clearly.



Check all JA ports, set up the resistor ladder circuit, and connect the JA ports to the resistor ladder. Use the scope to see the output of the ladder. You must see a sawtooth signal.

Evaluation of Results(FPGA)

- Analyse the module codes, find out the period of each JA port.

- Explain the implementation of the Verilog codes, find out what FPGA blocks are used, and their locations.
- Derive the output of the ladder circuit as a function of JA ports.
- Propose a similar module that can result in a sinusoidal output.

Safety Precautions

The Pmod pins are connected to AMD Artix™ 7 FPGA pins using a 3.3V logic standard; care should be taken not to drive these pins over 3.4V.

References

- https://vemu.org/uploads/lecture_notes/19_12_2022_1847616401.pdf
- https://www.mathworks.com/help/dsp/ug/discrete-time-signals.html
- https://www.analog.com/media/en/training-seminars/tutorials/MT-015.pdf
- https://en.wikipedia.org/wiki/General-purpose_input/output
- https://digilent.com/reference/programmable-logic/basys-3/reference-manual
- https://digilent.com/reference/programmable-logic/guides/getting-started-with-vivado
- https://digilent.com/reference/test-and-measurement/guides/waveforms-oscilloscope

Experiment 2: Sum of Two Sinusoidal Signals

Objective

To write a MATLAB program to find the sum of two sinusoidal signals and to find the frequency response (magnitude and phase).

Required Equipment

- PC
- MATLAB Software

Theoretical Background

In signal processing, the superposition or summation of signals plays a critical role in both analysis and synthesis of complex waveforms. According to the **principle of linearity**, any signal can be represented as a combination of simpler signals, often sinusoids, especially when dealing with linear time-invariant (LTI) systems.

Sine Wave Fundamentals

A sine wave is a fundamental periodic signal expressed as:

$$x(t) = Asin(2\pi ft + \varphi)$$

where:

- A is the amplitude,
- f is the frequency (in Hz),
- t is time,
- φ is the phase angle,

Sine waves are essential building blocks of more complex signals in Fourier analysis. When two or more sine waves are added together in the time domain, the resulting signal exhibits characteristics influenced by both constituent signals. If the frequencies are different, the resultant waveform exhibits beats or amplitude modulation patterns, depending on the frequency ratio and phase difference.

$$x_1(t)=Asin(2\pi f^1 t + \varphi_1), \quad x_2(t)= Bsin(2\pi f^2 t + \varphi_2)$$

Then their sum is: $y(t) = x_1(t) + x_2(t)$

This results in constructive and destructive interference, depending on the instantaneous phase alignment between the two waves.

Procedure

- Open MATLAB
- Open a new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output, see the command window\ Figure window

Simulation

```
Clear all;
Close all;
t=0:0.001:0.1;
f1=50;
x1=2*pi*f1*t+pi/7;
y1=sin(x1);
figure;
subplot (3,1,1);
plot (t,y1);
title('sin(x1');
f2=100;
x2=2*pi*f2*t-pi/5;
y2=sin(x2);
subplot(3,1,2);
plot(t,y2);
title('sin(x2)');
y=y1+y2;
subplot(3,1,3);
plot(t,y);
title('sinx1=sinx2')
```

Evaluation of Results

- How do you determine the sum of sinusoids?
- Does the sum have a frequency component higher than the operands?
- Calculate a frequency that makes the first sinusoidal signal have the same values for the same time variable.
- How do you add two sinusoidal currents? (in an electrical circuit)
- What is the amplitude of a sinusoidal function?
- What is meant by a sinusoidal signal?
- What is the phase of the sinusoidal function?
- Implement the module that you had already designed in the last laboratory session for a sinusoidal output.

Safety Precautions

References

- https://www.mathworks.com/help/matlab/ref/double.sin.html

Experiment 3: Linear and Circular Convolution

Objective

To write MATLAB programs to find out the linear convolution and Circular convolution of two sequences.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

Convolution is a mathematical operation that describes how one signal modifies another. In digital signal processing, it's used to determine the output of a linear time-invariant (LTI) system when its input and impulse response are known. In digital signal processing, convolution is a critical operation that describes how two signals interact over time or how a system shapes an input signal. While linear convolution reflects how a signal passes through a linear time-invariant (LTI) system, circular convolution is used in the context of periodic signals and systems, particularly in DFT (Discrete Fourier Transform)-based signal processing.

Linear Convolution

Given two discrete signals; $x[n]$ the input signal and $h[n]$ the impulse response of the system, the convolution $y[n]$ is defined as

$$y[n] = (x * h)[n] = k = \sum_{0}^{\infty} x[k] \cdot h[n-k]$$

If the signals are finite-length, say $x[n]$ has length N and $h[n]$ has length M, the equation simplifies to

$$y[n] = (x * h)[n] = k = \sum_{k=0}^{N-1} x[k] \cdot h[n-k]$$

with y[n] being defined for $n = 0$ to $N + M - 2$

Circular Convolution

$$y[n] = (x * h)[n] = k = \sum_{k=0}^{N-1} x[k] \cdot h[n-k] \quad mod\ N$$

$mod\ N$ operation ensures wrap-around (periodic extension). The output $y[n]$ is also of length N

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

Program for Linear Convolution

```
%Program for linear convolution
%to get the input sequence
n1=input('enter the length of input
sequence');
n2=input('enter the length of impulse
sequence');
x=input('enter the input sequence');
h=input('enter the impulse
sequence');
%convolution operation
y=conv(x,h);
%to plot the signal
subplot(3,1,1);
stem(x);
ylabel('amplitude');
xlabel('n1....>');
title('input sequence')
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('n2....>');
title('impulse signal')
subplot(3,1,3);
stem(y);
ylabel('amplitude');
xlabel('n3');
disp('the resultant signal is');y
```

Output: Linear Convolution

```
Enter the length of input sequence 4
Enter the length of impulse sequence
4
Enter the input sequence [1 2 3 4]
Enter the impulse sequence [4 3 2 1]
The resultant signal is
y= 4 11 20 30 20 11 4
```

Program for Circular Convolution

```
%%Program for circular convolution
clc;clear all;close all;
%to get the input sequence
g=input('enter the input sequence');
h=input('enter the impulse
sequence');
N1=length(g);
N2=length(h);
N=max(N1,N2);
N3=N1-N2
%loop for getting equal length
sequence
if(N3>=0)
h=[h,zeros(1,N3)];
else
g=[g,zeros(1,-N3)];
end
%computation of circular convoluted
sequence
for n=1:N;
y(n)=0;
for i=1:N;
j=n-i+1;
if(j<=0)
j=N+j;
end
y(n)=y(n)+g(i)*h(j);
end
end
figure;
subplot(3,1,1);
stem(g);
ylabel('amplitude');
xlabel('n1..>');
title('input sequence')
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('n2');
title('impulse sequence')
subplot(3,1,3);
stem(y);
ylabel('amplitude');
xlabel('n3');
disp('the resultant signal is');
```

```
OUTPUT : CIRCULAR CONVOLUTION
Enter the input sequence [1 2 2 1]
Enter the impulse sequence [4 3 2 1]
The resultant signal is
y= 15 17 15 13
```

Evaluation of Results

- Define Convolution?
- What are the types of Convolution?
- What is Linear Convolution & Circular Convolution?
- State the methods available to compute Convolution Sum?
- What is the significance of convolution?

Safety Precautions

References

- https://digilent.com/reference/programmable-logic/basys-3/demos/start
- https://www.mathworks.com/help/matlab/ref/conv.html
- https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_ch6.pdf

Experiment 4: Auto-Correlation and Cross-Correlation

Objective

To write a Matlab program to compute the autocorrelation for the given sequence and cross-correlation between two signals.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

In digital signal processing, correlation is a statistical and mathematical operation used to measure the similarity between two signals. It is a vital tool for analysing signal periodicity, synchronisation, pattern matching, and spectral features. This experiment focuses on two core types of correlation: autocorrelation and cross-correlation, applied to sinusoidal signals.

Auto-correlation measures the similarity of a signal with a delayed version of itself. For a continuous-time or discrete-time signal $x(t)$ or $x[n]$ the auto-correlation function $Rxx(\tau)$ is defined as:

$$Rxx(\tau) = \sum_{n} x[n] \cdot x[n-\tau]$$

In MATLAB, xcorr(x1, x2) computes the auto-correlation of signal x1.

Cross-correlation measures the similarity between two different signals, often to determine how much one signal is delayed with respect to another. It is mathematically defined as:

$$R_{xx}(\tau) = \sum_{n} x[n] \cdot x[n-\tau]$$

In MATLAB, xcorr(x1, x2) computes the cross-correlation between x1 and x2.

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
clc; clear all; close all;
t=0:0.01:1;
f1=3;
x1=sin(2*pi*f1*t);
figure;
subplot(2,1,1);
plot(t,x1);
title('sine wave');
```

```
xlabel('time ---->');
ylabel('amplitude---->');
grid;
[rxx lag1]=xcorr(x1);
subplot(2,1,2);
plot(lag1,rxx);
grid;
title('auto-correlation function of sine wave');
figure;
subplot(2,2,1);
plot(t,x1);
title('sine wave x1');
xlabel('time ---->');
ylabel('amplitude---->');
grid;
f2=2;
x2=sin(2*pi*f2*t);
subplot(2,2,2);
plot(t,x2);
title('sine wave x2');
xlabel('time ---->');,ylabel('amplitude---->');
grid;
[cxx lag2]=xcorr(x1,x2);
subplot(2,2,[3,4]);
plot(lag2,cxx);
grid;
title('cross-correlation function of sine wave');
```

Evaluation of Results

- Define Cross Correlation?
- What are the applications of Cross Correlation in signal de-noising?
- Define Cross Power Spectral Density?
- How is Cross Correlation different from Auto Correlation?
- Describe the formula to compute cross-correlation?

Safety Precautions

References

- https://www.mathworks.com/help/stats/corr.html
- https://www.mathworks.com/help/matlab/ref/xcorr.html
- https://www.mathworks.com/help/matlab/ref/corrcoef.html

Experiment 5: Discrete Fourier Transform

Objective

To find DFT of a given sequence and compute the power density spectrum of the sequence.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

Basic equation to find the DFT of a sequence is given below.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$where \quad W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \text{ [TWIDDLE FACTOR]}$$

Basic equation to find the IDFT of a sequence is given below.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1.$$

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
clc;
close all;
clear all;
tic;
fprintf('Date & Time:');
Date= datestr(now);
disp(Date);
%DFT
disp('D.F.T');
a=input('Enter the input sequence:');
n=length(a);
x=fft(a,n);
for k=1:n;
y(k)=0;
for i=1:n
y(k)=y(k)+a(i)*exp((-j)*2*pi*(i-1)*(k-1)*(1/n));
end
end
error=x-y;
```

```
disp(x);
disp(y);
disp(error);
subplot(3,2,1);
stem(a);
xlabel('time index n------>');
ylabel('amplitude');
title('input sequence');
subplot(3,2,2);
stem(0:n-1,abs(x));
xlabel('time index n------>');
ylabel('amplitude');
title('FFT sequence by inbuilt command');
subplot(3,2,5);
stem(0:n-1,abs(y));
xlabel('time index n------>');
ylabel('amplitude');
title('DFT by formula calculation');
subplot(3,2,6);
stem(error);
xlabel('time index n------>');
ylabel('amplitude');
title('error sequence');
% Power Spectral Density
P = x.* conj(x) / 512;
f = 1000*(0:256)/512;
figure,plot(f,P(1:257))
title('Frequency content of y');
xlabel('frequency (Hz)');
```

Evaluation of Results

- What is the difference between DTFT and DFT?
- Write any Two Properties of DFT?
- What is zero Padding and Explain the effect of it on magnitude Spectrum?
- Write the Two properties of Twiddle Factor?
- How many no. of Complex Multiplications and Additions are required to compute N-Point DFT?

Safety Precautions

References

- https://www.mathworks.com/help/matlab/math/fourier-transforms.html
- https://www.mathworks.com/help/matlab/ref/fft.html
- https://en.wikipedia.org/wiki/Fourier_transform

Experiment 6: FFT Using Decimation-in-Time (DIT) Algorithm

Objective

Write a MATLAB program to perform N-point DIT- FFT of a given signal and to plot its magnitude and phase spectrum.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

The Fast Fourier Transform (FFT) using the Decimation-In-Time (DIT) algorithm is one of the most widely used and efficient methods for computing the Discrete Fourier Transform (DFT) of a signal. It reduces the computational complexity of DFT from to $O(\mathbf{N}\log_{\mathbf{2}}\mathbf{N})$

Given a discrete signal $x[n]$ of length N, the DFT is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n].\, e^{-j(2\pi/N)kn}$$

However, computing the DFT directly requires $O(N^2)$ operations, which is computationally expensive for large sequences. The Fast Fourier Transform (FFT) is an efficient algorithm that reduces the complexity of DFT computation from $O(N^2)$ to $O(N\ log\ N)$. FFT exploits symmetries in the complex exponential terms and recursively divides the DFT computation into smaller DFTs.

DIT-FFT is a radix-2 FFT algorithm that splits the input sequence into even and odd indexed components recursively. The algorithm:

- Assumes input length N is a power of 2,
- Applies bit-reversal permutation for reordering,
- Uses butterfly operations to combine results from smaller DFTs

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
clc;
close all;
clear all;
tic;
fprintf('Date & Time:');
Date= Datestr(now);
```

```
disp(Date);
fprintf('DIT-FFT');
fprintf('\n\n');
N=input('Enter the length of the input sequence:');
for i=1:N
re(i)=input('Enter the real part of the time domain sequence:');
im(i)=input('Enter the imaginary part of the time domain
sequence:');
end
[re1,im1]=DITFFT(re,im,N);
subplot(3,2,1);
stem(0:N-1,re);
xlabel('time index n------>');
ylabel('amplitude');
title('input real sequence');
subplot(3,2,2);
stem(0:N-1,im);
xlabel('time index n------>');
ylabel('amplitude');
title('input imaginary sequence');
subplot(3,2,5);
stem(0:N-1,re1);
xlabel('frequency------>');
ylabel('amplitude');
title('output real sequence');
subplot(3,2,6);
stem(0:N-1,im1);
xlabel('frequency------>');
ylabel('amplitude');
title('output imaginary sequence');
disp(re1);
disp(im1);
```

Evaluation of Results

- Write the Block diagram of 8-Point DIT FFT & DIF DFFT radix 2 Algorithm?
- Explain using convolution the effects of taking an FFT of a sample with no windowing.
- What's the difference between FFT and DFT?
- Why do we need Fourier transform in DSP?
- What is "decimation-in-time" versus "decimation-in-frequency"?

Safety Precautions

References

- https://www.mathworks.com/help/signal/ug/discrete-fourier-transform.html
- https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform

Experiment 7: Inverse Fast Fourier Transform (IFFT)

Objective

Write a MATLAB program to perform N-point IFFT of a given sequence.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

The Inverse FFT (IFFT) converts a frequency-domain signal back into the time domain. It reconstructs the original discrete-time sequence from its frequency components.

The IFFT is mathematically defined as;

$$\boldsymbol{x[n]} = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, e^{j(\frac{2\pi}{N})kn} \qquad n = 0,1,2 \ldots., N-1$$

In MATLAB, the function ifft(y) is used to recover the original time-domain sequence from the FFT result y.

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
clc;
clear all;
close all;
n=input('enter value of n=');
x=input('enter input sequence=');
a=1:1:n;
y=fft(x,n);
disp('fft of input sequence');
disp(y);
z=ifft(y);
disp('ifft of input sequence');
disp(z);
```

Evaluation of Results

- What is the need of FFT?
- Write the Applications of FFT?
- FFT is in complex domain how to use it in real life signals optimally?

- What is the difference between FFT and IFFT?
- How many no. of Complex Multiplications and Additions are required to compute N-Point DFT using FFT?

Safety Precautions

References

- https://www.mathworks.com/help/matlab/ref/ifft.html
- https://en.wikipedia.org/wiki/Fourier_transform

Experiment 8: Design of IIR Butterworth (LP/HP) Filters

Objective

To write a MATLAB program to design Butterworth IIR LP\HP filters.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

Digital filters are systems used to suppress, pass, or modify specific frequency components of a digital signal. These filters are widely used in applications such as audio processing, communications, image processing, and biomedical signal analysis. The primary goal is to suppress unwanted frequencies and preserve the desired ones.

Digital filters are generally categorized into two types:

- **FIR (Finite Impulse Response):** Filters with finite-duration impulse responses.
- **IIR (Infinite Impulse Response):** Filters with theoretically infinite impulse responses.

These experiments (Exp-8, and Exp-9) focuses on digital filters, with an emphasis on IIR filter. A digital filter takes a discrete-time input signal $x[n]$ and produces an output signal $y[n]$ by applying a specific mathematical operation. This relationship can be described in both the time and frequency domains.

- **Transfer Function**

The behavior of a digital filter is often analyzed using its transfer function in the z-domain:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}$$

Where,

- $X(z)$, $Y(z)$ is Z-transforms of the input and output signals.
- b_k, a_k is filter coefficient.
- M, N is filter orders.

This representation is essential for design and analysis in the frequency domain.

- **Difference Equation**

In the time domain, the digital filter operates based on a difference equation, which defines how each output sample is computed:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots\ b_M x[n-M] + a_1 y[n-1] + \cdots + a_N y[n-N]$$

This equation is the discrete-time counterpart of a linear system and is directly implementable in code or hardware.

Digital filters mostly observed with Impulse Response (for time domain), Frequency Response (for frequency domain).

- **Impulse Response**

The impulse response uniquely characterizes the filter in the time domain. When the input signal is a unit impulse $\delta[n]$, the resulting output is the filter's impulse response:

$$h[n] = y[n] \qquad (\text{for } x[n] = \delta[n])$$

- FIR filters have a finite impulse response.
- IIR filters have an infinite-duration impulse response (theoretically).

- **Frequency Response**

The frequency response is essential for understanding how the filter behaves in the frequency domain, and shows how the filter affects sinusoidal components of different frequencies. It is obtained by evaluating the transfer function on the unit circle in the z-domain:

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}$$

- $|H(e^{j\omega})|$ is magnitude response
- $\angle H(e^{j\omega})$ is phase response

Digital filters can also be categorized according to frequency response characteristics into four types:

- Low-Pass Filter (LPF)
  Allows low frequencies to pass while attenuating high frequencies.
- High-Pass Filter (HPF)
  Allows high frequencies to pass while attenuating low frequencies.
- Band-Pass Filter (BPF)
  Allows a specific range of frequencies to pass and attenuates frequencies outside this range.
- Band-Stop Filter (Notch Filter)
  Attenuates a specific frequency range while passing frequencies outside that range.

Characteristics of IIR Filters

- Achieve desired frequency response with lower order than FIR filters.
- Contain feedback, which can lead to infinite-duration impulse response.
- Require careful stability analysis.
- Implemented efficiently using recursive structures.

Butterworth Filter

The Butterworth filter is one of the most widely used types of IIR filters in digital signal processing. It is known for its maximally flat frequency response in the passband, meaning that there are no ripples, and the transition from passband to stopband is smooth and monotonic. This characteristic makes Butterworth filters ideal for applications that require smooth spectral shaping without distortion in the passband. The analog low-pass Butterworth magnitude response is defined as:

$$|H(j\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}}$$

Where,

- ω_c is cutoff angular frequency.
- N is filter order.

After designing the filter on analog domain then it can be transformed into digital domain using bilinear transformation as:

$$s = \frac{2}{T} * \frac{1 - z^{-1}}{1 + z^{-1}}$$

Where,

- s is Laplace variable (analog domain)
- z is complex variable in the z-domain (digital domain)
- T is sampling period ($T = 1/f_s$)

On the other hand, digital filter can be designed directly with defining parameters:

- Passband frequency f_p
- Stopband frequency f_s
- Passband ripple r_p (in dB)
- Stopband attenuation r_s (in dB)
- Sampling frequency f_s

And then using built in functions of MATLAB "buttord()" to get minimum order and corresponding normalized cutoff frequency, and "butter()" to calculate the filter coefficients.

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
clc;
clear all;
close all;
display('enter the iir filter design specifications');
rp=input('enter the pass band ripple:');
rs=input('enter the stop band ripple:');
wp=input('enter the pass band freq:');
ws=input('enter the stop band freq:');
fs=input('enter the sampling freq:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
c=input('enter choice filter 1.lpf 2.hpf /n');
if(c==1)
display('frequency response of IIR lpf is:');
[b,a]=butter(n,wn,'low');
end
if(c==2)
```

```
display('freq response of IIR hpf IS:');
[b,a]=butter(n,wn,'high');
end
w=0:0.01:pi;
h=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure;
subplot(2,1,1);
plot(w/pi,m);
title('mignitude response of IIR filter is:');
xlabel('(a)normalized frequency-->');
ylabel('gain in db-->');
subplot(2,1,2);
plot(w/pi,an);
title('phase response of IIR filter is;');
xlabel('(b) normalized frequency-->');
ylabel('phase in radians-->');
```

Evaluation of Results

- What do you mean by cut-off frequency?
- Give the differences between analog and digital filters?
- What is the difference between type 1 and type 2 filter structures?
- What is the role of delay element in filter design?
- List the Differences between Butterworth and chebyshev filters?

Safety Precautions

References

- https://en.wikipedia.org/wiki/Infinite_impulse_response
- https://en.wikipedia.org/wiki/Digital_filter
- https://en.wikipedia.org/wiki/Butterworth_filter
- https://www.mathworks.com/help/signal/ug/iir-filter-design.html
- https://www.mathworks.com/help/signal/ref/buttord.html
- https://www.mathworks.com/help/signal/ref/butter.html

Experiment 9: Design of IIR Chebyshev (LP/HP) Filters

Objective

To write a MATLAB program to design Chebyshev IIR LP\HP filters.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

Digital filters are systems used to suppress, pass, or modify specific frequency components of a digital signal. These filters are widely used in applications such as audio processing, communications, image processing, and biomedical signal analysis. The primary goal is to suppress unwanted frequencies and preserve the desired ones.

Digital filters are generally categorized into two types:

- **FIR (Finite Impulse Response):** Filters with finite-duration impulse responses.
- **IIR (Infinite Impulse Response):** Filters with theoretically infinite impulse responses.

These experiments (Exp-8, and Exp-9) focuses on digital filters, with an emphasis on IIR filter. A digital filter takes a discrete-time input signal $x[n]$ and produces an output signal $y[n]$ by applying a specific mathematical operation. This relationship can be described in both the time and frequency domains.

- **Transfer Function**

The behavior of a digital filter is often analyzed using its transfer function in the z-domain:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}$$

Where,

- $X(z)$, $Y(z)$ is Z-transforms of the input and output signals.
- b_k, a_k is filter coefficient.
- M, N is filter orders.

This representation is essential for design and analysis in the frequency domain.

- **Difference Equation**

In the time domain, the digital filter operates based on a difference equation, which defines how each output sample is computed:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots\ b_M x[n-M] + a_1 y[n-1] + \cdots + a_N y[n-N]$$

This equation is the discrete-time counterpart of a linear system and is directly implementable in code or hardware.

Digital filters mostly observed with Impulse Response (for time domain), Frequency Response (for frequency domain).

- **Impulse Response**

The impulse response uniquely characterizes the filter in the time domain. When the input signal is a unit impulse $\delta[n]$, the resulting output is the filter's impulse response:

$$h[n] = y[n] \qquad (\text{for } x[n] = \delta[n])$$

- FIR filters have a finite impulse response.
- IIR filters have an infinite-duration impulse response (theoretically).

- **Frequency Response**

The frequency response is essential for understanding how the filter behaves in the frequency domain, and shows how the filter affects sinusoidal components of different frequencies. It is obtained by evaluating the transfer function on the unit circle in the z-domain:

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}$$

- $|H(e^{j\omega})|$ is magnitude response
- $\angle H(e^{j\omega})$ is phase response

Digital filters can also be categorized according to frequency response characteristics into four types:

- Low-Pass Filter (LPF)
  Allows low frequencies to pass while attenuating high frequencies.
- High-Pass Filter (HPF)
  Allows high frequencies to pass while attenuating low frequencies.
- Band-Pass Filter (BPF)
  Allows a specific range of frequencies to pass and attenuates frequencies outside this range.
- Band-Stop Filter (Notch Filter)
  Attenuates a specific frequency range while passing frequencies outside that range.

Characteristics of IIR Filters

- Achieve desired frequency response with lower order than FIR filters.
- Contain feedback, which can lead to infinite-duration impulse response.
- Require careful stability analysis.
- Implemented efficiently using recursive structures.

Chebyshev Filter

The Chebyshev filter is a type of IIR (Infinite Impulse Response) filter that offers a sharper transition between the passband and stopband compared to the Butterworth filter. This is achieved by allowing ripples in either the passband (Type I) or stopband (Type II), depending on the filter type used.

Chebyshev filters are useful in applications where a fast roll-off is desired, and some deviation (ripple) in magnitude response is acceptable in either passband or stopband. According to the ripples being at passband or stopband Chebyshev Filters categorized in to types:

- Chebyshev Type I
- Chebyshev Type II

<u>*Chebyshev Type-I*</u>

Allows ripple behavior in the passband, while monotonic in the stopband, and more commonly used in practice. The magnitude response of Chebyshev Type-I is as given:

$$|H(j\omega)|^2 = \frac{1}{1 + e^2 T_n^2\left(\frac{\omega}{\omega_c}\right)}$$

Where,

- w_c is cutoff ffrequency
- e is ripple factor (related to passband ripple in dB)
- $T_n(\,)$ is nth-order Chebyshev polynomial

The phase response of Chebyshev Type-I can be formulated as all the other digital filters as given:

$$\angle H(j\omega) = \tan^{-1}\left(\frac{Im[H(j\omega)]}{Re[H(j\omega)]}\right)$$

Chebyshev Type-II

Allows ripple behavior in the stopband, while monotonic in the passband. The magnitude response of Chebyshev Type-II is as given:

$$|H(j\omega)|^2 = \frac{1}{1 + \frac{1}{e^2 T_n^2\left(\frac{\omega_c}{\omega}\right)}}$$

The phase response of Chebyshev Type-II can be formulated as all the other digital filters.

Filter Design Using MATLAB

Chebyshev Filter can be designed using commands:

- "cheb1ord(w1,w2,rp,rs)" for defining order of filter Type-I
- "cheb1(n,rs,wn,'low')" for defining low-pass filter coefficients
- "cheb2ord(w1,w2,rp,rs)" for defining order of filter Type-II
- "cheb2(n,rs,wn,'low')" for defining filter coefficients,

Check code block for the parameters.

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
% Program for the design of Chebyshev Type-1 low-pass filter
clc;
close all;clear all;
format long
rp=0.34;    %input('enter the passband ripple...');
rs=47;      %input('enter the stopband ripple...');
wp=1400;    %input('enter the passband freq...');
ws=1600;    %input('enter the stopband freq...');
fs=10000;   %input('enter the sampling freq...');
w1=2*wp/fs;
```

```
w2=2*ws/fs;
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rs,wn,'low');
w=0:.01:pi;
[h,f]=freqz(b,a,fs,fs);
m=20*log10(abs(h));
an=angle(h);
% f=om*fs/(2*pi);
subplot(2,1,1);
plot(f,m);
ylabel('Gain in dB --.');
xlabel('(a) Normalised frequency --.');
subplot(2,1,2);
plot(f,an);
xlabel('(b) Normalised frequency --.');
ylabel('Phase in radians --.');
```

Evaluation of Results

- What is the difference between type 1 and type 2 filter structures?
- IIR Filters are non linear phase filters. Why?
- IIR filters are not always stable. Why?
- Write the realization Techniques for IIR Filter?
- Compare all realization techniques vailable for IIR Filter?

Safety Precautions

References

- https://en.wikipedia.org/wiki/Infinite_impulse_response
- https://en.wikipedia.org/wiki/Digital_filter
- https://en.wikipedia.org/wiki/Chebyshev_filter
- https://www.mathworks.com/help/signal/ug/iir-filter-design.html
- https://www.mathworks.com/help/signal/ref/cheb1ord.html
- https://www.mathworks.com/help/signal/ref/cheby1.html

Experiment 10: Design of FIR (LP/HP) Using Windowing Technique

Objective

To write a MATLAB program to design FIR with Low pass filter and High Pass filter using any three Windowing techniques.

Required Equipment

- PC
- MATLAB Software

Theoretical Background

FIR (Finite Impulse Response) filters are linear time-invariant digital filters characterized by a finite-duration impulse response. The general form of an FIR filter is:

$$y[n] = \sum_{k=0}^{N} b_k \, x[n-k]$$

Where:

- $y[n]$ is output signal
- $x[n]$ is input signal
- b_k is filter coefficients (impulse response)
- N is filter order

FIR filters are always stable and can be designed to have exactly linear phase.

Windowing Technique

Ideal filters (e.g., ideal low-pass) have infinite-length impulse responses:

$$h_d[n] = \frac{\sin\big(\omega_c(n-M)\big)}{\pi(n-M)}$$

Where, $M = \frac{N}{2}$. This ideal impulse response must be truncated using a window function $w[n]$:

$$h[n] = h_d[n]w[n]$$

Common windows:

- Rectengular: $w[n] = 1$
- Hamming: $w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N}\right)$
- Kaiser: flexible with parameter β

Design Steps

- Specify f_p, f_s, A_p, A_s, $f_{sampling}$
- Estimate filter order N using transition bandwidth and desired stopband attenuation:

$$N \cong \frac{-20\log_{10}\big(\sqrt{\delta_p\delta_s}\big) - 13}{14.6\Delta f / f_s}$$

- Generate $h_d[n]$, apply window $w[n]$, compte $h[n]$
- Use “fir1(N, wc, window)” in MATLAB

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
%FIR Low Pass/High pass filter design using Rectangular/Hamming/Kaiser window
clc; clear all; close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter passband freq');
ws=input('enter stopband freq');
fs=input('enter sampling freq ');
beta=input('enter beta value');
w1=2*wp/fs;
w2=2*ws/fs;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(ws-wp)/fs;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
n1=n; n=n-1;
end
c=input('enter your choice of window function 1. rectangular
2. Hamming 3.kaiser: \n ');
if(c==1)
y=rectwin(n1);
disp('Rectangular window filter response');
end
if (c==2)
y=hamming(n1);
disp('Hamming window filter response');
end
if(c==3)
y=kaiser(n1,beta);
disp('kaiser window filter response');
end
ch=input('give type of filter 1:LPF,2:HPF');
switch ch
case 1
b=fir1(n,w1,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
plot(o/pi,m);
title('LPF');
xlabel('(a) Normalized frequency-->');
ylabel('Gain in dB-->');
case 2
b=fir1(n,w1,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
plot(o/pi,m);
```

```
title('HPF');
xlabel('(b) Normalized frequency-->');
ylabel('Gain in dB-->');
end
```

Evaluation of Results

- What is filter?
- What is FIR and IIR filter define, and distinguish between these two?
- What is window method? How you will design an FIR filter using window method?
- What are low-pass and band-pass filter and what is the difference between these two?
- What is the matlab command for Hamming window? Explain.

Safety Precautions

References

- https://en.wikipedia.org/wiki/Finite_impulse_response
- https://www.mathworks.com/help/signal/ref/rectwin.html
- https://www.mathworks.com/help/signal/ref/hamming.html
- https://www.mathworks.com/help/signal/ref/kaiser.html
- https://www.mathworks.com/help/signal/ref/fir1.html

Experiment 11: Generation of FIR Low Pass / High Pass Filter Coefficients

Objective

To generate FIR filter (LP/HP) coefficients using windowing techniques

- Using Rectangular Window
- Using Triangular Window
- Using Keiser Window

Required Equipment

- PC
- MATLAB Software

Theoretical Background

Steps to design linear phase FIR filter using windowing method:

1. Clearly specify the filter specificatons
    a. Order of filter (**N**)
    b. Sampling rate ($\mathbf{F_s}$)
    c. Cutoff frequency ($\mathbf{f_c}$)
2. Compute the cutoff frequency ($\mathbf{w_c}$)

$$\omega_c = \frac{2\pi f_c}{F_s}$$

3. Compute the desired impulse response h(n) using particular window.
    a. Lowpass Filter with Cutoff frequency ($\mathbf{w_c}$)
        i. Coefficients of Linear Phase Filter

$$h_d(n) = \frac{\omega_c}{\pi}, \qquad for\ n = \alpha$$

$$h_d(n) = \frac{\sin(\omega_c(n-\alpha))}{\pi(n-\alpha)}, \qquad for\ n \neq \alpha$$

Where $\alpha = \frac{N-1}{2}$

        ii. Window Sequence

$$h_d(n) = \frac{\omega_c}{\pi}, \qquad for\ n = \alpha$$

$$h_d(n) = \frac{\sin(\omega_c(n-\alpha))}{\pi(n-\alpha)}, \qquad for\ n \neq \alpha$$

Where $\alpha = \frac{N-1}{2}$

1. For Rectangular Window

$$W(n) = 1, \qquad for\ 0 < n < (N-1)$$

$$W(n) = 0, \qquad otherwise$$

2. For Triangular Window

$$W(n) = \frac{2n}{N-1}, \qquad for\ 0 < n < \frac{(N-1)}{2}$$

$$W(n) = 2 - \frac{2n}{N-1}, \qquad for\ \frac{(N-1)}{2} < n < (N-1)$$

3. For Keiser Window

$$W(n) = \frac{I_0\left[\beta\left(\left(\frac{(N-1)^2}{4}\right)-\left[n-\frac{N-1}{2}\right]^2\right)^{\frac{1}{2}}\right]}{I_0\left[\frac{\beta(N-1)}{2}\right]}, \qquad for\ 0 \le n \le (N-1)$$

$$W(n) = 0, \qquad otherwise$$

Where β is the variable parameter whose choice control the tradeoff between side lobe amplitude and side lobe width.

iii. The Impulse Response

$$h(n) = h_d(n) * w(n)$$

b. Highpass Filter with Cutoff frequency ($\mathbf{w_c}$)

i. Coefficients of Linear Phase Filter

$$h_d(n) = 1 - \frac{\omega_c}{\pi}, \qquad for\ n = \alpha$$

$$h_d(n) = \frac{1}{\pi(n-\alpha)}\left[\sin\left(\pi(n-\alpha)\right) - \sin\left(w_c(n-\alpha)\right)\right], \qquad for\ n \neq \alpha$$

Where $\alpha = \frac{N-1}{2}$

ii. Window sequence

1. For Rectangular Window

$$W(n) = 1, \qquad for\ 0 < n < \frac{N-1}{2}$$

$$W(n) = 0, \qquad otherwise$$

2. For Triangular Window

$$W(n) = \frac{2n}{N-1}, \qquad for\ 0 < n < \frac{(N-1)}{2}$$

$$W(n) = 2 - \frac{2n}{N-1}, \qquad for\ \frac{(N-1)}{2} < n < (N-1)$$

3. For Keiser Window

$$W(n) = \frac{I_0\left[\beta\left(\left(\frac{(N-1)^2}{4}\right)-\left[n-\frac{N-1}{2}\right]^2\right)^{\frac{1}{2}}\right]}{I_0\left[\frac{\beta(N-1)}{2}\right]}, \qquad for\ 0 \le n \le (N-1)$$

$$W(n) = 0, \qquad otherwise$$

Where β is the variable parameter whose choice control the tradeoff between side lobe amplitude and side lobe width.

iii. The Impulse Response

$$h(n) = h_d(n) * w(n)$$

4. Concolve input sequence with truncated impulse response

$$y(n) = x(n) * h(n)$$

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

MATLAB Program to generate 'FIR Filter-Low Pass' Coefficients using FIR1

```
% FIR Low pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000
order = 30;
cf=[500/4000,1000/4000,1500/4000]; cf--> contains set of cut-off
frequencies[Wc ]
% cutoff frequency - 500
b_rect1=fir1(order,cf(1),boxcar(31)); Rectangular
b_tri1=fir1(order,cf(1),bartlett(31)); Triangular
b_kai1=fir1(order,cf(1),kaiser(31,8)); Kaisar [Where 8-->Beta Co-efficient]
% cutoff frequency - 1000
b_rect2=fir1(order,cf(2),boxcar(31));
b_tri2=fir1(order,cf(2),bartlett(31));
b_kai2=fir1(order,cf(2),kaiser(31,8));
% cutoff frequency - 1500
b_rect3=fir1(order,cf(3),boxcar(31));
b_tri3=fir1(order,cf(3),bartlett(31));
b_kai3=fir1(order,cf(3),kaiser(31,8));
fid=fopen('FIR_lowpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);
winopen('FIR_highpass_rectangular.txt');
```

MATLAB Program to generate 'FIR Filter-High Pass' Coefficients using FIR1

```
% FIR High pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000
order = 30;
cf=[400/4000,800/4000,1200/4000]; ;cf--> contains set of cut-off
frequencies[Wc]
% cutoff frequency - 400
b_rect1=fir1(order,cf(1),'high',boxcar(31));
b_tri1=fir1(order,cf(1),'high',bartlett(31));
b_kai1=fir1(order,cf(1),'high',kaiser(31,8)); Where Kaiser(31,8)--> '8'defines
the value of 'beta'.
% cutoff frequency - 800
b_rect2=fir1(order,cf(2),'high',boxcar(31));
b_tri2=fir1(order,cf(2),'high',bartlett(31));
b_kai2=fir1(order,cf(2),'high',kaiser(31,8));
```

```
% cutoff frequency - 1200
b_rect3=fir1(order,cf(3),'high',boxcar(31));
b_tri3=fir1(order,cf(3),'high',bartlett(31));
b_kai3=fir1(order,cf(3),'high',kaiser(31,8));
fid=fopen('FIR_highpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);
winopen('FIR_highpass_rectangular.txt');
```

Evaluation of Results

- What are windowing techniques in FIR filters?
- Which window is best for FIR filter?
- Which technique is not used in design os FIR FILETR?
- Why FIR filters are always stable?
- Why windows are necessary in FIR filters?

Safety Precautions

References

- https://en.wikipedia.org/wiki/Finite_impulse_response
- https://www.mathworks.com/help/signal/ref/rectwin.html
- https://www.mathworks.com/help/signal/ref/hamming.html
- https://www.mathworks.com/help/signal/ref/kaiser.html
- https://www.mathworks.com/help/signal/ref/fir1.html

Experiment 12: Generation of IIR Butterworth/Chebyshev Filter Coefficients

Objective

To generate IIR filter Butterworth/Chebyshev coefficients

Required Equipment

- PC
- MATLAB Software

Theoretical Background

IIR filters are recursive digital filters with both feedforward and feedback terms:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k]$$

These are derived from analog filter prototypes via bilinear transformation:

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$$

Butterworth Filters

Maximally flat in passband:

$$|H(j\omega)|^2 = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2N}}$$

Chebyshev Type I Filters

Equiripple in passband:

$$|H(j\omega)| = \frac{1}{\sqrt{1+e^2 T_N^2\left(\frac{\omega}{\omega_c}\right)}}$$

Chebyshev Type II Filters

Maximally flat in passband, ripple in stopband

$$|H(j\omega)|^2 = \frac{1}{1+\frac{1}{e^2 T_n^2\left(\frac{\omega_c}{\omega}\right)}}$$

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
% IIR Low pass Butterworth and Chebyshev filters
% sampling rate - 24000
```

```
order = 2;
cf=[2500/12000,8000/12000,1600/12000];
% cutoff frequency - 2500
[num_bw1,den_bw1]=butter(order,cf(1));
[num_cb1,den_cb1]=cheby1(order,3,cf(1));
% cutoff frequency - 8000
[num_bw2,den_bw2]=butter(order,cf(2));
[num_cb2,den_cb2]=cheby1(order,3,cf(2));
fid=fopen('IIR_LP_BW.txt','wt');
fprintf(fid,'\t\t----------Pass band range: 0-2500Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Monotonic-----\n\n\');
fprintf(fid,'\n float num_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw1);
fprintf(fid,'\nfloat den_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw1);
fprintf(fid,'\n\n\n\t\t----------Pass band range: 0-8000Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Monotonic-----\n\n');
fprintf(fid,'\nfloat num_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw2);
fprintf(fid,'\nfloat den_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw2);
fclose(fid);
winopen('IIR_LP_BW.txt');
fid=fopen('IIR_LP_CHEB Type1.txt','wt');
fprintf(fid,'\t\t----------Pass band range: 2500Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Rippled (3dB) -----\n\n\');
fprintf(fid,'\nfloat num_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb1);
fprintf(fid,'\nfloat den_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb1);
fprintf(fid,'\n\n\n\t\t----------Pass band range: 8000Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Rippled (3dB)-----\n\n');
fprintf(fid,'\nfloat num_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb2);
fprintf(fid,'\nfloat den_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb2);
fclose(fid);
winopen('IIR_LP_CHEB Type1.txt');
%%%%%%%%%%%%%%%%%%
figure(1);
[h,w]=freqz(num_bw1,den_bw1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb1,den_cb1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response of Low pass IIR filters (Fc=2500Hz)');
figure(2);
[h,w]=freqz(num_bw2,den_bw2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
```

```
[h,w]=freqz(num_cb2,den_cb2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1 (Ripple: 3dB)');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response in the passband');
axis([0 12000 -20 20]);
```

Evaluation of Results

- What is bilinear transformation in IIIR filter design?
- Which method is not suitable for designing IIR filters?
- What are the steps involved to design digital filter using bilinear transformations?
- What are the limitations of impulse invariance method?
- What are the differences between IIR and FIR filters?

Safety Precautions

References

- https://en.wikipedia.org/wiki/Infinite_impulse_response
- https://en.wikipedia.org/wiki/Digital_filter
- https://en.wikipedia.org/wiki/Butterworth_filter
- https://www.mathworks.com/help/signal/ug/iir-filter-design.html
- https://www.mathworks.com/help/signal/ref/buttord.html
- https://www.mathworks.com/help/signal/ref/butter.html
- https://en.wikipedia.org/wiki/Chebyshev_filter
- https://www.mathworks.com/help/signal/ref/cheb1ord.html
- https://www.mathworks.com/help/signal/ref/cheby1.html

Experiment 13: Decimation and Interpolation

Objective

To write a MATLAB program to decimation and interpolation of a give sequence

Required Equipment

- PC
- MATLAB Software

Theoretical Background

Decimation (Downsampling)

Decimation reduces the sampling rate by an integer factor M. The downsampled signal is:

$$y[n] = x[nM]$$

Requires pre-filtering to prevent aliasing

Interpolation (Upsampling)

Interpolation means that generation a new samples betweens the existing ones. There are multiple ways of interpolation techniques, e.g zero-padding, Taylor Series approach. Below is a simple yet effective one with application of low-pass filter after adding zeroes to the existing samples, yet it has to be used carefully couse zeroes might effect more than the signal itself.

Interpolation increases the sampling rate by an integer factor L:

$$x_{\uparrow}[n] = \begin{cases} x[n/L], & n = 0 \ mod \ L \\ 0, & otherwise \end{cases}$$

Followed by a low-pass filter to smooth the inserted zeros.

Procedure

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

Simulation

```
% PROGRAM DECIMATION:
clc;
close all;
clear all;
M = input('enter Down-sampling factor : ');
N = input('enter number of samples :');
n = 0:N-1;
x = sin(2*pi*0.043*n) + sin(2*pi*0.031*n);
y = decimate(x,M,'fir');
subplot(2,1,1);
stem(n,x(1:N));
```

```
title('Input Sequence');
xlabel('Time index n');
ylabel('Amplitude');
subplot(2,1,2);
m = 0:(N/M)-1
title('Output Sequence');
xlabel('Time index n');ylabel('Amplitude');
```

%PROGRAM INTERPOLATION

```
clc;
clear all;
close all;
L=input('enter the upsampling factor');
N=input('enter the length of the input signal'); % Length should be greater
than 8
f1=input('enter the frequency of first sinusodal');
f2=input('enter the frequency of second sinusodal');
n=0:N-1;
x=sin(2*pi*f1*n)+sin(2*pi*f2*n);
y=interp(x,L);
subplot(2,1,1)
stem(n,x(1:N))
title('input sequence');
xlabel('time(n)');
ylabel('amplitude');
subplot(2,1,2)
m=0:N*L-1;
stem(m,y(1:N*L))
title('output sequence ');
xlabel('time(n)');
ylabel('amplitude');
```

Evaluation of Results

- Explain about multi rate digital signal processing.
- List the Applications of multi rate digital signal processing.
- Define interpolation.
- Define decimation.
- Define aliasing.

Safety Precautions

References

- https://en.wikipedia.org/wiki/Downsampling_(signal_processing)
- https://www.mathworks.com/help/signal/ref/decimate.html
- https://en.wikipedia.org/wiki/Interpolation
- https://www.mathworks.com/help/signal/ref/interp.html