MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 1. PROGRAMMING SIMPLE FUNCTIONS IN LABVIEW ON A PC

I. Introduction

This experiment is about the construction of a LabVIEW project on a personal computer. Since the project VI's are run on PC, the implementation may not be real time. In fact, the computer operating system is not real-time and hence screen updates, Ethernet port services, etc. take precedence during the execution. While today's PC's have powerful CPU's, this fact can be observed easily for critical real-time processing tasks.

In this experiment, two VI's are constructed under the project both of which are run on the PC. This is achieved by placing the VI's under the "My Computer" item. In the Section III, the steps of constructing the project and the details of the first VI are described. The second VI should be designed individually by considering the first VI as an example as the experiment work.

II. Preliminary Work

1)

a) Find the minimum sampling rate in both rad/s (Ω) and Hz (f) for the following band-limited analog signal whose CTFT is given in Fig. 1 such that it can be reconstructed from its samples.

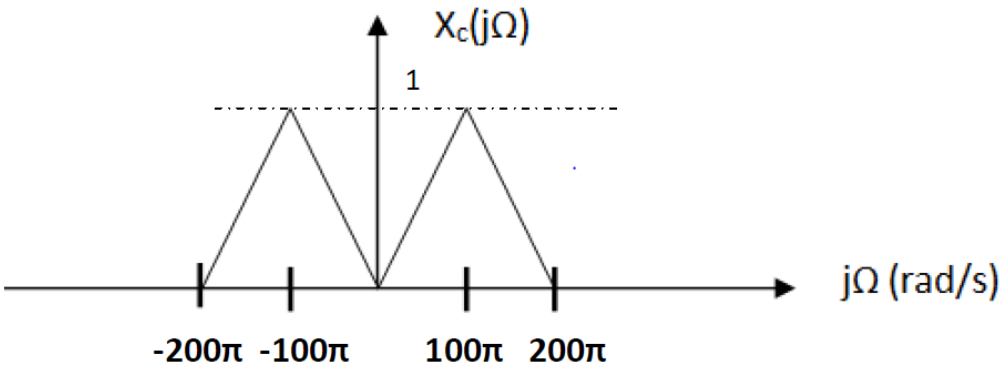


Fig. 1. CTFT of the band-limited analog signal in question 1

b) The analog signal given in Fig. 1 is sampled with a sampling period T=1/150 s. Plot the **discrete time Fourier transform (DTFT)** of the sampled sequence by indicating all necessary information on the x- and y-axes. Determine whether the aliasing occurs or not. If aliasing occurs, specify the frequency region where aliasing occurs.

c) The analog signal given in Fig. 1 is sampled with a sampling period T=1/500 s. Determine whether the aliasing occurs or not. What is the highest frequency in the sampled signal? Let $\Omega_a = 100\pi$rad/s be an analog frequency value. Which is the corresponding discrete frequency ω_a in DTFT?

2) DFT is implemented using **Fast Fourier Transform (FFT)** algorithm in practice and MATLAB environment. In this part, you will explore **"fft"** and **"ifft"** functions in MATLAB for taking DFT and inverse DFT, respectively. You can write **"help fft"** and **"help ifft"** in the **Command Window of MATLAB** to find out these functions.

- fft(x,N) is the N-point fft of x, padded with zeros if x has less than N points and truncated if it has more.
- ifft(x,N) is the N-point inverse fft of x.
- If N is not given, such as fft(x), the fft and ifft are implemented by taking DFT and inverse DFT size as the length of input sequence.
- You are first required to define an input sequence in MATLAB, which consists of the **first 6 digits** of your student ID number. For example, if your student ID is 1234567, the input of the MATLAB code is given as follows:

  x= [1 2 3 4 5 6];

- Take N=6-point fft of x and call it "y". Plot the magnitude and phase of y using ***"stem"*** command in MATLAB. Attach these plots to your preliminary work.
- Is there some kind of symmetry in the plots? If there is a symmetry, explain the reason considering input sequence x.
- Then, take N=9-point fft of x and call it "z". Compare z with y. Are they the same?
- Now take N=9-point ifft of z and compare the result with x.
- Take N=6-point ifft of z and compare the result with x. Why is it different than x?
- Take N=4-point fft of x and call it "v". Then, take ifft of v and compare the result with x.

Attach your MATLAB codes to your preliminary work.

3) In this part, you are required to implement convolution of two discrete time sequences in LabVIEW environment. The front panel is seen in Fig. 2.

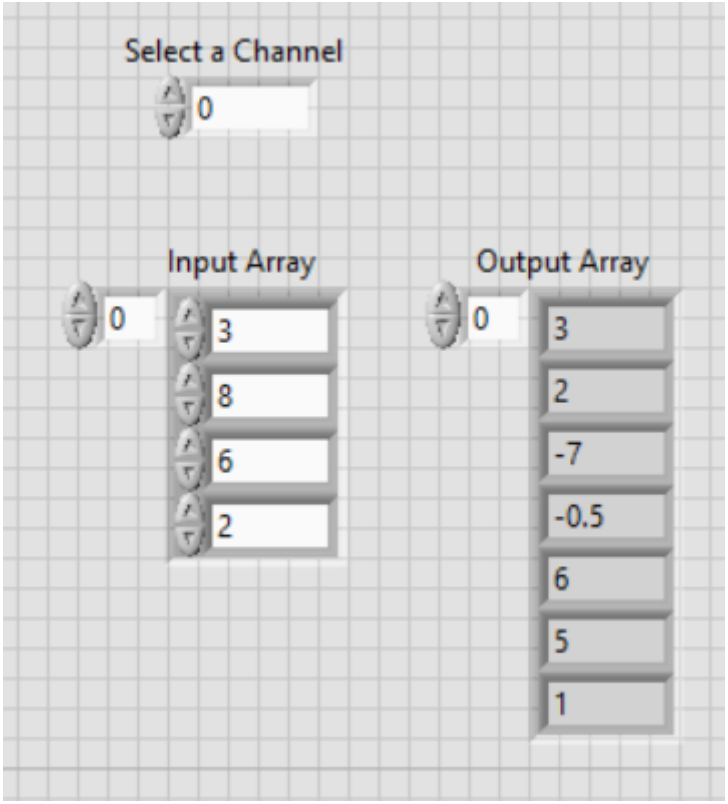


Fig. 2. Front Panel of the Programming Task of First Question

The input sequence is entered manually as shown in Fig. 2 through an Array Numeric control. The convolution of this input sequence with two different channels (impulse responses) will be implemented. If **"Select a channel"** input is entered as **0**, then the impulse response is given by,

$h_0[n]=\delta[n]-2\delta[n-1]+\delta[n-2]+1/2\delta[n-3]$.

If **"Select a channel"** input is entered as **1**, then the impulse response is given by,

$h_1[n]=\delta[n]+2/3*\delta[n-1]+1/3*\delta[n-2]-5/3*\delta[n-3]$.

The default channel can be any of these two channels if another number is entered as **"Select a channel"**.
Assume that the first element of the input sequence corresponds to n=0, that is, the example input sequence in Fig. 2 is

$x[n]= 3*\delta[n]+8*\delta[n-1]+6*\delta[n-2]+2*\delta[n-3]$.

- In order to implement convolution, you can use shift registers or feedback nodes. In both cases, please **do not forget** to take **initial values of registers as 0**.
- Remember that the length of the convolved sequence of two sequences is (N+P-1), if the length of input sequence and impulse response is N and P, respectively. Since the

length of the channel impulse responses in this task is P=4, we expect the size of the indicator array which shows the output sequence in Fig. 2 is (N+3). As a hint, you can append three zeros at the end of the input sequence. But, **you are required to do it inside the code (i.e., block diagram)**, not by entering three additional zeros. That means if the input sequence is [3 8 6 2] as in Fig. 2, we only enter 3,8,6,2 values without additional zeros.

- You are required to ***attach*** the ***screenshots of the front panel*** of your VI with **two different channel selections** to your preliminary work as shown in Fig. 3. You should also attach the **screenshot of your block diagram**.
- The **input should be given** as the **last 4 digits** of your **student ID** number. For example, if your ID is 1753862, the input should be [3 8 6 2] as shown in Fig. 3.

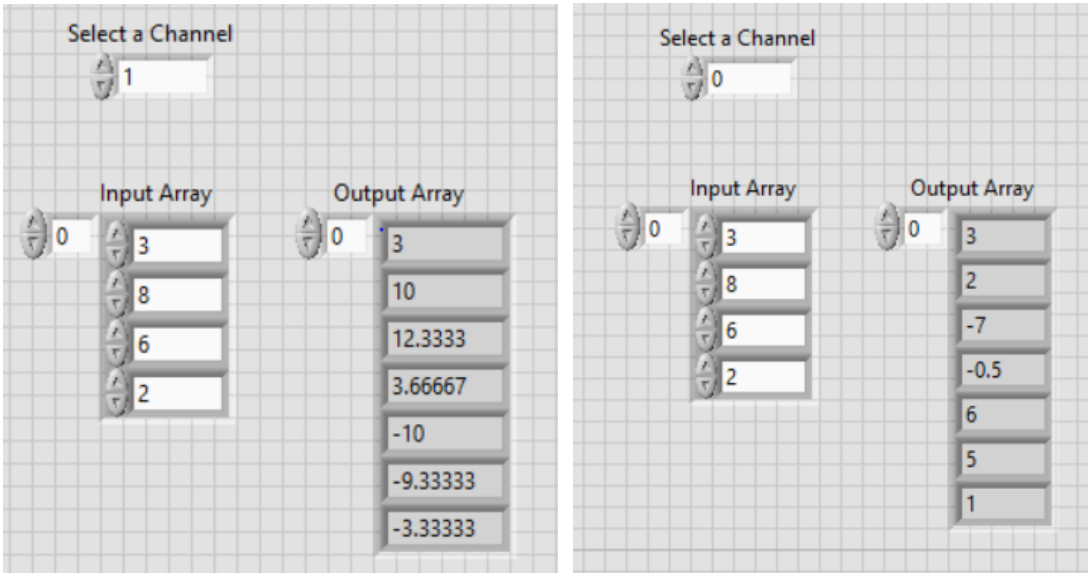


Fig. 3. Screenshots of Front Panel for two channel selections

III. Experimental Work

1. Project Generation and the Run_Sine.vi

Run LabVIEW and select Blank Project from the templates as shown in the Fig. 4.

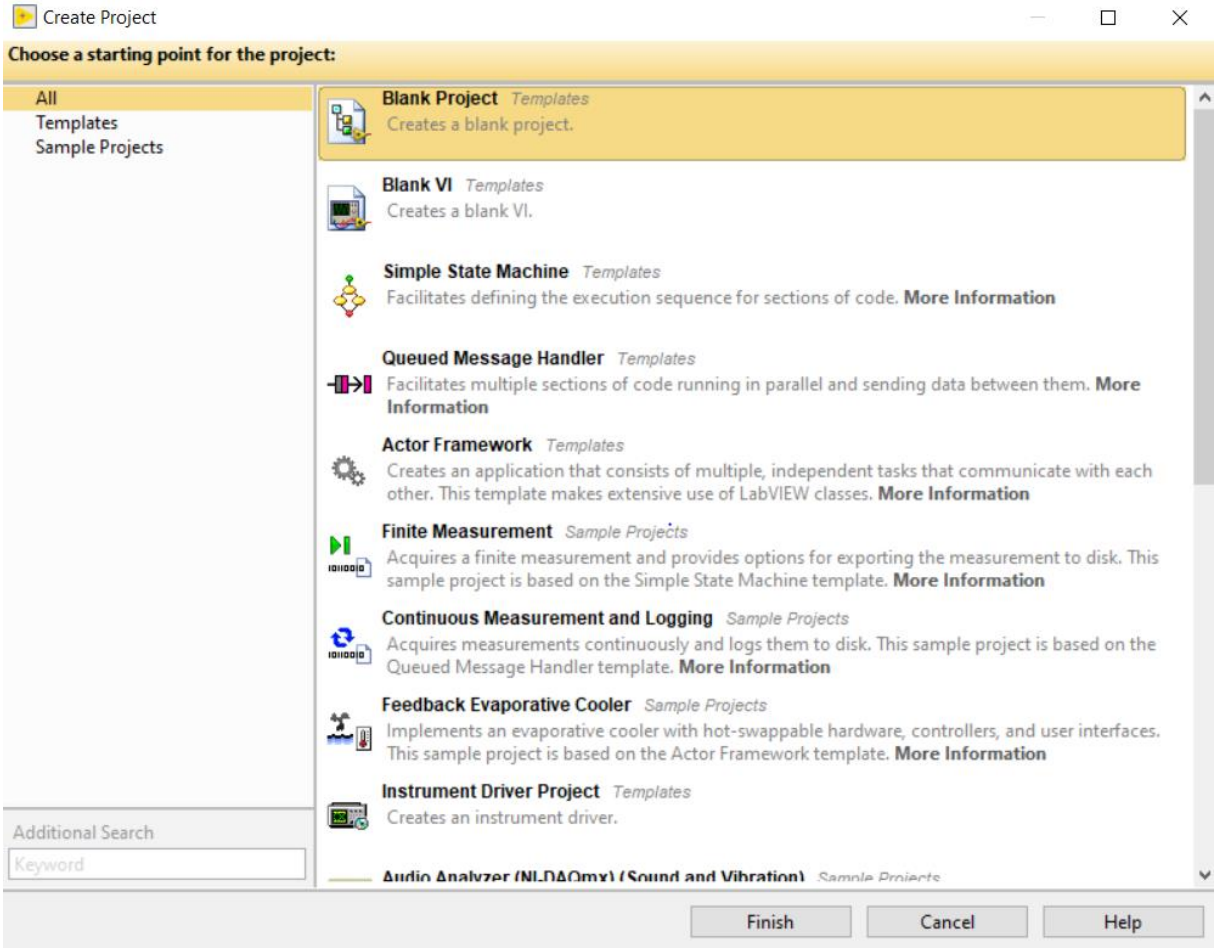


Fig. 4. Generation of a LabVIEW project

Click Finish and a new screen comes up as shown in Fig. 5.

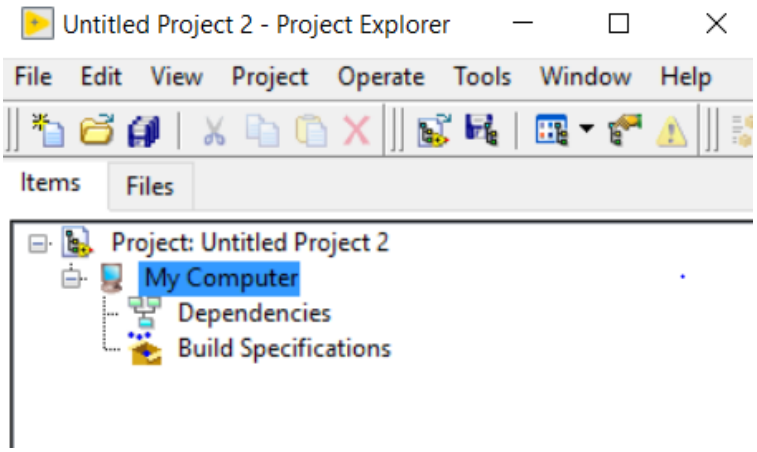


Fig. 5. Project screen

Next, you can create your own VI that is run on PC by **right clicking on My Computer and selecting VI** as shown in Fig. 6.

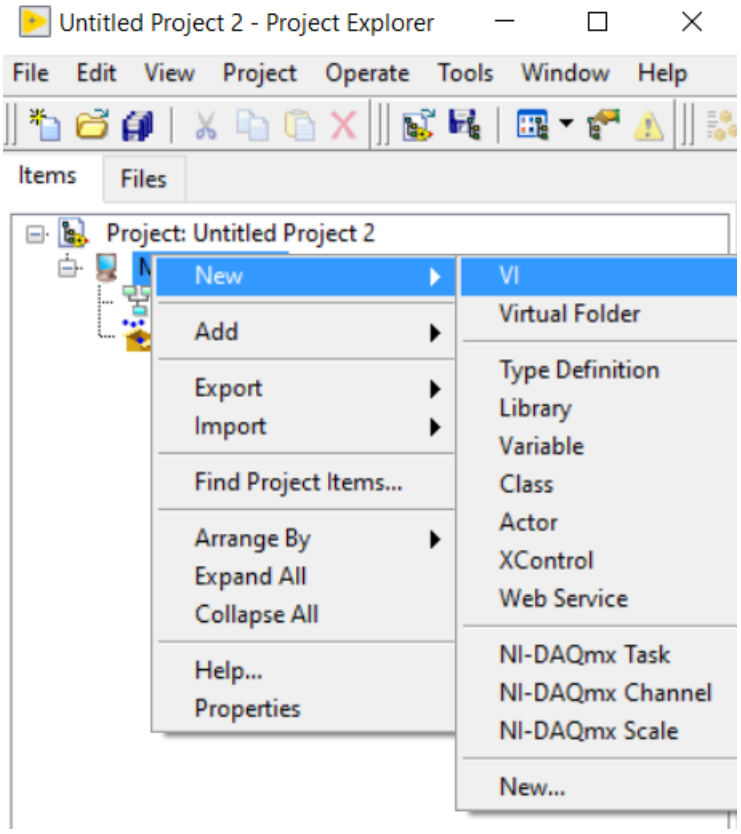


Fig. 6. Generating a VI under My Computer

At this point, two items pop-up. One is the **Front Panel** of the VI, the other is **Block Diagram**. Front Panel is used to add certain gadgets, buttons for user interface and control. Most of the programming is implemented on the Block Diagram. You can save your VI and give it a name such as **Run_Sine.vi**. In the Block Diagram, right click and select while loop from the **Functions menu under Structures** as shown in Fig. 7.

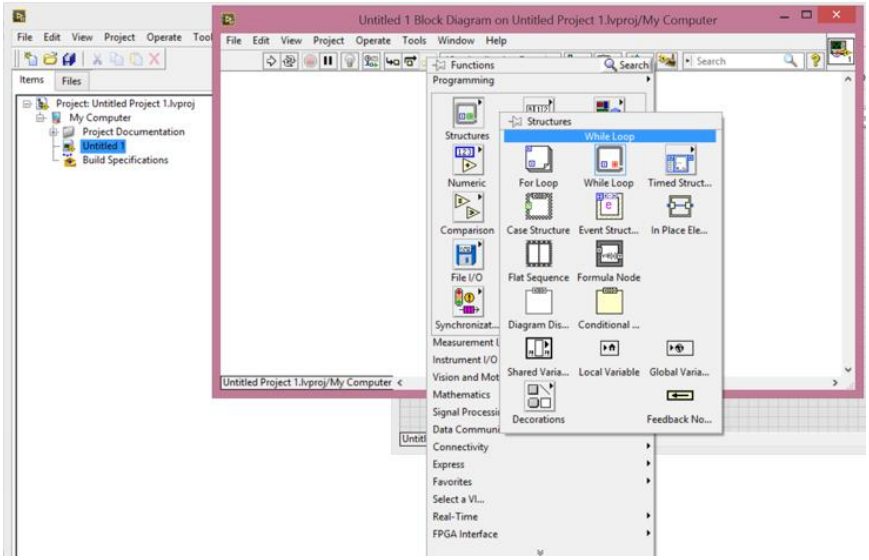


Fig. 7. Functions menu and the while loop function

Create a stop button by right clicking the red button and selecting **Create → Control** as shown in Fig. 8.

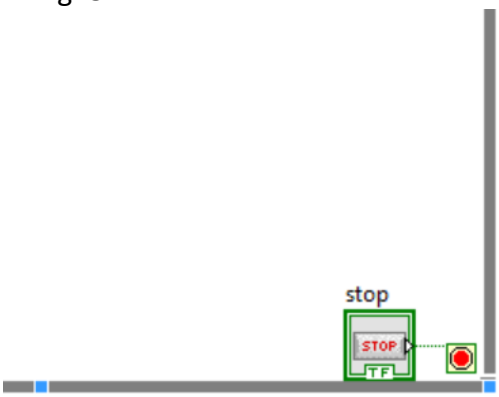


Fig. 8. While Loop and the control button

Now the while loop is ready but functionality should be added inside. The Run_Sine.vi is designed to generate a **sinusoid corrupted by AWGN and plot its DFT magnitude and phase**. There is a special **express VI** for this purpose in LabVIEW, **"Simulate Signal"**, which you can select as shown in Fig. 9 (**Functions → Signal Processing → Waveform Generation → Simulate Signal**).

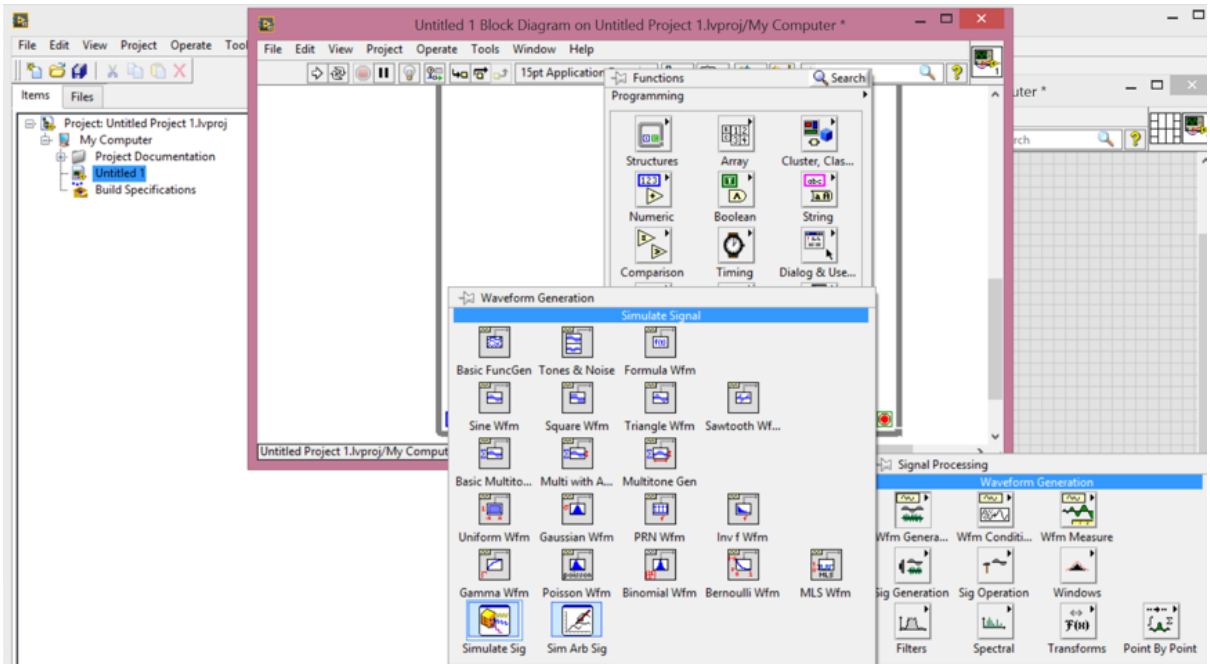


Fig. 9. Selecting the Simulate Signal function

When you place this function on the Block Diagram, you can **configure** its inputs to set frequency, phase offset, noise variance, etc. We will enter **frequency, amplitude, phase offset and noise standard deviation** as inputs. In the Configuration window, click on **"Add noise"** and select **Gaussian White Noise** as noise type as shown in Fig. 10. Set "**Samples per second"** as **8000**, number of samples would be automatically adjusted as 800 corresponding to 100 ms duration.

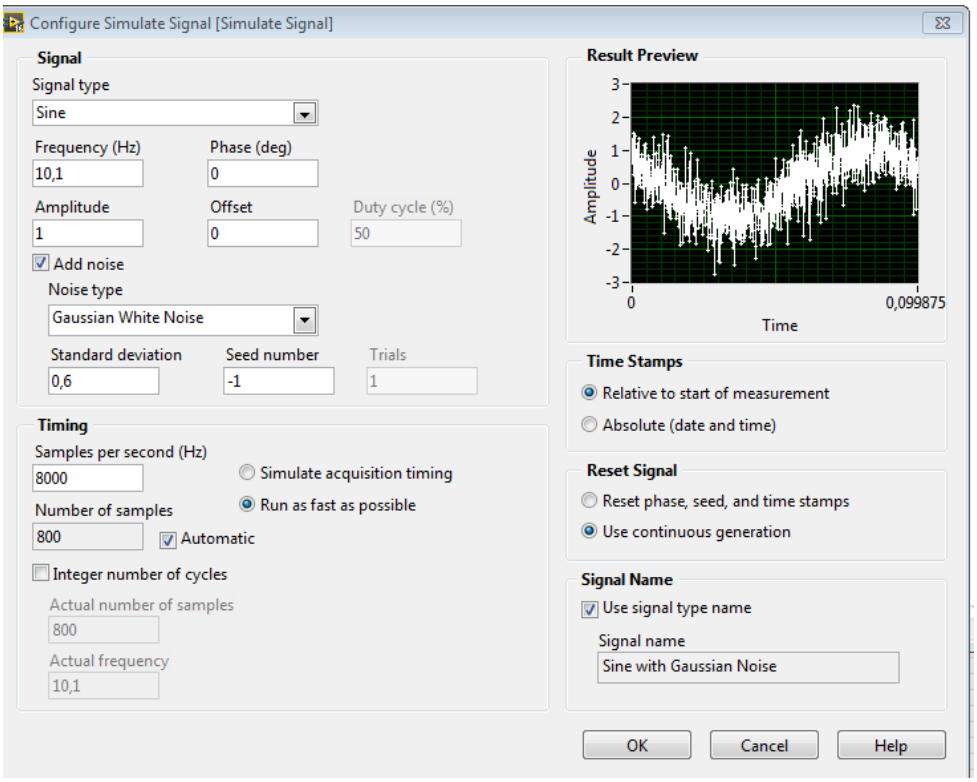


Fig. 10. Configuration of Simulate Signal

Create numeric controls for frequency, amplitude, phase and standard deviation as shown in Fig. 11.

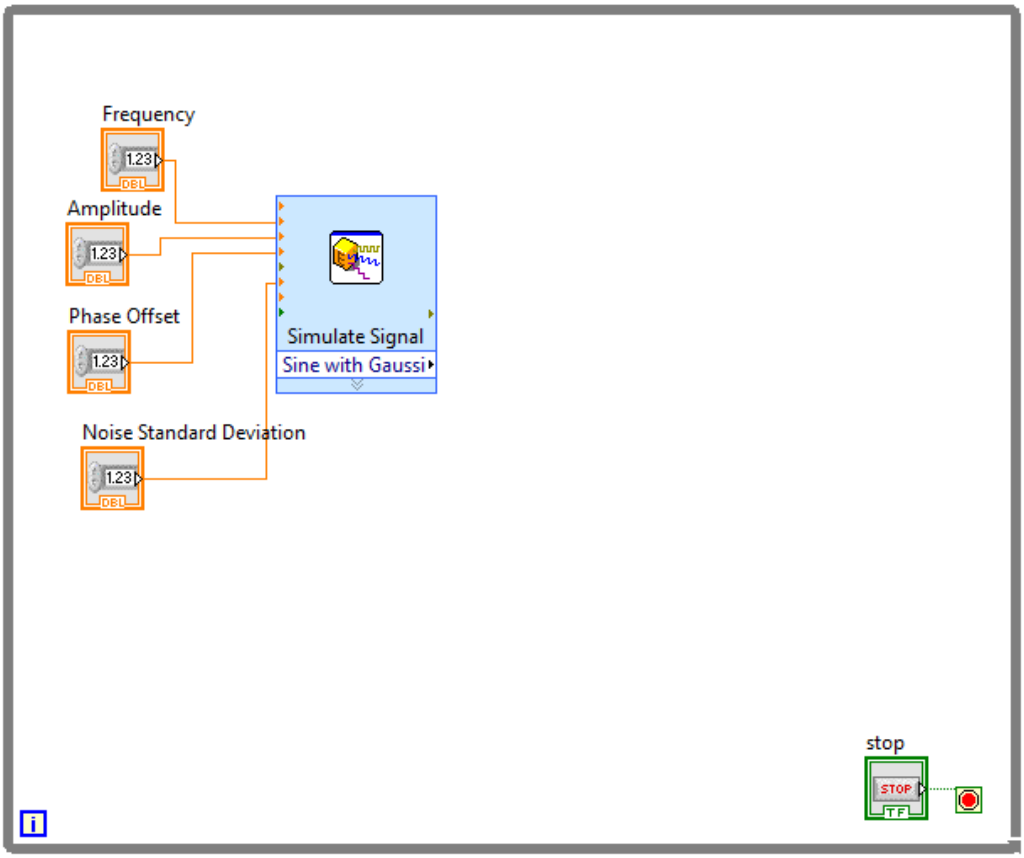


Fig. 11. Numeric Controls for Simulate Signal

Now, create a graph indicator by **right clicking on "Sine with Gaussian"** terminal on the express VI Simulate Signal and select **Create → Graph Indicator**. Furthermore, place an FFT function by selecting **Functions → Signal Processing → Transforms → FFT**. Connect the **"X" input of FFT function** to the **Sine with Gaussian** output of **Simulate Signal**. When you make connection, **Convert from Dynamic Data** function is automatically wired between two terminals. Fig. 12 shows the view of block diagram at this point.

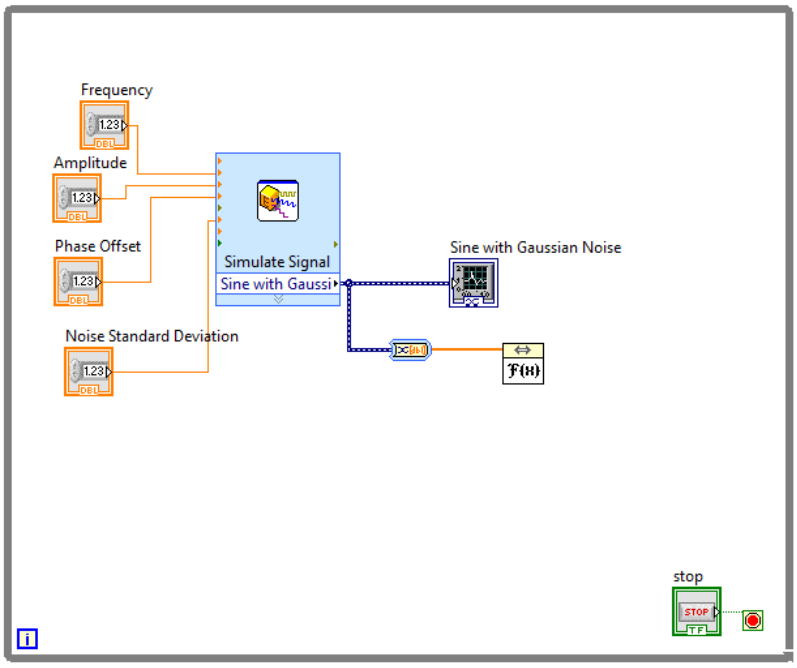


Fig. 12. Block Diagram after placing FFT function

Fig. 13 shows the complete Block Diagram for the Run_Sine.vi. Our aim is to plot magnitude in log domain and unwrapped phase (in order to eliminate discontinuities) of FFT of the simulated signal.

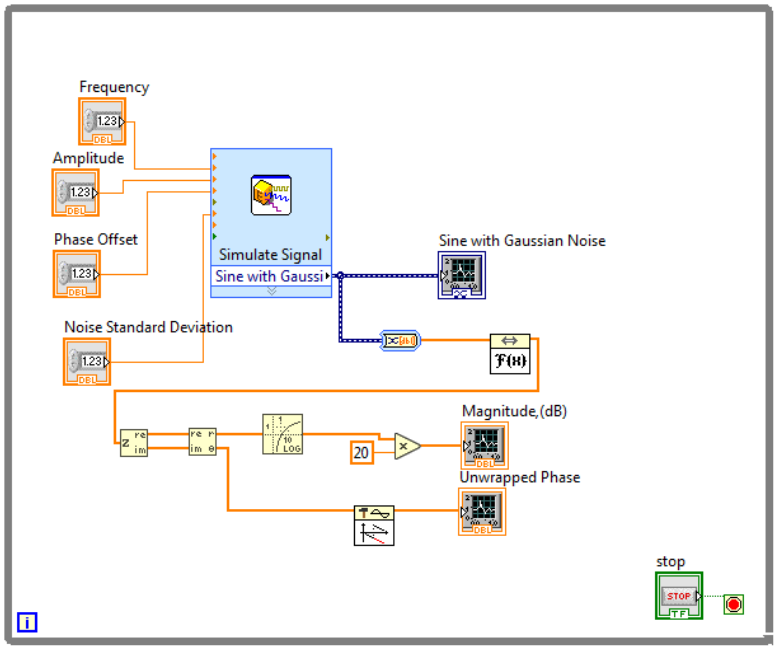


Fig. 13. Run_Sine.vi Block Diagram

As a first step, add **"Complex To Re/Im"** function in order to break a complex number to its rectangular components. You can find this function using **Search** button in **Functions palette**. Then, add **"Re/Im To Polar"** function to convert rectangular components into polar components. Place **"Logarithm Base 10"** function to take the logarithm of the magnitude and plot **20log10(magnitude of FFT coefficients)** using a **Waveform Graph**. Place **"Unwrap Phase.vi"** to eliminate 2π discontinuities and plot unwrapped phase using a **Waveform Graph.**

The Front Panel for this VI is shown in Fig. 14.

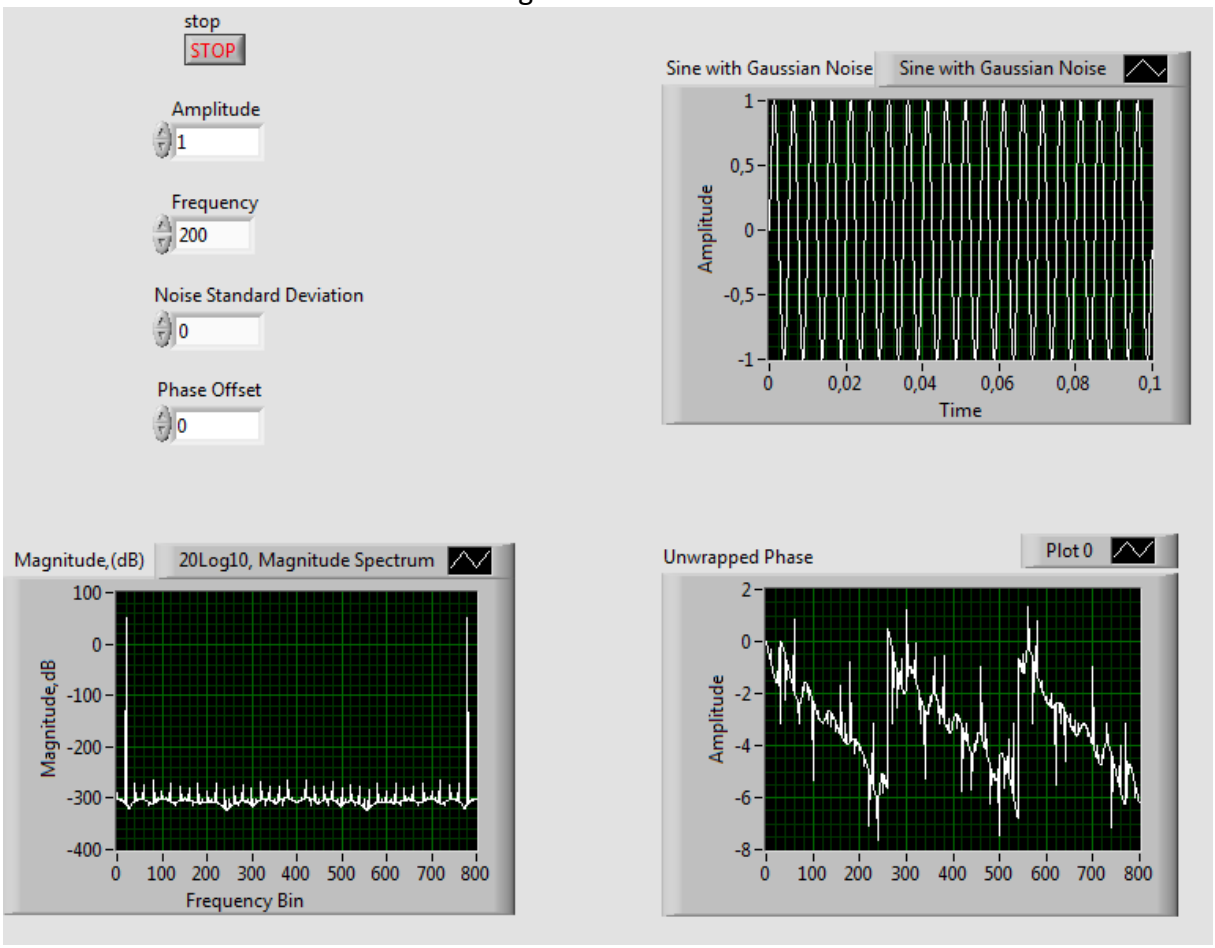


Fig. 14. Front Panel of the Run_Sine.vi

Now this VI can be run by clicking on it. Note that all the processing is done on PC and this may not be a real-time implementation in strict sense.

Programming Task 1:

a) Take the **Noise Standard Deviation** as **0** and the remaining parameters as shown in Fig. 14. Explain the Magnitude and Phase plot axis. The input sinusoid is **200Hz** and the sampling rate for the Simulate Signal block is **8kHz**. Identify the frequency bin corresponding to the input frequency in magnitude spectrum. You can **right click on any waveform** and select **Export → Export Data To Excel** to access the x- and y-axis values.

b) Now, change the **number of samples from 800 to 799** under **Configuration window**. Why does the sinusoid spectrum have a different shape while in theory it is expected

to be a single line as in Fig. 14? Note that there is no noise component in the sine in this step.

c) Set the **number of samples as 800** again as in the step a). Increase the Noise Standard deviation gradually (you can select 10-fold increases). Explain if you can identify the sinusoid better in time or frequency? Note that optimum detection technique of a single sinusoid corrupted by noise is to look at the magnitude spectrum. Determine the standard deviation value by which you can still identify the sinusoid in frequency. Calculate the SNR value for this case,

$$SNR = 10Log_{10}\frac{\sigma_x^2}{\sigma_n^2} \quad \text{(dB)}$$

where σ_x^2 and σ_n^2 stand for the signal and noise power respectively.

d) What is the information given by the phase plot?

2. Design of The Second vi, Run_Chirp.vi

In this part, you are required to **design Run_Chirp.vi** whose function is to generate a **chirp** waveform with a given parameter set and display its time and frequency characteristics. The **input parameters** are **sampling frequency, chirp amplitude, number of samples, start frequency, and end frequency**. There are three displays for **time waveform, magnitude spectrum and phase spectrum** respectively. The Front Panel for this VI is shown in Fig. 15. Note that in this case you cannot use Simulate Signal function since it has no chirp waveform. In order to generate chirp, you need to use the **Chirp Pattern.vi** function in the Block Diagram functions menu.

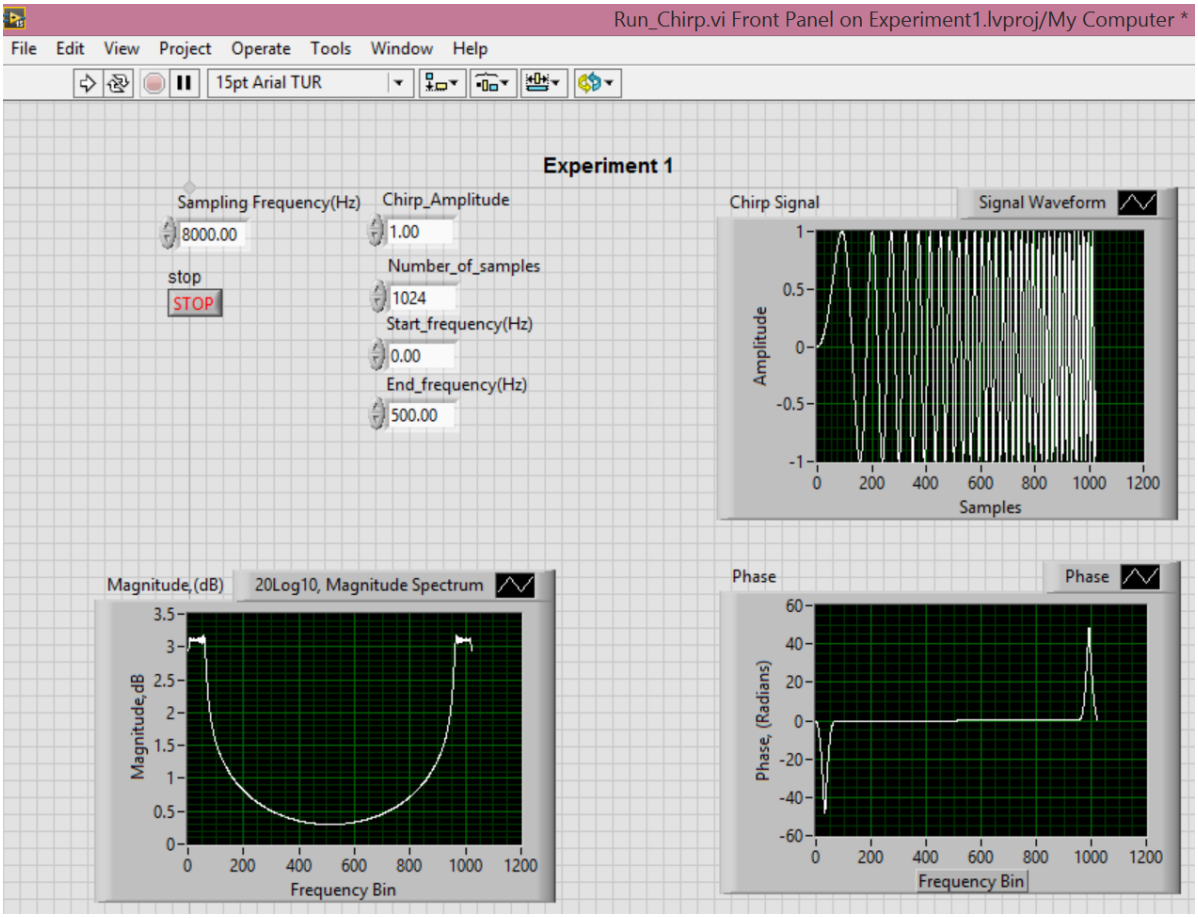


Fig. 15. Run_Chirp.vi Front Panel

Fig. 16 shows the **project view** of Experiment 1. Your implementation can be different but you can use it as an example.

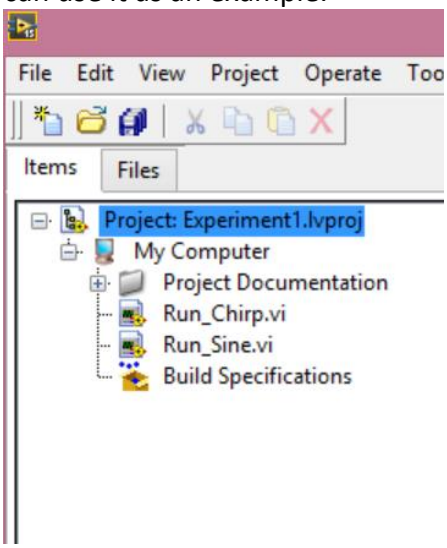


Fig. 16. Project for the Experiment 1

Programming Task 2:

Repeat the items given in programming Task 1 and explain the differences between two cases when the sampling rates are the same. **In addition** realize the following items:

a) Decrease the sampling rate gradually to a level at which you would not observe the chirp waveform characteristics given in Fig. 15. Note this value in your report. Can you identify a general rule for the minimum sampling rate?

b) Comment on the Magnitude and Phase characteristics of chirp signal. Chirp is usually used in radar systems. Explain this in comparison to a sinusoid. In other words, why the radar systems use chirp instead of a sinusoid?

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 2. PROGRAMMING BOTH MYRIO CPU AND FPGA

I. Introduction

This experiment is designed to implement filtering in both myRIO CPU and FPGA. However, due to online education, we will implement filtering in PC. Still, we will mimic the FPGA implementation by using a fixed-point representation on a PC. Therefore, you will also observe the FPGA implementation results and compare the CPU and FPGA implementations. myRIO has two processing environments. One is the myRIO CPU which performs floating-point operations. This is good since quantization and overflow errors in filter implementations are much less observable. The second processing medium is the myRIO FPGA. myRIO FPGA requires integer arithmetic since it uses fixed-point representation. When integer arithmetic is used, coefficient quantization, overflow and limit cycle errors can be observed in filter implementations. In this experiment, these sources of error are investigated by implementing IIR and FIR filters.

II. Preliminary Work

1) Read the following documents:

https://www.ni.com/documentation/en/labview-comms/latest/data-types/intro-fixed-point-numbers/

https://zone.ni.com/reference/en-XX/help/371361R-01/lvhowto/floating_point_numbers/

You may also make additional readings about floating-point and fixed-point representations from additional sources.

2) Read the following information about numeric representation in filter implementation.

Importance of Numeric Representation for Filter Implementation

Numbers can only be represented by finite number of bits in digital systems. Hence there is finite numerical precision in computers and embedded systems. Usually ***floating-point***

implementation is used in computers. This employs ***32 bits to 64 bits*** depending on the CPU and digital platform structure. In MATLAB, you usually would not sense the problems associated with numerical precision. However, in certain implementations you should be aware of that even 32 bits may not be sufficient especially for ***IIR filter*** implementation leading to unexpected filter outputs.

Embedded systems employ either ***fixed-point*** or ***floating-point*** representation. In fixed-point representation, numbers are represented by integers. Hence, the result of any multiplication, or addition should be quantized to an integer fitting into the numerical range imposed by the used number of bits (Ex. 16 bits).

In filter implementation, certain problems arise due to fixed-point realization. These are as follows.

a) ***Coefficient quantization:*** When the coefficients of the IIR filters are quantized, poles may move outside the unit circle leading to unstable filter realization. Coefficient quantization distorts the magnitude and phase characteristics of both FIR and IIR filters. Hence the frequency characteristics of filters should always be checked when the filter coefficients are quantized.

b) ***Finite register length:*** Arithmetic operations are performed using registers with finite length. When two numbers with 8-bit representation are multiplied, the resulting number is a 16-bit number. If the length of the register is 8 bits, then the numerical precision is lost due to quantization of a 16-bit number to 8-bit representation. Therefore, the filter output may deviate significantly from the perfect response. In addition, filter frequency response is also distorted.

c) ***Limit cycle:*** IIR filter are more adversely affected from finite numerical precision. This is due to the feedback from the output to the input. In this case, IIR filter output may oscillate even when there is no input. This oscillatory periodic output is called limit cycle.

As a result, one should be careful during the FIR or IIR filter realizations in ***embedded platforms***. MyRIO has two processing units, namely the CPU and FPGA. ***MyRIO CPU*** uses ***floating-point*** representation. Hence, numerical precision problems are less significant. ***MyRIO FPGA*** uses ***fixed-point*** representation and filters should be implemented by considering the above points.

One of the most critical aspects of filter implementation is the ***computational complexity***. Especially, real-time systems have limited computational resources. In such cases, long FIR filters cannot be implemented. Signal Processing theory shows that the computational complexity for FIR filter implementation is less if it is implemented in Fourier Domain. Hence FFT algorithm is employed for efficient FIR filter implementation.

An alternative for computationally ***efficient FIR filter*** implementation is to design and use ***"multiplierless"*** filters. The filter coefficients for these FIR filters are powers of two which can be implemented by simple ***"bit shift"*** operation. Hence these filters can be implemented by

using only addition. This is very significant since ***multiplication is a costly operation***. While multiplierless filters are advantageous, their frequency characteristics are inferior to the cases where the filter coefficients are represented by floating-point representation.

3) In order to familiarize with the FIR and IIR basic filter structures, do the following MATLAB assignment.

FIR and IIR Filter Implementation in MATLAB

a) Run ***fdatool*** to design filters by writing "fdatool" to the ***Command Window*** in ***MATLAB***. After you press Enter, ***Filter Design & Analysis Tool*** window appears as shown in Fig. 1.

b) Design a ***FIR lowpass equiripple*** filter. The ***passband*** and ***stopband*** frequencies are ***0.3π*** and ***0.4π*** respectively. Passband and stopband ripple sizes are ***1dB*** and ***-60dB*** respectively. These parameters are selected as shown in Fig. 1. Note that when Normalized (0 to 1) option is selected, you should enter ***0.3*** and ***0.4*** as passband and stopband frequencies. Here, the discrete frequency π is normalized to 1. Plot the ***magnitude and phase characteristics*** of this filter by clicking on ***Design Filter***. Note that this filter is optimum in minimax sense. Obtain the filter coefficients using the fdatool. Is this a ***symmetric filter***? Is this a ***linear-phase filter?*** Plot the pole-zero plot using the fdatool item. ***Comment*** on the placement of zeros. Please do not forget to ***attach magnitude & phase characteristics and plot of filter coefficients and pole-zero plot to your preliminary work***.

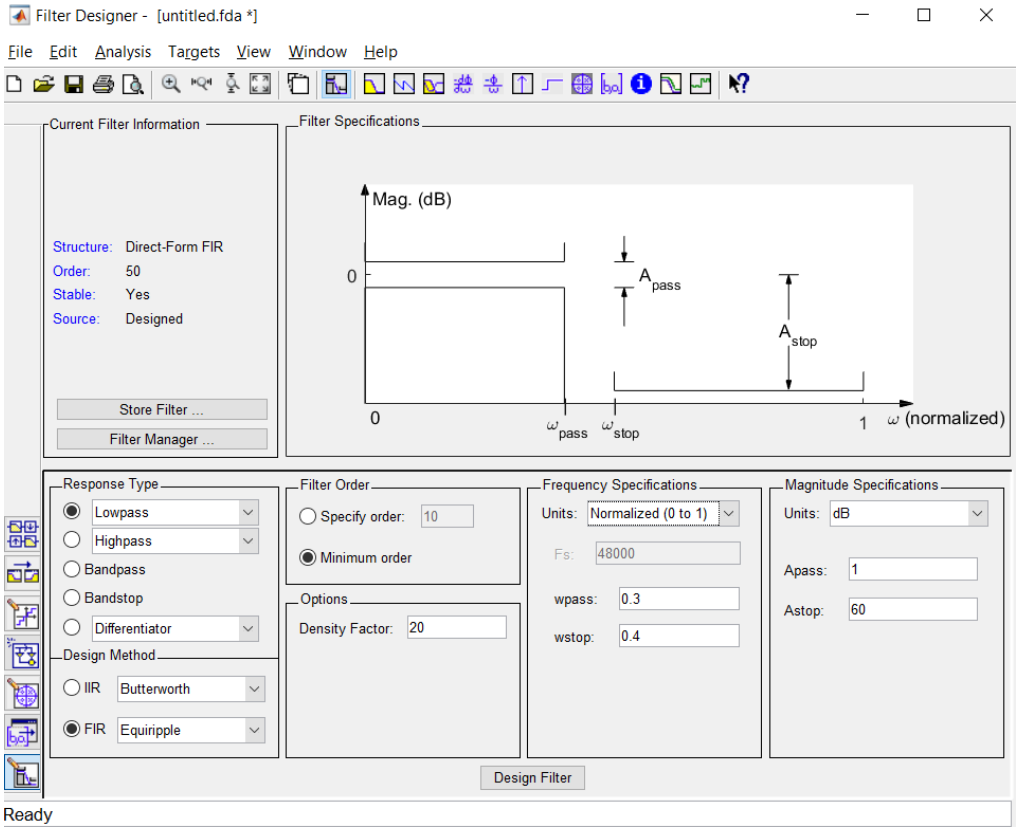


Fig. 1. Filter Design & Analysis Tool.

c) Now design an ***IIR Chebyshey Type II*** filter with same parameters. Write the order of the filter and compare it with previous FIR equiripple filter. In addition, ***comment*** on the placement of zeros and poles considering the magnitude characteristic of the filter. Please do not forget to ***attach magnitude & phase characteristics and plot of filter coefficients and pole-zero plot to your preliminary work.***

d) In this part, you need to write a MATLAB code to implement ***first-order FIR and IIR lowpass filters***. The ***input*** to these filters should be a ***triangular waveform*** where the ***amplitude is 1*** and ***frequency is 200Hz*** with ***sampling frequency 4000 samples/sec***. The first-order filter structures should be implemented in its basic form and ***MATLAB built-in functions should not be used***. The filter coefficients can be selected arbitrarily. The filter outputs should be observed both ***in time and in frequency***. Don't forget to ***attach all the plots*** to your preliminary work.

- Attach ***all the MATLAB codes*** you will write in this assignment to the preliminary work.

III. Experimental Work

1. Reading - Simple Real-time Programming on MyRIO CPU

- In LabVIEW, a project describes the hardware and software components. Hardware components are usually composed of a PC, and a real-time embedded system. Real-time system has a ***"Chassis"*** under which it has both a CPU, FPGA and some analog and digital ports. You can run a program (in our case a ".vi") on a PC, on myRIO CPU or myRIO FPGA. The only thing you need to do to select the processing medium is to place your ".vi" under the selected medium. For example, if you want to run your .vi on myRIO CPU, you need to place your .vi under the myRIO Chassis. An example is shown in Fig. 2.

- In the first part of this experiment, the steps for constructing filtering using **floating-point representation in *PC*** are outlined. In the second part, filtering using **fixed-point representation in *PC*** is discussed. Then, the required programming tasks are described

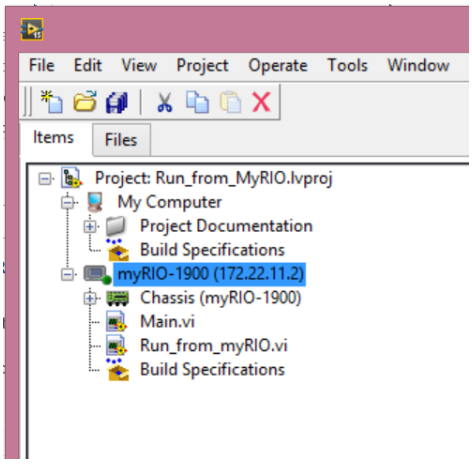


Fig. 2. LabVIEW project for real-time programming in myRIO.

2. Programming Task 1 – Filtering operations in PC using floating-point representation

a) First, create a new project and .vi as described in Experiment 1.

b) In this part, you need to implement ***first-order IIR and FIR filters on PC CPU using floating-point representation***. The ***input waveform*** for these filters is taken from a ***triangular waveform generator***. The filter implementation is done using only the basic building blocks of LabVIEW and hence the ***filter block of LabVIEW should not be used***. The outputs of both filters should be presented in the front panel both ***in time and in frequency***.

- The transfer function of a first-order IIR filter is given below,

$$\frac{Y(z)}{X(z)} = H(z) = \frac{1}{1 - az^{-1}} \qquad (1)$$

where a is the filter coefficient and it is given to the program as input. The difference equation for this filter is,

$$y[n] = ay[n-1] + x[n] \qquad (2)$$

- The transfer function of a first-order FIR filter is given as,

$$\frac{Y(z)}{X(z)} = H(z) = 1 - bz^{-1} \qquad (3)$$

where b is the FIR filter coefficient which is given as the input from the front panel. The difference equation for the FIR filter is given as,

$$y[n] = x[n] - bx[n-1] \quad (4)$$

When you compare the equations (2) and (4), it is easily seen that (2) has feedback and hence leads to infinite length impulse response whereas (4) has only 2 coefficients where the first one is one for simplicity.

- You need to write your own code to compute the output samples of these filters using (2) and (4). The ***inputs on the front panel*** will be the ***triangular wave amplitude, frequency, IIR and FIR filter coefficients (a and b).***

i) Generate a triangular waveform using built-in LabVIEW functions. One such function is ***Triangle Wave PtByPt.vi*** whose detailed help window is shown in Fig. 3. This function generates the ***samples of a triangle wave one at a time***. Hence this and the other program blocks should be ***inside a while loop so that you can process many samples in sequence***. You should be able to control the amplitude and frequency of the triangle wave from the front panel. Set phase as 0 and time as ***(0,0001*iteration number)*** as shown in Fig. 4. So sampling rate is 10 kHz for the triangular wave.

ii) Enter the ***IIR filter coefficient*** from the front panel input. In the block diagram, this coefficient is used in the IIR filter structure. The input to this filter structure is the triangle wave samples. The filter is implemented using a feedback node.

iii) Use ***Power Spectrum.vi*** to compute the Fourier spectrum of the filter output. Plot the output spectrum using ***Logarithm Base 10.vi*** with ***waveform graph*** in the front panel. You can evaluate Fourier spectrum for ***blocks with 256 points.*** For this purpose, you can create a shift register for a data array with 256 elements as shown in Fig. 5.

iv) You should also ***plot the time-domain output*** on the front panel with the ***waveform chart block, point by point***.

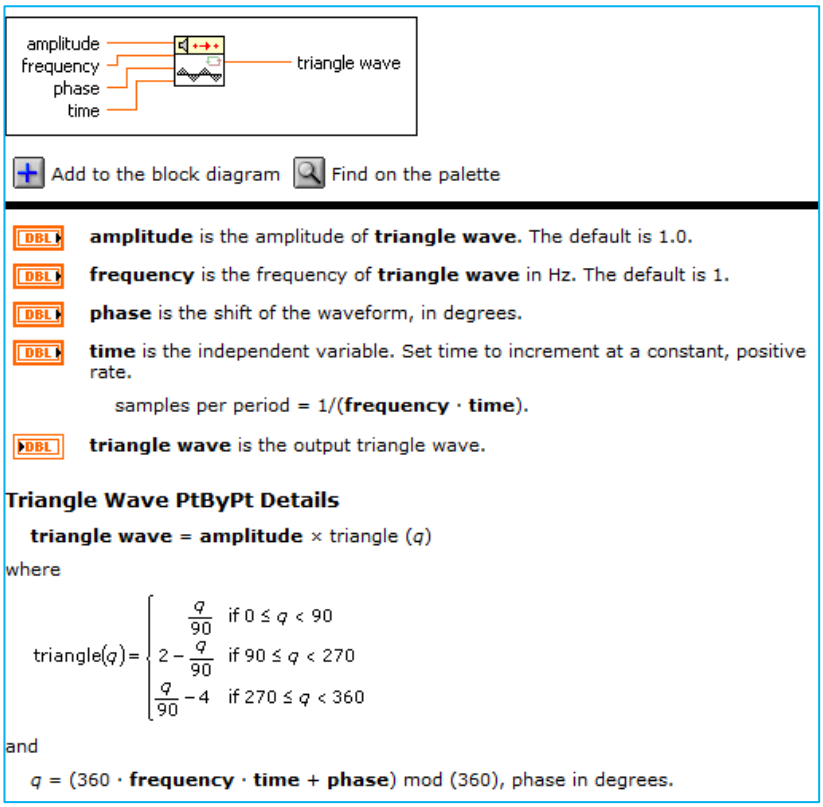


Fig. 3. Triangle Wave PtByPt Details.

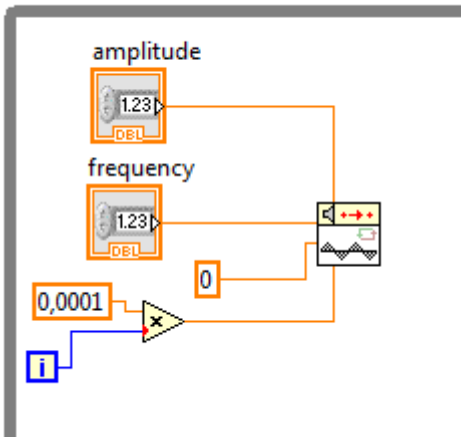


Fig 4. Triangle waveform generation in a while loop.

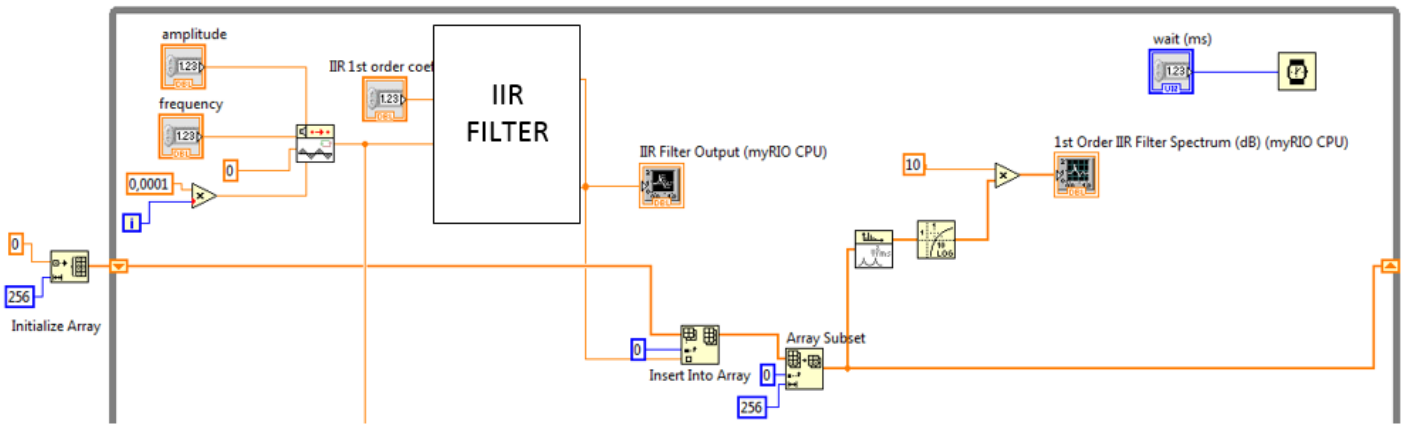


Fig 5. Processing of the last 256 points.

v) ***Repeat ii, iii and iv*** for the ***FIR filter*** in order to display its output in ***time and Fourier domain.***

3. Programming Task 2 – Filtering operations in PC using fixed-point representation

- In this part, you need to implement a ***first-order IIR filter on PC CPU using fixed-point representation***. myRIO FPGA uses fixed-point representation. Hence here we mimic FPGA implementation in a PC.

- Define ***"Bit Depth"*** as a control in PC program. This item determines how many bits are used to represent the filter coefficients and the triangular waveform samples. As this item gets smaller, a course representation is used for each component.

- FPGA uses fixed-point arithmetic. Therefore, we need to convert our data to integer format. For this purpose, we will multiply the controls to be used in the FPGA by ***2^(Bit Depth)*** and convert it to the ***32-bit integer*** as shown in Fig. 6. Here, we use ***Scale By Power Of 2*** and ***To Long Integer*** functions. Note that, Bit Depth determines the number of bits representing the numbers between -1 and 1 in integer format in the FPGA. (Note that if we were using myRIO FPGA, you would need to transfer the coefficients through the procedure called ***"Front Panel Communications.***)

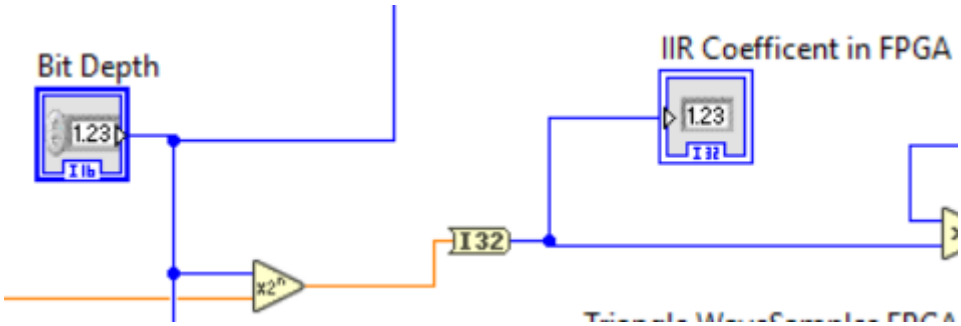


Fig.6. Conversion of the controls to integers

- We also need to convert the triangle wave generator's output to integer to be processed in the FPGA. Multiply the output of the triangle wave generator by ***2^(Bit Depth)*** and convert it to the ***32-bit integer.*** (Note that if we were using myRIO FPGA, you would need to transfer the triangle waveform outputs through the procedure called ***Direct Memory Access (DMA) using FIFO's.***)
- Once you converted your filter coefficients and triangle wave generator's output to integer format, you can implement the filtering similar to the floating-point representation case. However, be careful about the followings:

  i) In the fixed-point representation case, we ***divide the multiplication of IIR coefficient and the previous output*** by ***2^(Bit Depth)*** before summing it with the new input sample. For the division, we use ***Quotient & Remainder*** function since floating-point arithmetic is not allowed in the FPGA. The reason for this division can be seen from the following IIR filter recursion equation,

  $$y[n] = ay[n-1] + x[n] \qquad (5)$$

  (5) is implemented in the FPGA by (6), i.e.,

  $$2^{BitDepth} y[n] = 2^{BitDepth} ay[n-1] + 2^{BitDepth} x[n] \qquad (6)$$

  So, at each iteration we find the filter output as $2^{BitDepth} y[n]$. Since the feedback $2^{BitDepth} y[n-1]$ is multiplied by $2^{BitDepth} a$, we need to divide the output of this multiplication by ***2^(Bit Depth)*** in order to apply recursion in accordance with (6).

  ii) You will also need to divide the final output by ***2^(Bit Depth)*** before visualization for scaling.

- Use IIR Filter Output found in the fixed-point representation to ***display power spectrum*** using another shift register for 256 element array as in the previous steps.

- Add a Wait function with a control inside the while loop in PC VI as shown in Fig. 7s.

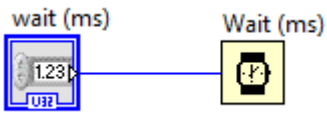


Fig. 7. Wait function with a Control in CPU VI.

- Your front panel must look like Fig. 8.

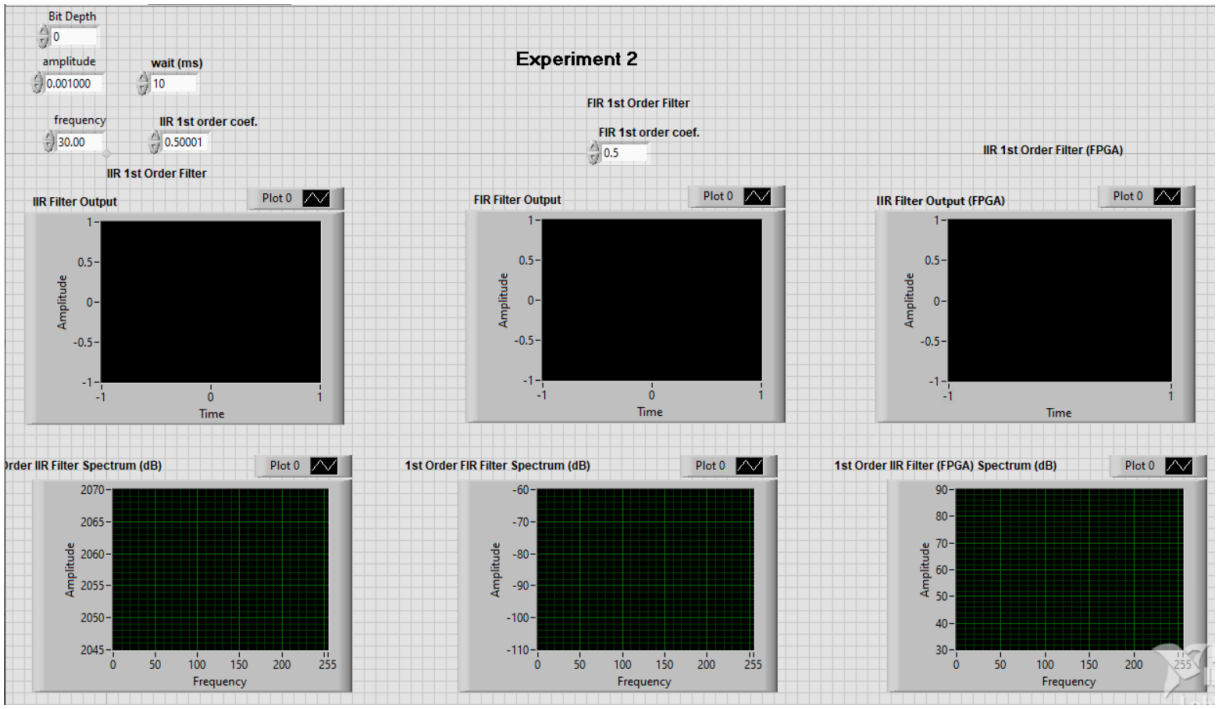


Fig.8. Front panel for the Experiment 2.

i) Set ***amplitude*** and ***frequency*** of Triangular waveform as ***0,001*** and ***30***, respectively. Set ***wait(ms)*** and ***Count(Ticks)*** as 10.

ii) Choose the ***IIR and FIR filter coefficients*** as ***0.5*** and ***Bit Depth*** as ***14***. Note and ***plot*** the filtered outputs for both CPU and FPGA implementations.

iii) Change ***filter coefficients*** as ***0.99***. Note the ***changes in myRIO CPU and FPGA implementations***. ***Comment*** on the results. What happens to ***FIR filter output***? Does it change significantly?

iv) ***Decrease Bit Depth*** one by one from ***14*** to ***10***. Note the changes in the FPGA output. ***Explain*** them.

v) Now, ***increase Bit Depth*** from ***14*** to ***19*** one by one and note the changes. Why do we observe such a response for the ***FPGA implementation*** which is ***different*** than the ***CPU implementation*** in myRIO.

vi) Change the ***IIR filter coefficient*** to ***1.2***. Why do we observe such a response for both of the implementations in myRIO?

vii) Change the ***FIR filter coefficient*** as ***1.2***. ***Why don't*** we see a similar change for the FIR filter?

viii) Implement ***second-order FIR and IIR filters*** in the Fourier domain using FFT in a block-by-block manner on PC using ***floating-point representation***. Each input block is obtained and processed using the filter coefficients in the Fourier domain. You can use 256-point FFT in your implementation. The filter output is displayed on the front panel.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 3 PART 1
SIGNAL GENERATION, FILTERING, CROSS CORRELATION, A/D, D/A, DMA MATLAB IMPLEMENTATION

I. Introduction

In Experiment 3 Part 1, several signal processing tasks will be implemented in MATLAB.

II. Preliminary Work

1)

a) Generate the following **10-point sequence x** and **5-point sequence h** in MATLAB.

x= [1 2 3 4 5 6 7 8 9 10],

h= [10 -8 6 -4 2].

b) Using the ***"conv"*** command, **plot** the convolution of x and h. **Attach this plot** to your preliminary work.

c) Using the ***"xcorr"*** command, **plot** the cross-correlation of x and h. **Attach this plot** to your preliminary work.

d) Now, **plot** the convolution of x with **the lipped version of h**. **Attach this plot** to your preliminary work. You can use the ***"fliplr"*** command to obtain the flipped version of h.

e) Is there any **similarity** between the results obtained in **b, c, and d**? **Comment** on the **similarity and differences** between these three results.

2)

a) Generate **200 Hz sine sequence** with sampling rate **10000 samples/s in 0-80 ms** range. **Plot** it with the **appropriate time axis**, i.e., **0-80 ms**. **Attach this plot** to our preliminary work.

b) **Cross correlate the sine sequence** you generated in part a) with **itself**. **Plot** the result and **attach** it to your preliminary work.

c) Now, generate a **chirp signal** with a sampling rate of **10000 samples/s in 0-80 ms** range with **start frequency 0** and **end frequency 200 Hz**. **Plot** it with the **appropriate time axis, i.e., 0-80 ms**. **Attach this plot** to our preliminary work.

d) **Cross correlate the chirp sequence** you generated in part **c)** with **itself. Plot** the result and **attach** it to your preliminary work.

e) **Compare** the cross-correlated sequences you obtained in parts b) and d).

f) **Cross correlate the chirp signal** in part **c)** with the **sine** signal and **plot** the result. **Attach** it to your preliminary work.

g) Generate **another chirp signal** with sampling rate **10000 samples/s in 0-80 ms** range with **start frequency 200 Hz** and **end frequency 400 Hz**. **Cross correlate this chirp signal with the sine signal** and **plot** the result. **Attach this plot** to your preliminary work.

h) Generate **another chirp signal** with sampling rate **10000 samples/s in 0-80 ms range with start frequency 400 Hz** and **end frequency 600 Hz**. **Cross-correlate this chirp signal with the sine signal** and **plot** the result. **Attach this plot** to your preliminary work.

i) **Compare** the results in part **f), g), h)**. **Comment** on them.

j) Now generate a **200 Hz square wave** with a sampling rate of **10000 samples/s in 0-80 ms** range. **Plot** it with the **appropriate time axis, i.e., 0-80 ms**. **Attach this plot** to our preliminary work. (You can use the **"square"** command in MATLAB)

k) **Cross correlate the square wave** in part **j)** with the chirp signal in part **h)** and **plot** the result. **Attach this plot** to our preliminary work. **Compare** the results with the plot in **h)** and **comment on the differences** considering frequency content of the square wave.

III. Experimental Work

1) Write a MATLAB function that generates a **sine or square waveform** with **256 samples** based on the "sine or square" input. For both of the wave generations, you should use the **"sin"** function of MATLAB. You will adjust the frequency, sampling frequency, amplitude, phase of the waveforms based on the tasks in the report.

2) You should have **Frequency Hopping** control in your implementation. Nothing is done when Frequency Hopping control is false. When this control is True, we want that the frequency of the wave changes **randomly** in some neighborhood of the selected frequency. You will use **uniformly distributed random numbers** for this purpose. Deviation should be **symmetric** around the actual frequency. You will also define a control input called **"frequency deviation" in Hz**.
3) You should have **Noise** control in your implementation. Nothing is done when Noise control is false. When this control is True, we want to add noise to the signal. You will use **normally distributed random numbers** for this purpose. You will also define a control input called **"noise standard deviation."** Write **"rng default"** at the very beginning of your code so that random number generation is the same for all of us.
4) Write your own code to **estimate the frequency of a signal**. This code will operate in noisy and different sampling rate cases.
5) Write your own code for **SNR estimation**.
6) For filtering, you should use the **Butterworth filter**.
7) You should plot all of your signals both in the **time domain and frequency (in Hz) domain**.

Task

1) Generate a sine wave **without frequency hopping**. Choose the **frequency 3 kHz**, **sampling frequency 50 kHz**, **amplitude 1**. Increase the sine wave frequency to **24kHz** in **3kHz steps**. **Comment** on the change in the waveforms. **Attach** the plot for the **24kHz** case.

2) Keep the frequency of the sine wave at 3kHz. **Increase the sampling rate from 50kHz to 500 kHz in 50kHz steps**. **Comment** on the change in the waveforms. **Attach** the plot for the **500kHz** case.

3) Keep the frequency of the sine wave at 3kHz. **Decrease the sampling rate from 50kHz to 10 kHz in 10kHz steps**. **Comment** on the change in the waveforms. **Attach** the plot for the **10kHz** case.

4) Now set the **sine wave frequency** and **sampling frequency** to **1 kHz and 50 kHz**, respectively. **Activate the frequency hopping**. Set the **frequency deviation to 100 Hz**. Observe the waveforms. Increase the frequency deviation **from 100 Hz to 500 Hz in 100 Hz steps**. **Observe** the changes in the waveforms. **Comment** on the results.

5) **Repeat steps 1, 2, 3, and 4 for the square wave.**

6) Generate a sine wave **without frequency hopping** with a **sampling frequency of 100 Hz**. Adjust the frequency of the sine wave to **101 Hz**. **What** do you **observe**? **What** is the **frequency of the observed signal**? **Comment** on the results.

7) Now increase the frequency to **105 Hz in 1 Hz steps**. **Comment on the frequencies** of the waveforms.

8) Increase the sine wave frequency from **1001 Hz to 1005 Hz in 1 Hz steps** while the **sampling frequency is still 100 Hz**. Are the results **similar** to 7? **Why**?

9) Now generate a sinusoid at **1000 Hz** frequency and select the sampling frequency $\mathbf{f_{S1} = 16\ kHz}$. Assume that we have converted this signal to analog waveform and then sampled again with $\mathbf{f_{s2} = 16\ kHz}$. Increase and decrease **fs1** to **32kHz and 8kHz**, respectively and note the received signal's **estimated frequency**. Also, **note the differences in time and frequency between each case**. Now select $\mathbf{f_{s1}}$**=16kHz** and change $\mathbf{f_{s2}}$ **to 32kHz and 8 kHz**, respectively. **Note the estimated frequency and differences in time and frequency.**

10) Generate a **sinusoid at 1 kHz**. Choose the **sampling frequency of 16 kHz**. **Add noise** to the generated signal by changing the **noise standard deviation from 0 to 15 with an increase in 5 steps**. Note the **estimated frequency and SNR in each case**. Now **increase the noise standard deviation to a value** such that sinusoid frequency **cannot be estimated any longer**. **Note this value** and write it in your report.

11) Continue **from 10** but select **noise standard deviation as 0.1**. Observe the **cross-correlation of the input with the chirp**. Select the chirp **f1 and f2 frequencies as 160 Hz and 6400. Increase the value of f1 in 1600 steps until 6400. Note the changes in cross-correlation.**

12) **Repeat 10 using frequency hopping mode** where the **frequency deviation** is set as **50Hz**.

13) Generate a **square wave** with a **period** of **T=0.5ms**. Choose the **sampling rate as 64 kHz**. Choose the **normalized low cutoff frequency of the Butterworth filter** as $\frac{\mathrm{f_{square}}}{\left(\frac{\mathrm{f_{sampling}}}{2}\right)}$**.** You can use the **butter** and **filter** functions of MATLAB. Note the **input and filtered signal waveforms** both **in time and in frequency**. Determine **the main spectrum width** for the square wave and **its relation to the period, T**. You can change the period to see its effect on the frequency spectrum. **Decrease T by 2, 4**

and increase it by 2, 4 to identify the **time-frequency relation**. Also, **note the cross-correlation function** between **the input** and the **filtered output in each case**.

14) **Repeat 13 when the signal is corrupted by noise**. Select the **noise standard deviation as 10**. Note the input and filtered signal waveforms both in time and in frequency. **Comment on the similarity between the input and output in each case**.

15) **(Bonus 20pts)** Modulate **the square wave generated in step 13 by a 6kHz sinusoid**. Select the **noise standard deviation as 1**. **Filter the resulting signal with a bandpass filter to remove noise as much as possible while keeping the signal characteristics**. **Determine the minimum bandwidth that you can use for this purpose**. **Demodulate** the signal down to baseband by **multiplying with another sinusoid** with the same frequency and **lowpass filtering** the resulting waveform. **Plot the input and demodulated signal time and frequency characteristics in the front panel.** Note that this structure is used to transmit and receive signals, just like a standard communication system.

Experiment 3 Part 2

In this experiment, you will perform Experiment 3 Part 1 and additional tasks in LabVIEW. You can use your part 1 code to get an idea about these implementations in LabVIEW.

1. Create a **Sine vs Square** control. If this control is **true**, a **sine wave will be generated**. If this control is **false**, **a square wave** will be generated. For both of the wave generations, you should use "**Sine Wave PtByPt.vi."** This vi generates a sine wave point by point. Therefore, **to create an N-sample signal**, it should be placed in a for loop. For continuous processing, your block diagrams should be placed in a while loop.
2. Create a **Frequency Hooping** control. Nothing is done when Frequency Hopping control is false. When this control is true, we want that the frequency of the wave changes randomly in some neighborhood of the selected frequency. You can use **"Random Number (0-1)"** for the random number generation. Deviation should be **symmetric around the actual frequency**. You will also define a control input called **"frequency deviation"** in Hz. For instance, if frequency deviation and input frequency are set to **25 Hz and 200 Hz**, respectively, your wave frequency should change between **175 Hz and 225 Hz**.
3. Create an **AWGN** control. If this control is true, you will add Gaussian White Noise to your signal. For this purpose, use **"Gaussian White Noise PtByPt.vi."** You should also have **"AWGN Standard Deviation"** control to set the standard deviation of the generated noise.
4. Design a block diagram for the frequency estimation.
5. Design a block diagram for the SNR estimation.
6. You should plot your signals both in the **time-domain** and **frequency-domain**.
7. For the **appropriate time-domain** plots, use **"Build Waveform."** This vi takes input array **Y and dt** (specified time interval between two data points) as inputs and returns the waveform. **Input should be an array**. Therefore, you must store your signals in an array before building a waveform. Create **number of samples** control to set the length of signals.
8. We will also hear the generated signal by using the sound interface of LabVIEW. For this purpose, first, configure sound out device by using **"Sound Output Configure.vi**". Choose the **number of samples/ch as the number of samples control** and **sample mode as continuous samples.** Device ID ranges from 0 to n-1, where n is the number of output devices of the computer. It is usually selected as 0. For **sound format,** create a control. Set **sample rate to 44100, number of channels to 1, and bits per sample to 16. This means that your input signal is converted to an analog signal using a sampling rate of 44100 Hz**.
9. To control the volume of the output sound, use **"Sound Output Set Volume.vi."** Input Task ID and error in of this vi should be connected with the output Task ID and error out of **the Sound Output Configure.vi**. For volume input, you can create **slide control from**

numeric-> vertical pointer slide in the front panel. You can change its range from **properties -> scale**. Make the maximum value 100 and minimum value 0.

10. To play the sound, use **"Sound Output Write.vi."** Input Task ID and error in of this vi should be connected with the output Task ID and error out of the **Sound Output Set Volume**.vi. The data input should be the signal array.
11. Insert "**Sound Output Clear.vi.**" Input Task ID and error in of this vi should be connected with output Task ID and error out of the **Sound Output Write.vi.** Finally, insert "**Simple Error Handler.vi"** and connect error out of **Sound Output Clear.vi** to the error in of S**imple Error Handler.vi**. The whole procedure is shown in Fig. 1.

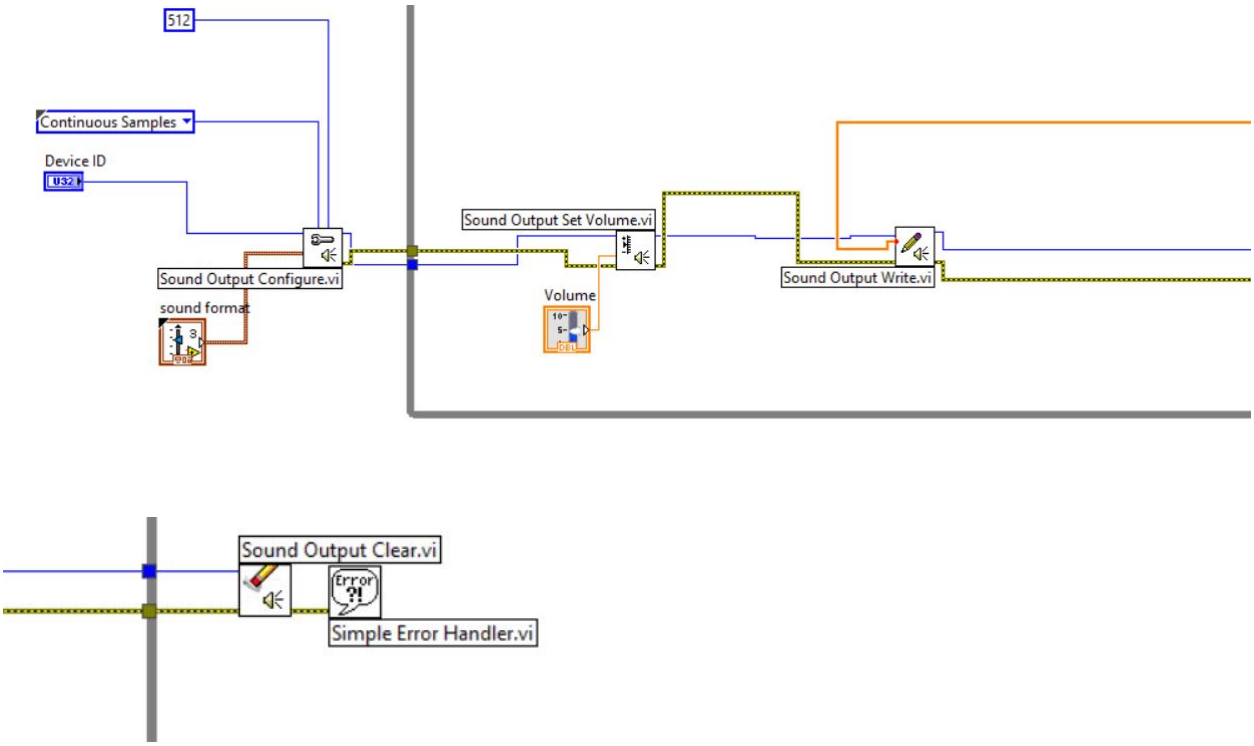


Fig.1. Sound Configuration

Tasks

1. Place the block diagram and front panel.
2. Set the amplitude to 1, number of samples to 512, phase to 0, and frequency to 500 Hz of the wave generator. Choose **Sine vs Square** true so that the sine wave will be generated. Choose the sampling frequency 25 kHz. Increase the sinewave frequency from 500 Hz to 2500 Hz by 500 Hz steps. **Comment on the change in the waveforms. Attach** the plot for the 2500 Hz case.

3. Keep the frequency of the sine wave at 500 Hz. **Increase the sampling rate from 25 kHz to 175 kHz in 50 kHz steps**. **Comment** on the change in the waveforms. **Attach** the plot for the **175 kHz** case.
4. Keep the sine wave frequency at 500 Hz. **Decrease the sampling rate from 25 kHz to 5 kHz in 5 kHz steps**. Comment on the change in the waveforms. **Attach** the plot for the **5 kHz** case.
5. Now, set the **sine wave frequency** and **sampling frequency** to **500 Hz and 25 kHz**, respectively. **Activate the frequency hopping**. Set the **frequency deviation to 1 Hz**. Observe the waveforms. Increase the frequency deviation **from 1 Hz to 250 Hz in 50 Hz steps**. **Observe** the changes in the waveforms. **Comment** on the results.
6. **Repeat steps 2, 3, 4, and 5 for the square wave.**
7. Generate a sine wave without frequency hopping with a **sampling frequency of 100 Hz**. Adjust the frequency of the sine wave to 101 Hz. What do you observe? What is the frequency of the observed signal? **Comment** on the results. Attach the front panel.
8. Now, increase the frequency to 105 Hz in 1 Hz steps. **Comment** on the frequencies of the waveforms.
9. **Increase** the sine wave frequency from **1001 Hz to 1005 Hz in 1 Hz steps** while the **sampling frequency is still 100 Hz**. Are the results similar to 8? **Why**?
10. Set the amplitude to 1, number of samples to 512, phase to 0, and frequency to 500 Hz of the wave generator. Choose the sampling frequency 25 kHz. **Note the estimated frequency**. Now increase the number of samples by powers of two, i.e., 1024, 2048, 4096. **Note the estimated frequencies**. **Comment** on the results.
11. Generate a sinusoid at 500 Hz. Choose the sampling frequency of 25 kHz. **Add noise** to the generated signal **by changing the noise standard deviation from 0 to 5 with an increase in 5 steps**. Note the **estimated frequency and SNR** in each case. Now increase the noise standard deviation to a value such that sinusoid frequency cannot be estimated any longer. **Note this value** and write it in your report.
12. Set the amplitude to 1, number of samples to 512, phase to 0, frequency to 500 Hz, and sampling frequency to 44100 Hz. Choose sine or square as true. **What** do you hear? Change the frequency to 3000 Hz by 500 Hz steps. **Comment** on the changes.
13. Now, choose sine or square as false, such that a square wave is generated. Repeat the 14. **Compare** your findings with 14 as well. **What is the difference between sine and square wave sounds?**
14. Set the amplitude to 1, number of samples to 512, phase to 0, frequency to 500 Hz, and sampling frequency to 44100 Hz. Now **change the sampling frequency of the sound output configure to 22050 and 88200** Hz. **What** do you hear? **What** is the reason? **Comment** on the results.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 4. DECIMATION, INTERPOLATION AND PHASE-LOCKED LOOP

I. Introduction

In this experiment, we will explore decimation, interpolation and phase-locked loop implementations in MATLAB and LabVIEW.

II. Preliminary Work

1) Read the following information on decimation, interpolation and digital phase-locked loop.

A. Sampling Rate Change

Sampling rate change is an operation to increase or decrease the sampling rate of a signal to match with the sampling rate of another signal. Sampling rate change is a frequently used since signal processing produces meaningful results only when two signals are at the same sampling rate. Sampling rate change can be easily done in digital domain by using decimation and interpolation. The rate change can be integer or a rational number but it cannot be a real number in general.

A.1. Decimation

Decimation reduces the sampling rate of the input signal by a factor of M where M is an integer. Hence decimation discards samples uniformly resulting compression in time. This generates expansion in frequency which in turn causes “aliasing”. Aliasing is a loss of information and is irreversible.

Decimation has two components, namely the decimation filter and the compressor. The function of the decimation filter is to bandlimit the input signal before the compressor such that there is no aliasing. Fig. 1 shows the block diagram of decimation.

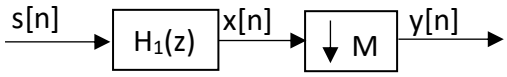


Fig. 1. Decimator composed of a filter and a compressor.

The relation between x[n] and s[n] is through time-domain convolution which is a multiplication in frequency given as below,

$$X\left(e^{j\omega}\right)=S\left(e^{j\omega}\right)H_1\left(e^{j\omega}\right)$$

$H_1(z)$ is a lowpass filter whose gain is 1 and cutoff frequency is $\frac{\pi}{M}$. The relation between y[n] and x[n] can be written as,

$$y[n]=x[Mn],\ n \text{ integer}$$

The above equation in frequency domain is,

$$Y\left(e^{j\omega}\right)=\frac{1}{M}\sum_{k=0}^{M-1}X\left(e^{j\frac{(w+2\pi k)}{M}}\right)$$

A.2. Interpolation

Interpolation increases the sampling rate by a factor of L and obtains the samples between data points. This is effectively expansion in time and compression in frequency. Hence an image spectrum is observed after compression which should be removed through a filter. Therefore, interpolation is composed of an expander followed by a filter. The block diagram of interpolation is given in Fig. 2.

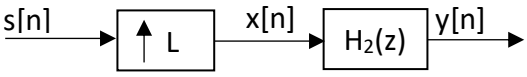


Fig. 2. Interpolation by a factor of L.

The relation between s[n] and x[n] can be given as below,

$$x[n]=\begin{cases} s\left[\frac{n}{L}\right], & n=0,\pm L,\pm 2L,\ldots \\ 0, & otherwise \end{cases}$$

Above expression in frequency domain can be expressed as,

$$X\left(e^{j\omega}\right)=S\left(e^{j\omega L}\right)$$

The relation between x[n] and y[n] can be established easily in frequency as,

$$Y\left(e^{j\omega}\right)=X\left(e^{j\omega}\right)H_2(e^{j\omega})$$

$H_2(z)$ is a lowpass filter whose gain is L and cutoff frequency is $\frac{\pi}{L}$.

A.3. Sampling Rate Change by a Rational Number

Usually, sampling rate change by a factor of a rational number is desired. In this case, decimation and interpolation in Fig. 1 and Fig. 2 should be combined. Since cascaded two filters is equivalent to a single filter, the block diagram for this system is given as in Fig.3.

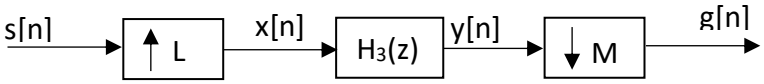


Fig. 3. Sampling rate change by a rational factor.

In the above system, $H_3(z)$ is a lowpass filter whose gain is L, cutoff frequency, ω_c, is the minimum of the both filters, i.e.,

$$\omega_c = \min\{\omega_{c1}, \omega_{c2}\}.$$

B. Digital Phase-locked Loop, DPLL

Phase-locked loops (PLLs) are widely used in a variety of fields including signal processing, communications, multimedia, etc. PLL is usually used for carrier phase and frequency tracking as well as clock recovery. While analog PLLs are still used, digital PLLs become more popular due their simplicity, robustness and cost effectiveness. PLL structure can be analysed using both a nonlinear structures and linear approximations. In our case, linear model is preferred due to its simplicity.

PLL structure is composed of a phase detector, a loop filter and a voltage-controlled oscillator (VCO). Figure 4 shows a simple PLL structure in nonlinear and linear forms.

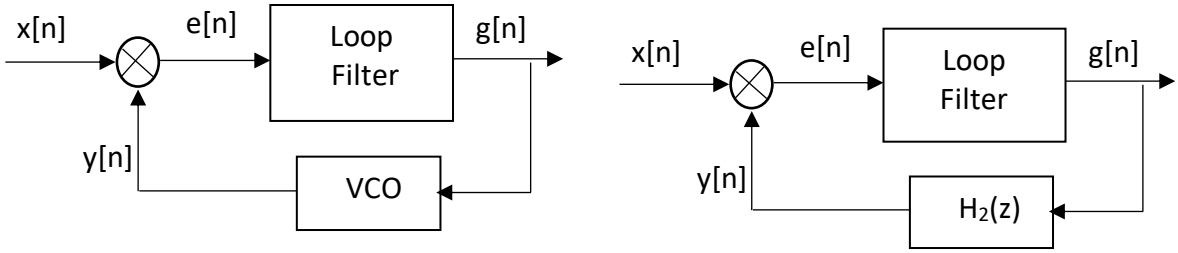


Fig. 4. A simple PLL structure. A.Nonlinear Structure B. Linear Structure

In Fig. 4.a, the phase detector is implemented as a multiplier and low pass filter. e[n] is the error signal representing the phase difference. The above structure multiplies the input signal x[n] and the generated sinusoid y[n] to obtain the phase difference which is then filtered by the loop filter to obtain a quantity which determines whether the phase/frequency of the VCO is increased or decreased.

Let x[n] and y [n] be given as follows, then e[n] can be written as,

$$x[n] = \sin(2\pi f_0 t + \theta)$$

$$y[n] = \cos(2\pi f_0 t)$$

$$e[n] = \frac{1}{2}(\sin(2\pi 2 f_0 t + \Theta) + \sin(\Theta))$$

Once e[n] is lowpass filtered to remove the twice frequency component, g[n]=sin(Θ)≈Θ. Hence the phase of the VCO is fed by g[n]. Note that if there is a small frequency offset between x[n] and y[n], it can also be tracked by PLL as long as the error is within the loop bandwidth.

The loop filter can be an IIR filter of the following form, i.e.,

$$H(z) = \frac{az - 1}{z - 1}$$

where a is the loop gain. VCO can be represented as a digital controlled oscillator, DCO, and it can be written as,

$$H_2(z) = \frac{c}{z - 1}$$

Note that both of the IIR filters have a pole at z=1 and are unstable in strict sense. Nevertheless, it is assumed that the system is operated at a frequency other than DC and used in a narrow band. While the above information is given for completeness, implementation of the DPLL in this experiment can be performed in a different way. For example, VCO in our case is replaced by a sinusoidal waveform generator as a LabVIEW module which has both frequency and phase inputs.

The input signal, x[n] can have a time varying frequency and phase, i.e.

$$x[n] = A\cos(\omega_c(n)n + \theta(n))$$

The VCO output, y[n], should match with both frequency and phase drifts, i.e.,

$$y[n] = B\cos(\omega_d(n)n + \phi(n))$$

PLL has a lock range out of which the system can no longer follow the input signal frequency. Usually this lock range is small and it is determined by the loop filter. In our PLL implementation, we use a first order PLL structure where the loop filter has zero order. In other words, loop filter in our case is FIR moving average filter whose coefficients are all 1/M where M is the length of the filter. This filter is equivalent to the mean operator in MATLAB or LABVIEW.

2) Read the section **"Changing Sampling Rate Using Discrete-Time Processing"** in Discrete Time Signal Processing by Oppenheim, Schafer.

3) In this question, you will observe the **change in sampling rate in MATLAB** by listening to the sound of a chirp signal with different sampling rates. **Don't forget to attach all the codes in MATLAB to your preliminary work**.

a) Generate a **chirp signal in 0-5 s** range with sampling **rate 8000 samples/s**, **start frequency 500 Hz** and **end frequency 1000 Hz**. Use ***"sound"*** command in MATLAB to **listen to** this chirp signal with **sampling frequency 8000 Hz (it is the sampling frequency used by "sound" command for sending the signal out to the speaker)**. Please be aware that the name of ***"sound"*** command may differ in different versions of MATLAB. **Write down** the **duration** of the sound you heard.

b) Now, use ***"sound"*** command to **listen to** the chirp signal you generated in **part a)** with **sampling frequency 4000 Hz (it is the sampling frequency used by "sound" command for sending the signal out to the speaker)**. Notice the **pitch** of the chirp sound has been **lowered** compared to part a). **Explain the reason** for this. Furthermore, **write down** the **duration** of the sound you heard**. Is there any change in comparison to part a)? Explain the reason for your answer.**

c) Use ***"sound"*** command to **listen to** the chirp signal you generated in **part a)** with **sampling frequency 16000 Hz (it is the sampling frequency used by "sound" command for sending the signal out to the speaker)**. Notice the **pitch** of the chirp sound has been **increased** compared to part a). **Explain the reason** for this. Furthermore, **write down** the **duration** of the sound you heard. **Is there any change in comparison to part a)? Explain the reason for your answer.**

d) Now, we will explore **sampling rate change on the chirp signal** generated in part a) using ***"resample"*** command. Use this function to **increase the sampling rate** of the chirp sequence in part a) **by 2**. Call this new sequence **"y"**. Use ***"sound"*** function to **listen to y** with **sampling frequency 8000 Hz (it is the sampling frequency used by "sound" command for sending the signal out to the speaker)**. Which sound in **part a), b), and c),** is this sound **similar to**? **Explain the reason**. Also, **write down** the **duration** of the signal and **comment on it**.

e) Use ***"sound***" function to **listen to y** with **sampling frequency 16000 Hz (it is the sampling frequency used by "sound" command for sending the signal out to the speaker)**. Which sound in **part a), b), and c)**, is this sound **similar to**? **Explain the reason**. Also, **write down** the **duration** of the signal and **comment on it**.

f) Now, use **"resample"** function to **decrease the sampling rate** of the chirp sequence in part a) **by 2**. Call this new sequence **"z"**. Use ***"sound"*** function to **listen to z** with **sampling frequency 4000 Hz (it is the sampling frequency used by "sound" command for sending the signal out to the speaker)**. Which sound in **part a), b), and c)**, is this sound **similar to**? **Explain the reason**. Also, **write down** the **duration** of the signal and **comment on it**.

4) In this question, you will explore **decimation** and **upsampling** functions in **LabVIEW**.

a) Generate a constant array consisting of 7 digits of your student I.D. number.

b) Construct the following block diagram. Note that in Fig. 5, the constant array consists of the digits of 1234567. This is the example I.D. number. In your own code, you should replace this array with the one consisting of your I.D. number digits.

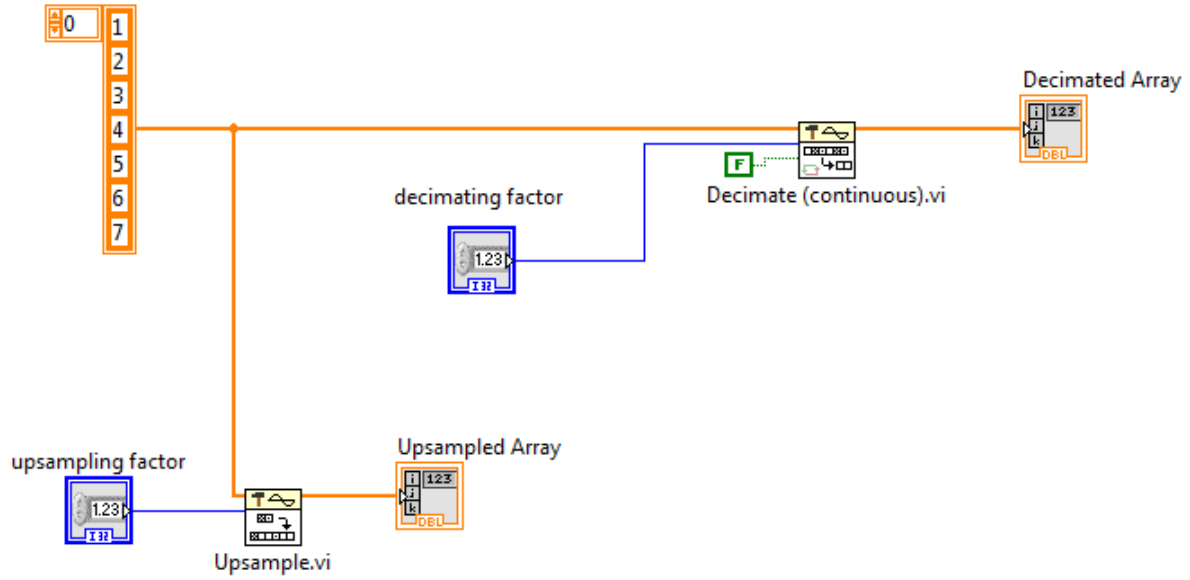


Fig. 5. Decimating and upsampling.

c) Try different factors for decimating and upsampling and observe the decimated and upsampled arrays. **Attach the Front Panel screenshot** for **decimating and upsampling factors as 2**. **All the elements of output arrays should be visible.**

d) Which blocks in Fig. 1 and 2 do the functions “Decimate (continuous).vi” and “Upsample.vi” correspond to?

e) Now, change the block diagram in Fig.5 as shown in Fig. 6 by adding lowpass filters. Again, your constant input array should be the digits of your I.D. number.

f) Try different factors for decimating and upsampling and observe the decimated and interpolated arrays. **Attach the Front Panel screenshot** for **decimating and upsampling factors as 2**. **All the elements of output arrays should be visible.**

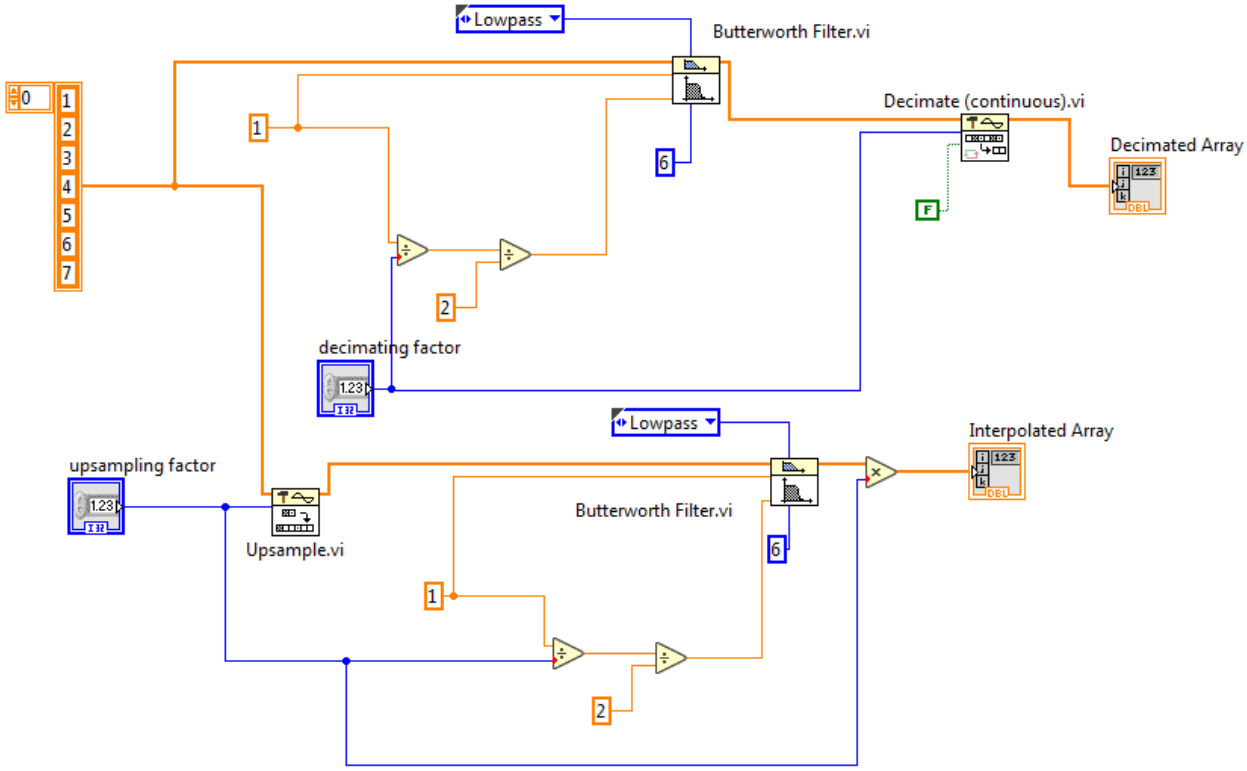


Fig. 6. Decimating with prefiltering and interpolation.

g) Why do we give the low cut-off frequencies of the filters as ***"1/(2*decimating factor)"*** and ***"1/(2*upsampling factor)"*** where **sampling frequency** is **1** ?

h) Now, change the ***constant 1*** **(sampling frequency) to 1000 for both filters. Do the output arrays change? Why?**

III. Experimental Work

PART 1

MATLAB Programming Tasks

a) Write a code in MATLAB to implement **decimation** operation. Given the input signal, and decimation factor, M, the code obtains the output. The **input** and **output** signals are **plotted** both in **time** and **frequency**.

b) Repeat **a)** for **interpolation**.

c) Write a code for sampling rate change with a **rational factor**. **Plot** the **input** and **output** both in **time** and **frequency**.

d) Implement a **digital PLL** structure. In this case, assume that there is a **sinusoid** where **you would lock onto its frequency and phase.** You should be able to **change the frequency and phase of this sinusoid.** The second sinusoid is an **internally produced** one. It may have a constant frequency and you can use decimation and interpolation to match its frequency with the first sinusoid. For this purpose, you need to **detect the frequency of the first sinusoid by using FFT magnitude.** Then **determine the decimation and interpolation factors.** After the frequency is of the second sinusoid is brought to the lock range of the PLL, PLL works on lock onto the fine frequency and phase. You should **plot all the critical signals in time and frequency**.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 4 PART 2

In this experiment, you will implement decimation, interpolation, sampling rate change by rational factor, and Phase Locked Loop (PLL) in LabVIEW. You can use your part 1 code, to get an idea about these implementations in LabVIEW.

1. Create sine wave by using **Sine Wave.vi**. It is shown in Fig.1. Set the amplitude to 1. Create controls for samples and frequency. **Frequency input** should be **normalized** with **sampling frequency**. See help of Sine Wave.vi for more details. For **sampling rate change** operations, you can take phase value as 0. **Do not forget** to place your block in a **while loop.**

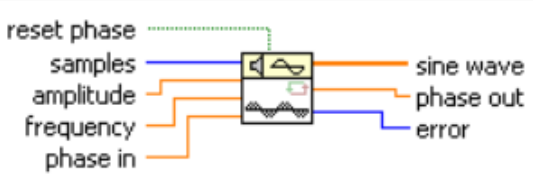


Fig.1. Sine Wave.vi

2. **Decimate** the sinus wave. For this purpose, you can use **Decimate (Single Shot).vi** and **Butterworth Filter.vi** in the right order. You should **set** the filter type as **low-pass filter.** You should also arrange the **cut-off frequency** properly by considering the **sampling frequency.** You can use help of **Butterworth Filter.vi,** and related preliminary work part of the experiment manual to understand how to arrange the cut-off frequency input properly. Also arrange the **filter order properly,** so that you do not have artifacts in the time domain decimated signal.
3. **Plot decimated and input** waveforms on the same graph. You can use **buildwaveform.vi** for this purpose. **Also plot frequency content of decimated and input** waveforms on the same graph. In order to have symmetric frequency spectrums, you can use **shift** option of **FFT.vi** which is shown in Fig.2.

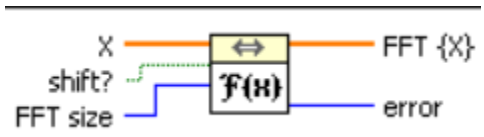


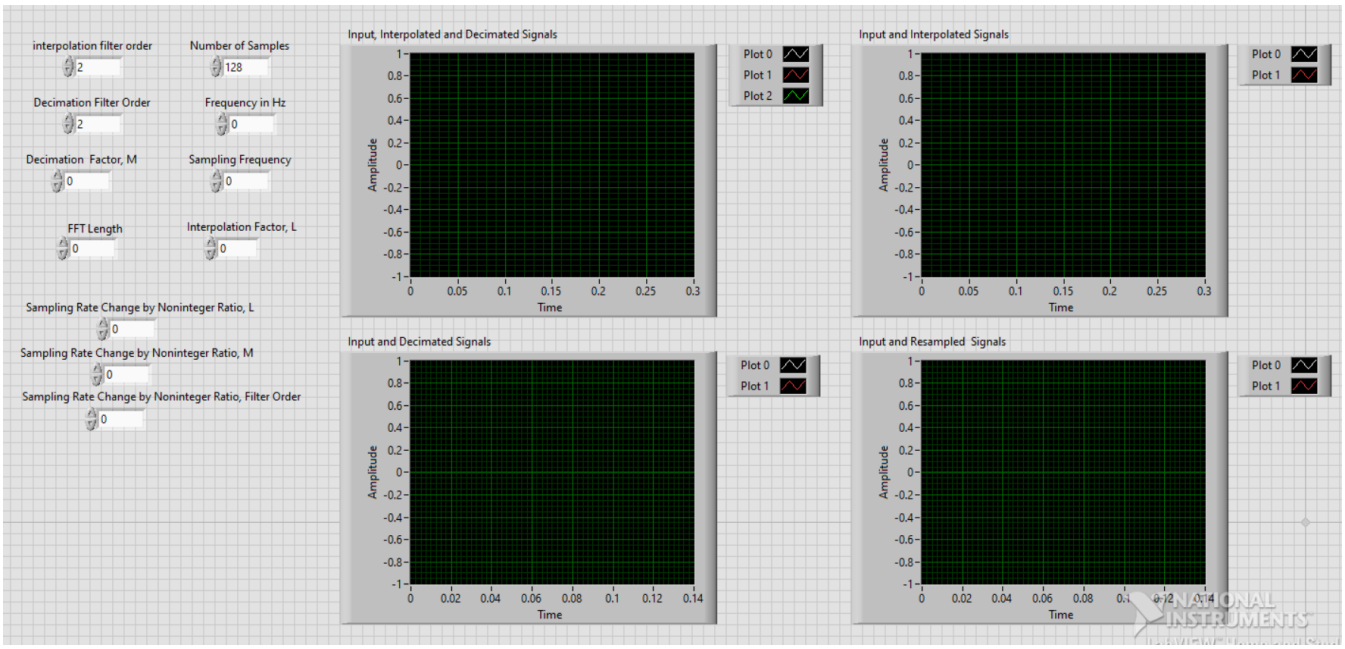
Fig. 2. FFT.vi

4. **Interpolate** the sinus wave. For this purpose, you can use **Upsample.vi** and **Butterworth Filter.vi** in the right order. You should **set** the filter type as **low-pass filter.** You should also arrange the **cut-off frequency** properly by considering the **sampling frequency.** Arrange the **filter order**

properly, so that you do not have artifacts in the time domain interpolated signal. **Create filter order as control** so that you can change it and observe its effects while the program runs.

5. **Plot interpolated and input** waveforms on the same graph. You can use **buildwaveform.vi** for this purpose. **Also plot frequency content of interpolated and input** waveforms on the same graph. In order to have symmetric frequency spectrums, you can use **shift** option of **FFT.vi** which is shown in Fig.2. Also choose **FFT size properly,** so that you get correct FFT for the interpolated signal.
6. **Change the sampling rate by rational factor** by using **decimate (single shot).vi, upsample.vi and Butterworth filter.vi.** Create new controls **for interpolation factor, decimation factor and filter order for this case, i.e., controls in this step and in step 2-4 are different. Again** arrange the **cut-off frequency, filter order, i.e., filter parameters**, properly, so that you observe the right waveform. For the **cut-off frequency**, you may use "**select" operation which is placed under comparison in functions palette.**
7. **Plot resampled and input** waveforms on the same graph. You can use **buildwaveform.vi** for this purpose. **Also plot frequency content of resampled and input** waveforms on the same graph. In order to have symmetric frequency spectrums, you can use **shift** option of **FFT.vi** which is shown in Fig.2. Also choose **FFT size properly,** so that you get correct FFT for the resampled signal.
8. Also plot **input, decimated and interpolated waveforms** in the same plot.

Your front panel must look like in Fig. 3.



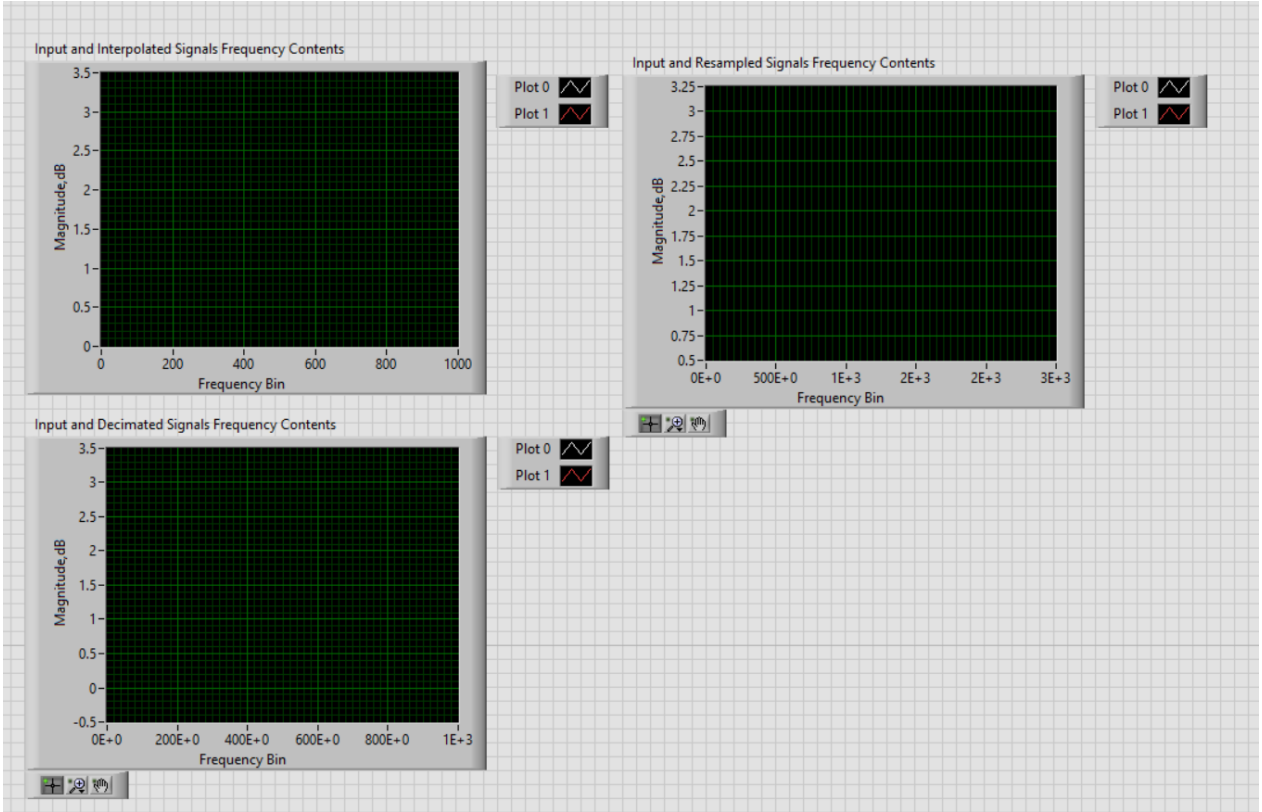


Fig.3. Example Front Panel for Decimation, Interpolation and Sampling Rate Change by Rational Factor

9. You must implement **PLL** in LabVIEW. For this purpose, **first** generate a sine wave with the desired frequency as described in 1. **Note that** this time your phase should be **control input.** Also **note that phase** should be in degrees. See **sine wave.vi** help for detailed explanation.
10. Estimate the **frequency of this sine wave** by using **FFT magnitude.** You may consult to your **MATLAB codes** to estimate **frequency** from **FFT magnitude.**
11. **Create** another **sine wave** by using **Sine Wave.vi** which will serve as your **VCO**. Note that in PLL, we use **cosine wave** for the implementation. You can get a **cosine wave** by using **sine wave.vi and manipulating its phase with 90 degrees.**
12. You should have a **constant frequency and changing phase** for your **VCO.** Resultantly **output waveform of VCO** will have **same frequency** at each iteration. Set your VCO frequency to estimated frequency in 10.
13. **Loop gain and Loop Filter Length** should also be control **inputs.**
14. For the low-pass filter you can use **Mean.vi.**
15. **Implement PLL** as described in **experiment manual.** You can also follow the **same steps as in your MATLAB** implementation.
    Hint: Shift registers, array subset.vi, insert into array.vi, array size may be useful.
16. You can **add wait (ms)** into your **while loop** to see the locking more clearly, as shown in Fig. 4.

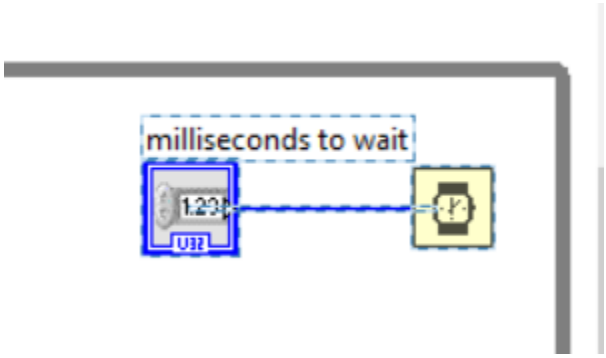


Fig.4. Wait(ms)

17. **Plot generated and locked waveforms both in time and frequency domain** in graphs. Your front panel must look like as in Fig. 5.

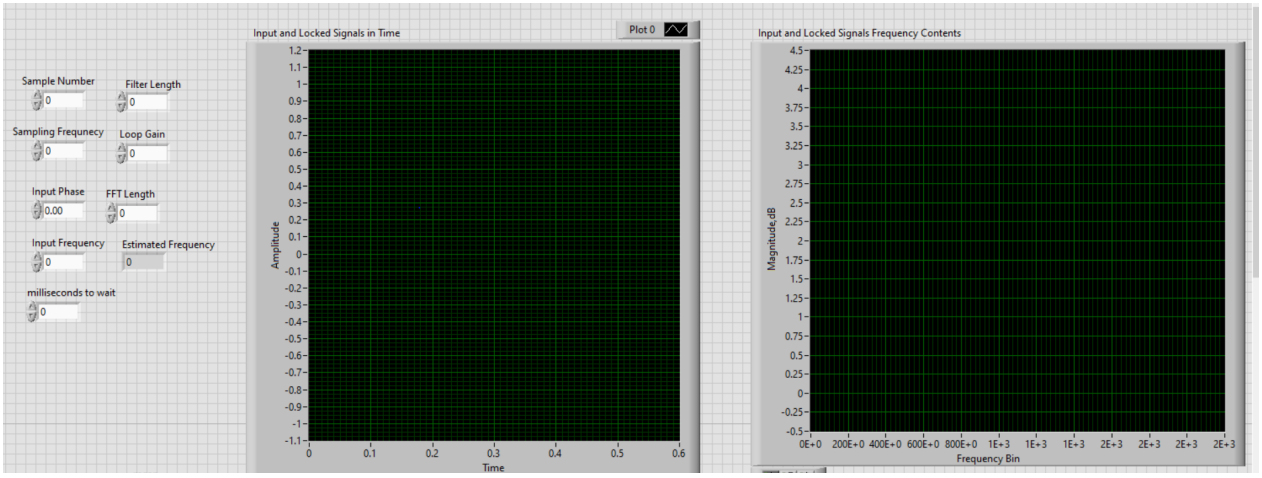


Fig.5. Front Panel for PLL

Do not forget to place screenshots of all your Block Diagrams and front panel. Also upload your VI files to ODTUClass.

TASKS

1) Decimate the sine signal with the specified decimation factor. **Attach plots of original and decimated signals. Also attach their frequency contents.**
2) Interpolate the sine signal with the specified interpolation factor. **Attach plots of original and interpolated signals. Also attach their frequency contents.**
3) Resample the sine signal with the specified resampling factor. **Attach plots of original and resampled signals. Also attach their frequency contents.**
4) **Plot input, decimated and interpolated signals in the same graph.**
5) Construct the PLL structure**. Only for this part** fix the VCO frequency to a given value, i.e., **it is not determined by the estimated frequency**. Gradually **increase the** input frequency with 1Hz steps. Determine the **frequency** where PLL is **not locked any longer.** Note this value in your **reports.**
6) **Return to your original frequency setting, where VCO frequency is determined by the estimated frequency**. **Change loop gain from 0.05 to 2 with 0.2 steps.** Determine the value of loop gain where the PLL does not lock on the input signal. **Comment on the effect of small and large loop gain.**
7) **Change the length of the filter.** Observe if there is any effect of filter length. **Comment** on your results.
8) **Change the phase input** and see whether PLL can achieve locking for different phase values. **Attach** plots for **phase 0** and **60 degrees**.
9) **Comment** on the optimality of your frequency detection method.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 5. OPTIMUM FILTERING: FIR WIENER FILTER IMPLEMENTATION FOR NOISE REMOVAL

1. Introduction

In this experiment, optimum filtering concept is investigated. Optimum filtering uses the statistical characteristics of the input signal to extract the useful information. Hence, some statistical information regarding to the input and the desired signal should be known in order to apply this type of filters.

2. Preliminary Work

- Read the following information on optimum filtering.

2.1. Optimum Filtering: Wiener Filters

The most common types of filter encountered in practice are deterministic filters in the form of a lowpass, highpass, etc. While these filters are easily designed and used, they are far from performing in the best manner especially when there is some information regarding to the signal statistics. Optimum filters perform the best with respect to a certain optimality criteria and they are designed using the signal statistics. While there are several other optimality criteria, we concentrate on a special one, namely the mean square error, MSE. In the following part, the theoretical background for MSE optimum filters or Wiener Filters is presented.

2.2. Derivation of Wiener Filters

The problem for MSE optimum filters can be outlined as follows. Given the input sequence, $x[n]$, desired response, $d[n]$ and the form of the filter structure, i.e.,

$$y[n] = \sum_{k=0}^{\infty} h[k]x[n-k], \quad n = 0,1,... \qquad (1)$$

the target is to find filter coefficients, $h[k]$, such that the MSE is minimized. Note that, y[n] is the Wiener filter output. While it is possible to have IIR Wiener filters, FIR Wiener filters are considered in the following parts for simplicity. The definitions for the error and MSE are given as follows, i.e.,

$$e[n] = d[n] - y[n] = d[n] - \sum_{k=0}^{\infty} h[k]x[n-k]$$
$$MSE = J = E\{e[n]^2\} \quad (2)$$

where $E\{.\}$ is the expectation operator. In order to find the optimum coefficients, derivative of J with respect to $h[k]$ should be equated to zero, i.e.,

$$\nabla_k J = \frac{\partial J}{\partial h_k} = 2E\left\{e[n]\frac{\partial e[n]}{\partial h_k}\right\} = -2E\{e[n]x[n-k]\} = 0 \quad (3)$$

Above is the principle of orthogonality. In other words, J attains its minimum if the estimation error $e[n]$ is orthogonal to the input, $x[n]$. If we insert, $e[n]$ into (3), we obtain,

$$E\left\{\left(d[n] - \sum_{i=0}^{\infty} h[i]x[n-i]\right)x[n-k]\right\} = 0$$
$$\sum_{i=0}^{\infty} h[i]E\{x[n-i]x[n-k]\} = E\{d[n]x[n-k]\} \quad (4)$$
$$\sum_{i=0}^{\infty} h[i]R_x[k-i] = r_{dx}[k], \quad k = 0,1,2,....$$

In the above equation, $R_x[.]$ is the autocorrelation function of the input and $r_{dx}[.]$ is the cross-correlation function of the desired and input signals. Let $\mathbf{R_x}$ denote the autocorrelation matrix of the input where $R_x[k-i]$ is the k^{th} row and i^{th} column element of $\mathbf{R_x}$. (Here, index of columns and zeros of autocorrelation matrix are assumed to start from index 0). Let $\mathbf{r_{dx}}$ be the cross-correlation vector of the desired and input signals where $r_{dx}[k]$ is the k^{th} element of $\mathbf{r_{dx}}$, again starting from 0 index. The last equation in (4) is called as the Wiener-Hopf equation and can be more compactly written in terms of matrix-vector notation as,

$$\mathbf{R_x h} = \mathbf{r_{dx}} \quad (5)$$

where $\mathbf{h}$ is the vector whose elements are $h[i]$.
The minimum MSE for the Wiener filtering is given as,

$$MSE_o = \sigma_e^2 = R_d[0] - \mathbf{h}^{\mathrm{T}}\mathbf{r_{dx}} = R_d[0] - \sum_{m=0}^{P-1} h[m]r_{dx}[m] \quad (6)$$

where $R_d[0]$ is the 0th term of the autocorrelation sequence of the desired signal, and P is the length of the FIR Wiener filter. Hence (6) can be used to compute theoretical minimum MSE. It is also possible to compute least squares error which is a deterministic counterpart of MSE which is computed by using signal samples, i.e.,

$$LSE = \frac{1}{N}\sum_{n=0}^{N-1} e[n]^2 \qquad (7)$$

2.3. Applications of Wiener Filters

Wiener filters can be used in a variety of applications. Filtering of a signal in noise (or noise removal), prediction of a signal in noise, smoothing of a signal in noise, and linear prediction are some examples of these applications. In this experiment, two applications namely the filtering of a signal in noise and prediction are considered. In the following part, examples of such applications are given. However, first a signal with a known correlation function is discussed.

2.3.1. Innovations Process

Innovations process is obtained as the output of a filter when the filter input is white Gaussian noise.

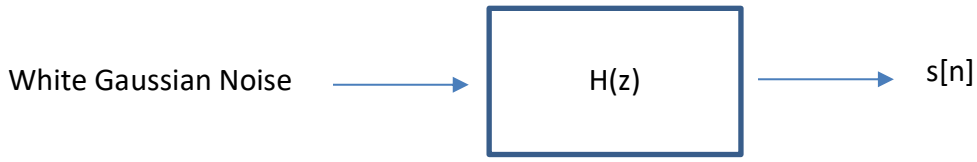


Fig.1. Generation of the innovations process, s[n].

Let $H(z)$ be selected as,

$$H(z) = \frac{1}{1-0.2z^{-1}} \qquad (8)$$

Then it is possible to show that the autocorrelation sequence for $s[n]$ is given as,

$$R_s[m] = \frac{1}{1-(0.2)^2}(0.2)^{|m|} = \frac{1}{0.96}(0.2)^{|m|} \qquad (9)$$

Assume that signal is corrupted by white Gaussian sequence $v[n]$ where $R_v[m] = \sigma_v^2\delta[m]$. Therefore observed signal, $x[n]$, is given as,

$$x[n] = s[n] + v[n] \qquad (10)$$

Noise is uncorrelated with the signal $s[n]$.

2.3.2. Filtering of a signal in noise

In this case, desired signal is $d[n]=s[n]$ and $R_d[m]=R_s[m]$. Note that the problem is as follows. Given the noisy signal $x[n]$, its correlation function, $R_x[m]$ and the cross-correlation $r_{dx}[m]$, find the MSE optimum filter coefficients $h[k]$ such that the filter output is the desired signal. The desired filter length is given as $P=3$. It is possible to write $r_{dx}[m]$ and $R_x[m]$ as follows, i.e.,

$$r_{dx}[m]=E\{s[n]x[n-m]\}=E\{s[n](s[n-m]+v[n-m])\}=R_s[m]=\frac{1}{0.96}(0.2)^{|m|} \qquad (11)$$

$$R_x[m]=E\{(s[n]+v[n])(s[n-m]+v[n-m])\}=R_s[m]+R_v[m]=\frac{1}{0.96}(0.2)^{|m|}+\sigma_v^2\delta[m] \quad (12)$$

Note that $v[n]$ and $s[n]$are assumed to be uncorrelated. Using the Wiener-Hopf equations in (4) or (5), we obtain,

$$\begin{bmatrix} R_x[0] & R_x[-1] & R_x[-2] \\ R_x[1] & R_x[0] & R_x[-1] \\ R_x[2] & R_x[1] & R_x[0] \end{bmatrix} \begin{bmatrix} h[0] \\ h[1] \\ h[2] \end{bmatrix} = \begin{bmatrix} r_{dx}[0] \\ r_{dx}[1] \\ r_{dx}[2] \end{bmatrix} \quad (13)$$

Note that correlation matrix is Hermitian symmetric, i.e., $R_x[-1]=R_x^*[1]$. Once the numerical values for the above expression are used, Wiener filter coefficients can be easily obtained through matrix inversion, i.e., $\mathbf{h_{opt}}=\mathbf{R_x^{-1}r_{dx}}$. Theoretical MSE is obtained from (6).

2.3.3. Prediction of a signal in noise

In this case, the problem is to predict the signal samples before K-steps. Hence the desired signal is $d[n]=s[n+K]$ where K=2 is selected for simplicity. $R_x[m]$ does not change and it is the same as in (12). $r_{dx}[m]$ changes and it is given below,

$$r_{dx}[m]=E\{s[n+2]x[n-m]\}=E\{s[n+2](s[n-m]+v[n-m])\}=R_s[m+2]=\frac{1}{0.96}(0.2)^{|m+2|} \quad (14)$$

It is possible to write a similar expression to (13) in this case and find the Wiener filter coefficients. MSE in this case is

$$MSE=R_d[0]-\sum_{m=0}^{2}h[m]\frac{1}{0.96}(0.2)^{|m+2|} \quad (15)$$

3. Experimental Work

PART 1

MATLAB Programming Tasks

- Write a program for optimum filtering.

i) The inputs for this program are:
- length of the input signal, N,
- length of the optimum filter, P,
- pole of the innovations filter, a, $\left(H(z) = \frac{1}{1 - az^{-1}} \right)$
- standard deviation of the noise, σ_v,
- the prediction step, P_{step}.

ii) Take P_{step}=0 for filtering of a signal in noise.

iii) Obtain the desired signal as shown in Fig. 1 and equation (8).

iv) Add noise to obtain $x[n] = s[n] + v[n]$.

v) Find $\mathbf{R_x}$, $\mathbf{r_{dx}}$ and compute **h** from (5).

vi) Obtain Wiener filter output, y[n], by convolving x[n] with h[n]. Obtain the LSE from the samples using (7). Compare it with MSE using the equation (6). Plot s[n], x[n], y[n], and e[n].

vii) Design a lowpass filter using windowing technique (or Parks-Mcclellan algorithm) and filter x[n] with this new filter to obtain g[n]. Compute LSE for g[n] and compare with the previous results. Explain your findings.

viii) Repeat iii-vii for prediction of a signal in noise by selecting P_{step}=2. Explain your plots and findings. What happens when P_{step} is increased?

PART 2

Real-time Programming Task

In this part, you need to program LabVIEW for the optimum filtering similar to the MATLAB implementation.

The front panel for the Experiment 5 may look like the one shown in Fig. 2. The input parameters for the system are described in MATLAB section and they are the same in MyRIO implementation. ***"milliseconds to wait"*** is added to slow down the execution in order to have a better visualization of the waveforms. The Wiener filter coefficients are reported as in Fig.2. The theoretical MSE, LSE of the Wiener filter and LSE of the lowpass Parks-Mcclellan (PM) filter are shown as well. The parameters of the PM filter are also presented. The two plots show the time response of the Wiener filter as well as the error signals.

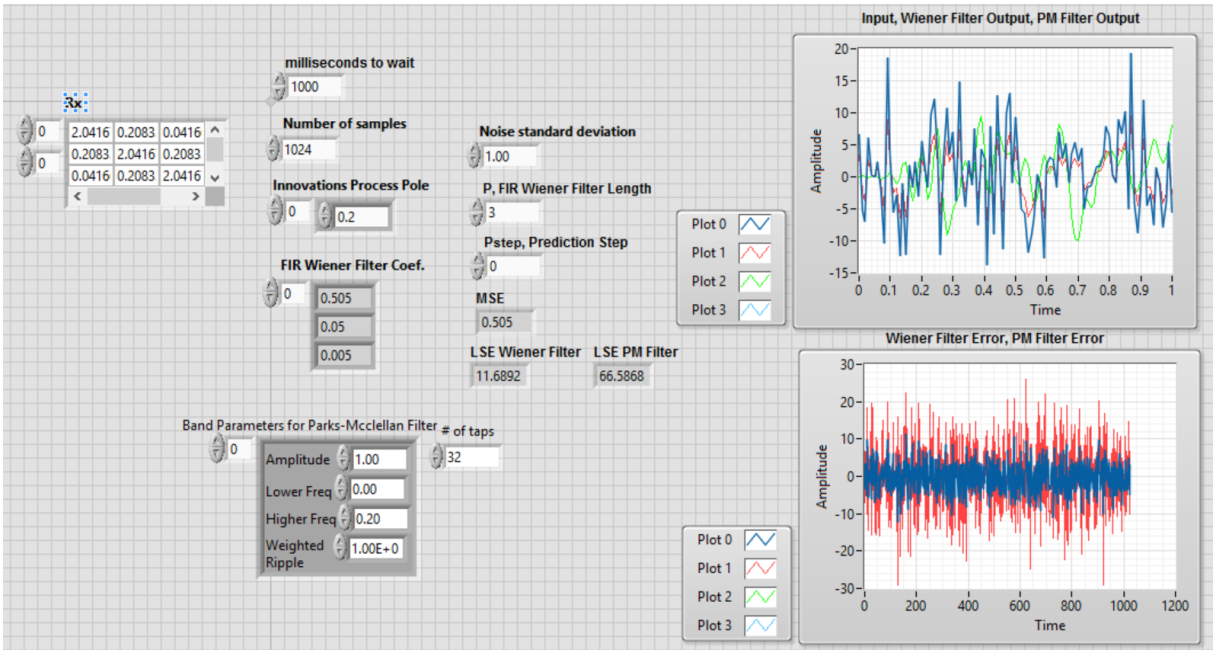


Fig. 2. Experiment 5 front panel.

- You can use ***Gaussian White Noise.vi*** function in Fig. 3 to generate random Gaussian white noise.

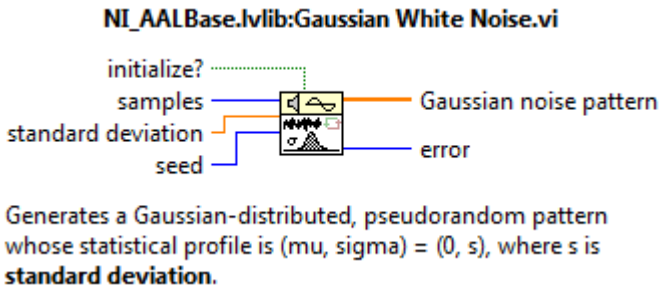


Fig. 3. Gaussian White Noise.

- You can use ***IIR Fiter.vi*** and ***FIR Filter.vi*** functions shown in Fig. 4, respectively.

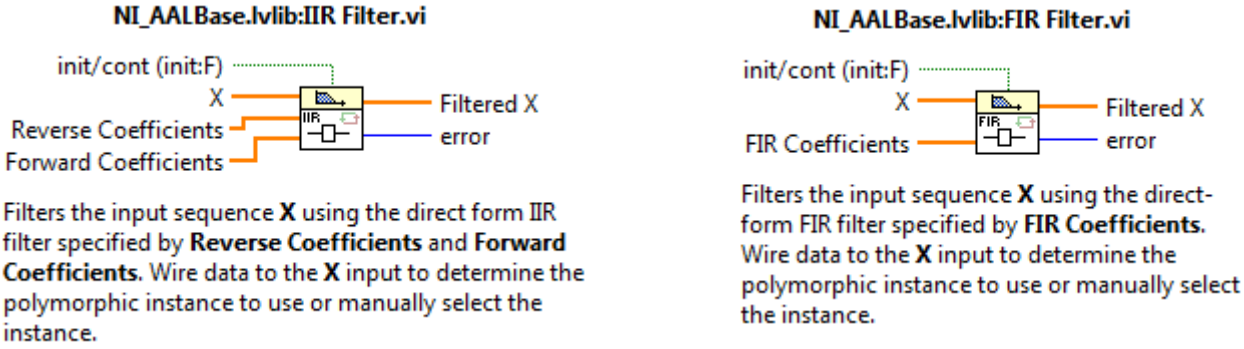


Fig. 4. IIR and FIR filters.

- You can use ***Parks-Mcclellan.vi*** function as shown in Fig. 5.

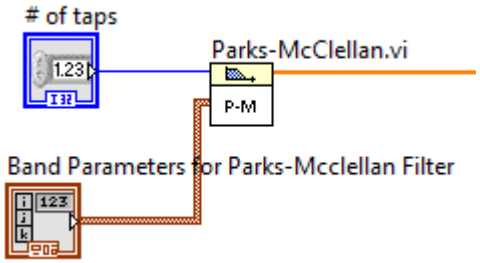


Fig. 5. Parks-McClellan filter.

- In order to generate autocorrelation matrix of the input, you can use ***Create Special Matrix.vi*** shown in Fig. 6 with ***matrix type*** as ***Toeplitz***.

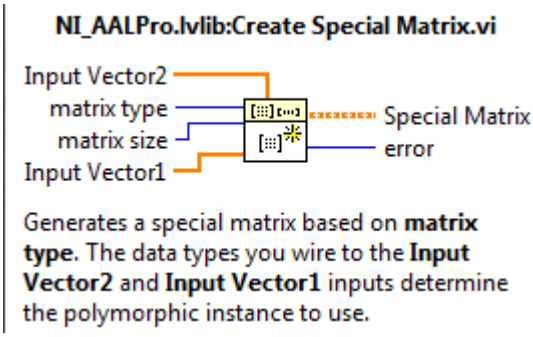


Fig. 6. Create Special Matrix.vi.

- You can use ***Inverse Matrix.vi*** and ***General Matrix-Vector Product.vi*** functions shown in Fig. 7 for matrix operations in optimum filtering.

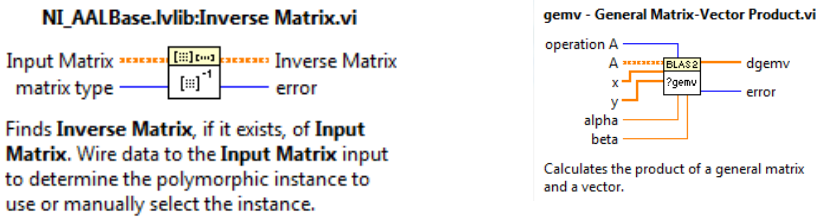


Fig. 7. Inverse Matrix.vi and General Matrix-Vector Product.vi.

Programming Tasks

a) Write the LabVIEW program for optimum filtering as described above.

b) In addition to the two plots in Fig. 2, add two plots for the frequency characteristics of the Wiener filter and PM filter.

c) Let the innovations process pole, a=0.8, Wiener filter length=3 and P_{step}=0. Obtain the filter coefficients, MSE and LSE values. Also include the waveform plots in your report. Can you design a PM filter which gives better LSE than Wiener filter?

d) Let P_{step}=2 and repeat c. Explain the differences between the results obtained in c and d.

e) Increase P_{step} to 100. Explain your observations.

f) Now increase the length of the Wiener filter one by one by noting the MSE. Explain the MSE result as the filter length increases.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 6. SYSTEM IDENTIFICATION WITH ADAPTIVE PROCESSING, DESIGN AND IMPLEMENTATION OF LMS FILTER

I. Introduction

In this experiment, adaptive filters are considered. Adaptive filters are used in many fields which require tracking the input statistics. Some examples of applications are system identification, equalization, linear predictive coding, spectrum estimation, signal detection, noise canceling, echo suppression, beamforming, etc. In this experiment, FIR adaptive filter structure is used together with the least mean square (LMS) adaptation algorithm for system identification. Evaluation of adaptive filters is done by considering factors such as rate of convergence, misadjustment, tracking performance, robustness, computational complexity, structure and numerical properties.

II. Preliminary Work

1) Read the following information on optimum filtering.

6.1. Adaptive Filtering: Least Mean Square, LMS, Algorithm

When there is sufficient statistical information, optimum solution for a variety of problems can be found by solving the Wiener-Hopf equations. Hence one can easily find the filter coefficients by

$$\mathbf{h}_{opt} = \mathbf{R}_x^{-1}\mathbf{r}_{dx}$$

An alternative for finding the optimum filter coefficients and optimum solution or output is to use an iterative algorithm that starts from an initial point and move towards the optimum in steps. One advantage of the adaptive approach is its ability to track the slowly changing statistics of the input signal. While there are a variety of adaptation algorithms, LMS algorithm is considered in this experiment due to its simplicity. Furthermore, the adaptive filter is assumed to have a transversal (or FIR) form. The structure of an adaptive filter is given below.

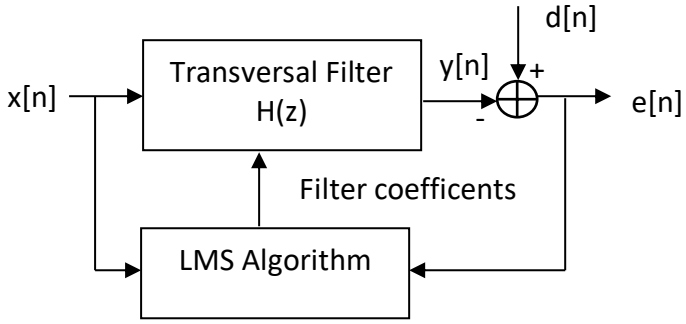


Fig. 1. Adaptive filter structure.

The input signal is filtered by H(z) and the adaptive filter output, y[n], is obtained. At each time-step, n, the error, e[n]=d[n]-y[n] is computed where d[n] is the desired response. LMS algorithm takes e[n] and the input, x[n], to compute the adaptive filter weights at the next iteration.

6.2. Derivation of the LMS algorithm

It is assumed that there are M filter coefficients which need to be set by the LMS algorithm. LMS algorithm tries to minimize the error between the desired response, d[n] and the adaptive filter output, y[n]. The cost function that is minimized is the sample error function. Hence LMS algorithm is a deterministic type of an adaptive filter. The cost function is given as,

$$J[n]=e^2[n]=(d[n]-y[n])^2=\left(d[n]-\sum_{k=0}^{M-1}h[k]x[n-k]\right)^2 \quad (1)$$

J[n] is a quadratic function of the adaptive weights, h[k], and it has a single minimum. At each iteration, J[n] is reduced by moving h[k] proportional to $-\frac{\partial J[n]}{\partial h_k}$. Hence the weight update equation is given as,

$$h_k[n+1]=h_k[n]-\hat{\mu}\frac{\partial J[n]}{\partial h_k} \quad (2)$$

At each time instant n, the kth coefficient's current value is changed and assigned to the n+1 time value. The derivative expression in (2) can be written as,

$$\frac{\partial J[n]}{\partial h_k}=\frac{\partial e^2[n]}{\partial h_k}=2e[n]\frac{\partial e[n]}{\partial h_k}=-2e[n]x[n-k] \quad (3)$$

Inserting (3) in (2), we obtain,

$$h_k[n+1]=h_k[n]+\mu e[n]x[n-k],\ \ k=0,1,\ldots,M-1 \quad (4)$$

where $\mu=2\hat{\mu}$.

6.3. System Identification

Adaptive filters can be used to identify the impulse response of an unknown system and track its characteristics in time if it changes. This is possible if the input and output signals of the unknown system are available. Fig. 2 shows the use of adaptive filter for finding the impulse response of the system.

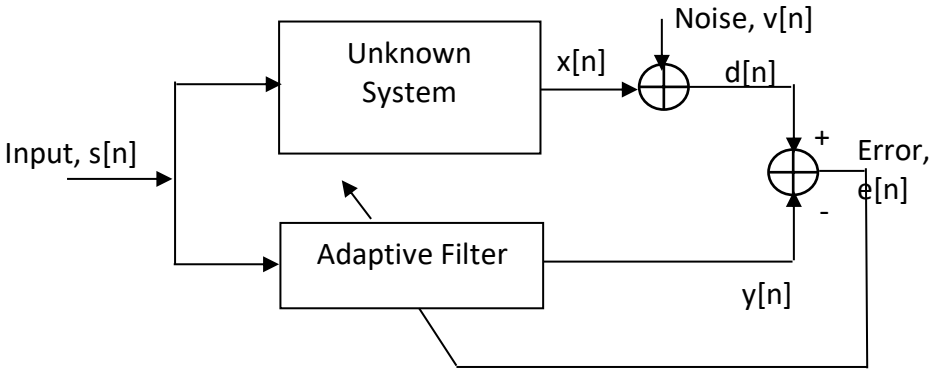


Fig. 2. System identification with an adaptive filter.

As it is seen in Fig. 2, unknown system output is x[n]. x[n] is noise corrupted by v[n] and the observable desired signal d[n] is obtained. When there is no noise, i.e., v[n]=0, d[n]=x[n]. In this case, adaptive filter converges more securely. The difference between d[n], and the output of the adaptive filter, y[n], is the error, e[n], which is used to update the adaptive filter coefficients. The adaptation causes y[n] to approach d[n] in time. A small step size ensures a stable response but convergence may be slow. In this case, *misadjustment* is expected to be small compared to a larger step size. A larger step size may lead faster convergence but *misadjustment* is larger and the response may be unstable leading to progressively larger output samples.

6.4. Interference and Noise Cancellation

Adaptive filters can be used for interference and noise cancellation. The advantage of adaptive filters is the adaptation to the time varying signal characteristics as opposed to other techniques. The structure for noise cancellation share the same idea as in system identification. Fig. 3 shows the structure of the noise cancellation system.

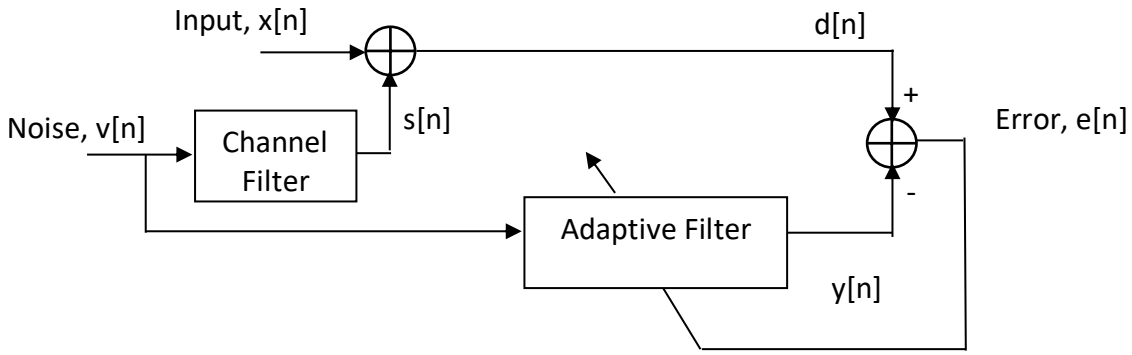


Fig. 3. Interference and noise cancellation system structure.

As it is seen in Fig. 3, desired signal x[n] is corrupted by the noise, s[n], which is the output of channel filter. The target is to clean d[n] from the noise by subtracting the adaptive filter output, y[n], from d[n]. The function of the adaptive filter is to converge to the channel filter coefficients such that it replicates the interference signal. As the adaptive filter converges, e[n] gets closer to x[n].

PART 1

6.5. MATLAB Programming Tasks

6.5.1. System Identification in MATLAB

In this part, the structure in Fig. 2 is implemented in MATLAB.

a) Generate a white Gaussian signal, s[n] with unit variance. Select a FIR filter with N coefficients. Filter the signal and obtain the output signal, x[n]. Add white Gaussian noise to the output signal and obtain d[n]. The noise standard deviation of the white Gaussian noise v[n] is a parameter.

b) Design an adaptive filter whose **parameters** are the **filter length**, **step size**, **input signal**, and the **error signal**. The output of the adaptive filter is y[n]. Input signal is s[n], error signal is e[n]=d[n]-y[n]. Use LMS algorithm for the adaptation of the filter coefficients.

c) The **program outputs** are the **adaptive filter coefficients (or weights)**, and **the least squares error**, i.e.

$$LSE = \frac{1}{K}\sum_{k=0}^{K-1} e^2[k]$$

d) Determine the converge speed for the selected step size by plotting $e^2[n]$ versus n.

e) Change the adaptive filter length to observe its effects. Explain your observations.

f) Change the FIR filter length, N and explain its effects.

g) Change the step size and explain its effect on the convergence speed as well as misadjustment which is the final error after convergence.

h) Change the standard deviation of noise, v[n], and explain your observations. Note that noise free case, v[n]=0, should return the best performance.

6.5.2. Noise Cancellation in MATLAB

In this part, the structure in Fig. 3 is implemented in MATLAB for noise removal.

a) Generate a white Gaussian signal, v[n] with unit variance. Select a FIR channel filter with N coefficients. Filter v[n] and obtain the output signal, s[n]. Add s[n] with the input signal, x[n] to obtain noise corrupted signal, d[n]. Note that ultimate target is to clear d[n] from noise to obtain x[n]. The noise standard deviation for v[n] is a parameter.

b) Design an adaptive filter whose **parameters** are the **filter length**, **step size**, **input signal**, and the **error signal**. The output of the adaptive filter is y[n]. Input signal is v[n], error signal is e[n]=d[n]-y[n]. Use LMS algorithm for the adaptation of the filter coefficients.

c) The program outputs are the adaptive filter coefficients (or weights), and the least squares error, i.e.

$$LSE = \frac{1}{K}\sum_{k=0}^{K-1} e^2[k]$$

d) Determine the converge speed for the selected step size by plotting $e^2[n]$ versus n.

e) Change the adaptive filter length to observe its effects. Explain your observations.

f) Change the FIR filter length, N and explain its effects.

g) Change the step size and explain its effect on the convergence speed as well as misadjustment which is the final error after convergence.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 6. SYSTEM IDENTIFICATION WITH ADAPTIVE PROCESSING, DESIGN AND IMPLEMENTATION OF LMS FILTER

PART 2

LabVIEW Programming Tasks

In this part, the programing tasks in MATLAB are realized in LabVIEW. Hence both system identification and noise cancellation are implemented.

1. System Identification

Fig. 1 shows the project structure for the System Identificaiton Part.

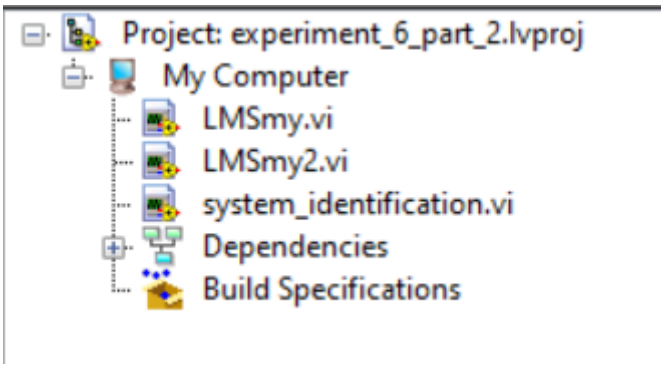


Fig. 1. Project Structure for the System Identification.

As it is seen in Fig. 1, there are three VI's. The two VI's are for the adaptive filter implemenetation using the LMS algorithm, namely **LMSmy.vi** and **LMSmy2.vi**, respectively. Actually, these two are exactly the same. They are named differently to avoid memory corruption when two adaptive filters are called simultaneously to observe the convergence speeds. The other VI is for the system identification.

a) Design an adaptive filter structure in LabVIEW, i.e. **LMSmy.vi**. The VI structure for the adaptive filter using the LMS algorithm is given in Fig. 2. Note that you should be able call this VI from another VI as a **subVI** and hence you should assign input and outputs for this VI appropriately.

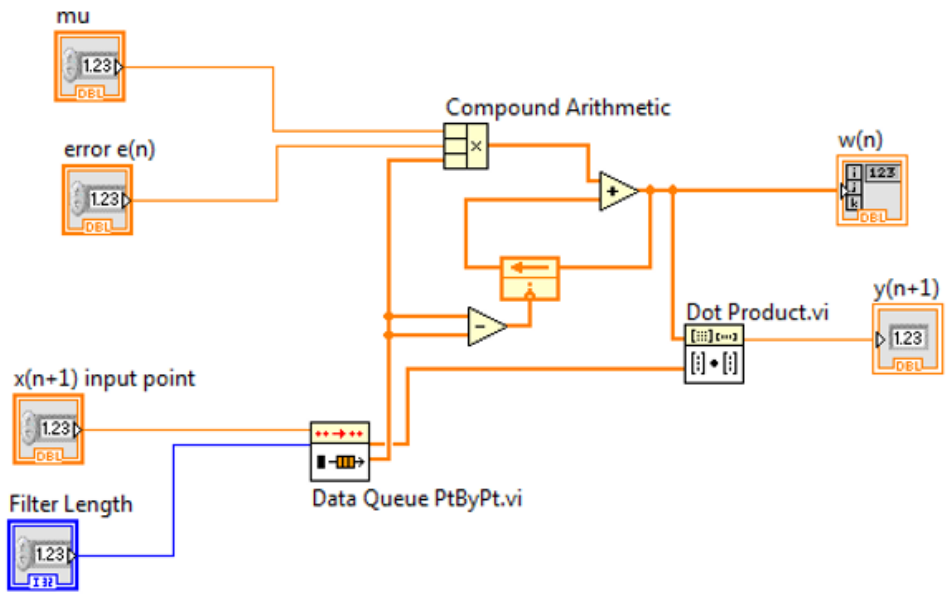


Fig. 2. Adaptive filter structure in LabVIEW.

b) Create another VI, **LMSmy2.vi** which is exactly the same as **LMSmy.vi.** Note that this VI also should be called from another VI as a **subVI** and hence you should assign input and outputs for this VI appropriately.

c) Create a new VI named as **system_identification.vi**. Generate a random input signal, filter it through a filter whose filter coefficients are adjusted from the front panel. Then build the adaptive filter structure as shown in Fig. 3 for obtaining the filter coefficients by the help of the adaptive filter. Note that filter output is corrupted by white noise and the result is the d[n] signal. A part of the LabVIEW structure is given in Fig. 4.

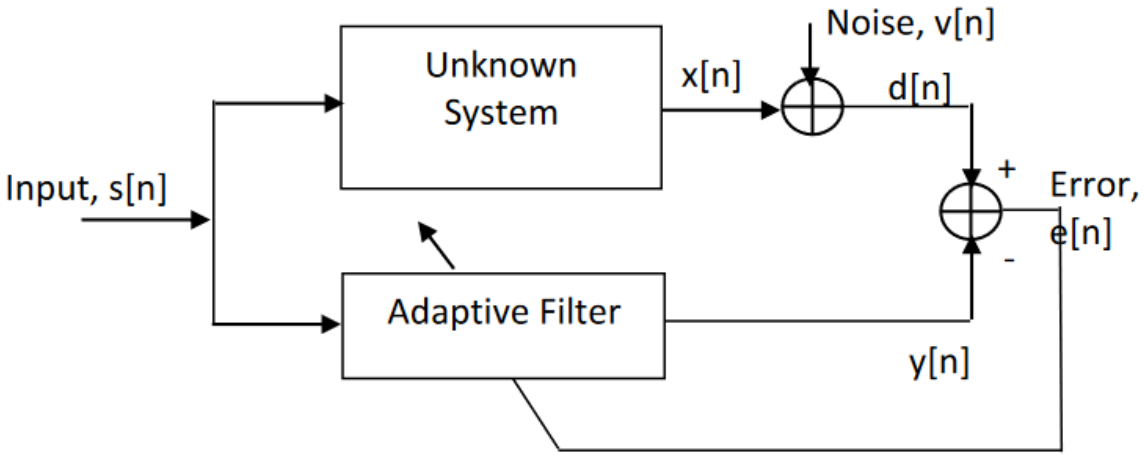


Fig. 3. System identification with an adaptive filter.

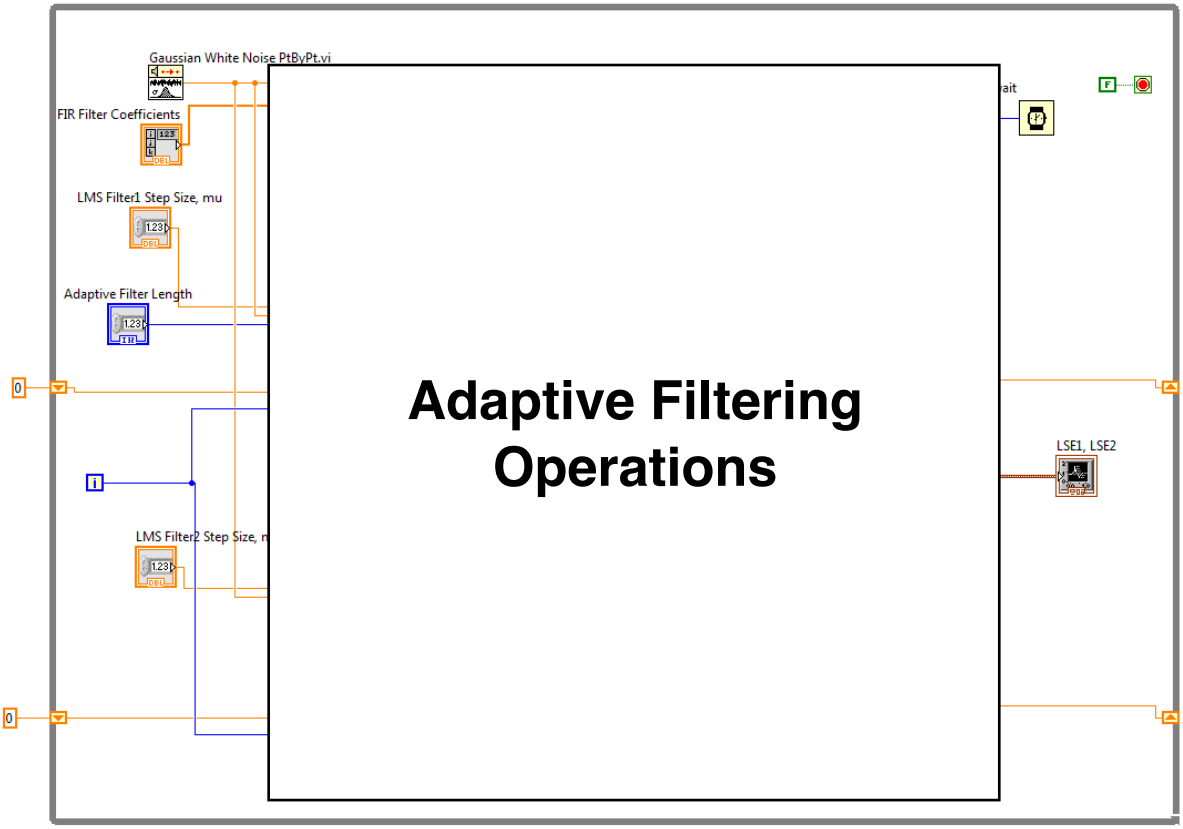


Fig. 4. The system identification structure in LabVIEW.

The front panel for system identification is shown in Fig. 5.

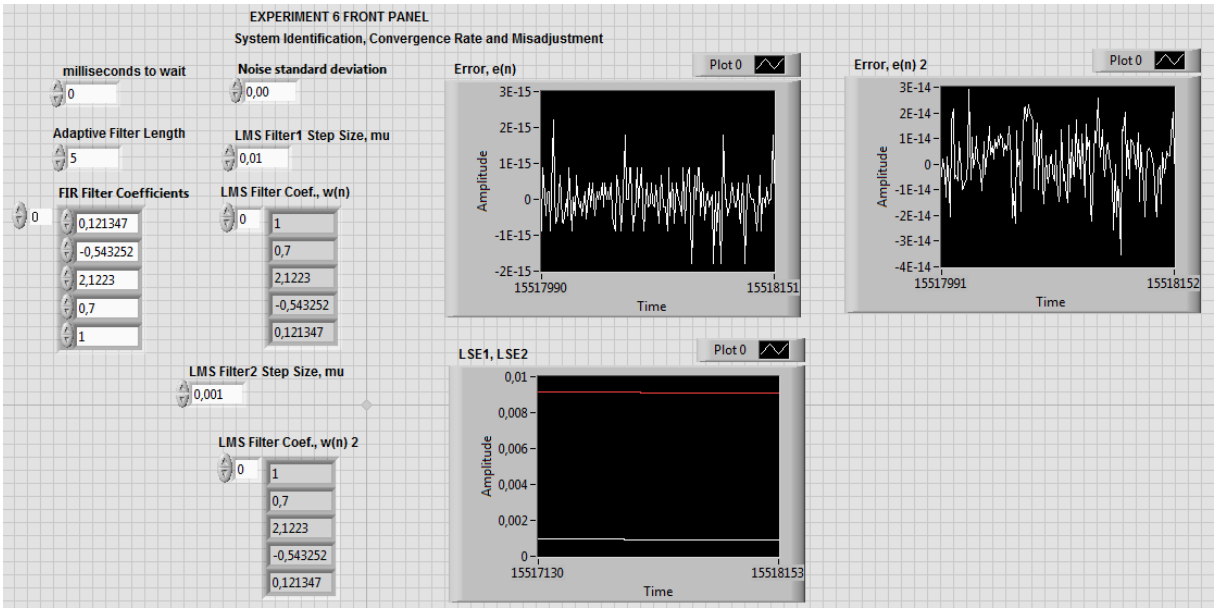


Fig.5. Front panel for system identification in Experiment 6

d) Set the noise standard deviation to zero and run the VI. Check if the FIR filter coefficients are found accurately by the adaptive filter. Note that the coefficients should be the same only their order is different. Set the step sizes for adaptive filters to 0.01 and 0.001 in order to observe the difference between convergence speeds as well as the misadjustments. Explain which filter has faster convergence and which filter has the smallest misadjustment.

e) Set the noise standard deviation to 1 and **repeat d**. Explain your results and observations.

f) Increase the number of coefficients for the first adaptive filter. What is the effect of this? **Explain**.

g) Decrease the number of coefficients for the first adaptive filter. What is the effect of this? **Explain**.

2. Noise Cancellation

In this part, noise cancellation will be implemented. You may implement noise cancellation by following your MATLAB Codes, Part 1 and Part 2 experiment manuals and Fig. 4.

a) Set the standard deviation to 1, and step sizes to 0.01 and 0.001. Observe the outputs. **Attach** their plots to the report.

b) Change noise standard deviation to 2 to see its effect. Observe the outputs. **Compare** your results with **a and comment** on your results.

c) Increase the number of coefficients for the first adaptive filter. What is the effect of this? **Explain**.

d) Decrease the number of coefficients for the first adaptive filter. What is the effect of this? **Explain**.

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 7 Part 1. IMAGE PROCESSING, 2D FFT, FILTERING, EDGE DETECTION

I. Introduction

In this experiment, one of the topics of signal processing, image processing is considered. Image processing uses mathematical operations to process images, or video. In image processing, an image is considered as a two-dimensional signal where x and y coordinates correspond to the location of information. While this treats an image as a continuous signal, it can also be used to represent discrete digital images where x and y are the positive integer indices for pixel values. Hence image is treated as a grid of discrete elements. In this respect, signal processing theory can be applied directly to this two dimensional signal. However the increase in computational complexity prohibits some techniques especially for systems with limited computational resources. In a similar manner, a video can be modeled as a three dimensional signal where the third dimension is time.

The acquisition of images is called as imaging and requires image capturing devices and interfaces. Computer vision is a closely related field where high-level processing methods are used including segmentation, recognition and reconstruction. This leads to the scene analysis mimicking what the human brain does to understand the image.

In this experiment, an introduction to image processing is done including image filtering, edge detection, two-dimensional Fourier transformation, and inverse filtering, etc.

II. Preliminary Work

1) Read the following information on image processing.

1. Image Representation and Filtering

An image can be either a grayscale or color image. A grayscale image is simple to consider. It can be considered as a surface graph with x and y coordinates (or indices) represent the location and z axis represents the intensity of light. Brighter areas correspond to higher z-axis values. Grayscale image depth is the range of pixel values and expressed in terms of the number of bits each pixel value is represented ,i.e., 2^N. The higher is the depth, the larger is the memory required to store the image. 1024x768 8-bit grayscale image would require Memory=1024x768x8 bits.

Color images are represented using either Red-Green_blue (RGB) or Hue-Saturation_Luminance (HSL) models. When 8-bit RGB channels are considered, a color image would require Memory=1024x768x8x3 bits.

Image filters either suppress or enhance data with respect to a certain criterion. Examples of filter operations are edge detection (or highpass filtering), and smoothing (or lowpass filtering). Image filters can be linear or nonlinear. Linear filters can be implemented through convolution hence their 2D impulse responses are also called as convolution kernels. An example of such a filter is given in Fig. 1 where a smoothing (or lowpass filter) is considered. While Fig. 1 (a) and (b) are the same filters, the filter in (a) may lead to image clipping. In other words, when the value of a pixel and its closest neighbours are added, resulting value may exceed the image depth leading to clipping. The filter in Fig. 1 (b) does not have this problem and its values add up to one indicating that the clipping will not occur.

0	1	0
1	1	1
0	1	0

(a)

0	1/5	0
1/5	1/5	1/5
0	1/5	0

(b)

Fig. 1. Smoothing filters. (a) and (b) are the same filter except a scale factor. (b) does not have image clipping.

Image clipping can be handled during the filter implementation by dividing with the scalar value and hence usually the image filters are represented with their integer values as in Fig. 1 (a) for simplicity. In Fig. 1, 3x3 filter is presented. Larger filters can be used to improve the filter characteristics with the additional computational complexity.

2D convolution of a NxM image, f(n,m), and LxL convolution kernel h(n,m) can be expressed as

$$s(n,m) = f(n,m) * h(n,m) = \sum_{k=0}^{L-1}\sum_{p=0}^{L-1} h(k,p)\, f(n-k, m-p), \quad 0 \le n \le L+N-1,\ 0 \le m \le L+M-1$$

(1)

Convolution in time is multiplication in frequency and the above equation is written in frequency as,

$$S(w_1, w_2) = F(w_1, w_2) H(w_1, w_2), \qquad -\pi \le w_1 \le \pi, \quad -\pi \le w_2 \le \pi \qquad (2)$$

where $S(w_1,w_2)$ is the 2D DTFT of s(n,m). Note that w1 and w2 are continuous variables and cannot be implemented in digital systems. Hence 2D DFT should be used for practical applications. In the following part, the definition of 2D DFT is given.

1.1. 2D DFT

2D discrete Fourier transform is important for image processing applications. In practice, FFT algorithm is used to have the DFT coefficients since it is a computationally efficient algorithm. The 2D DFT of h(n,m) is also a discrete sequence given as,

$$H(k_1,k_2)=\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}h(n_1,n_2)e^{-j\frac{2\pi n_1 k_1}{N_1}}e^{-j\frac{2\pi n_2 k_2}{N_2}},\quad 0\le k_1\le N_1-1,\ 0\le k_2\le N_2-1 \qquad (3)$$

Inverse 2D DFT is written as,

$$h(n_1,n_2)=\frac{1}{N_1N_2}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}H(k_1,k_2)e^{j\frac{2\pi n_1 k_1}{N_1}}e^{j\frac{2\pi n_2 k_2}{N_2}},\quad 0\le n_1\le N_1-1,\ 0\le n_2\le N_2-1 \qquad (4)$$

Note that DFT in (3) can be grouped in two summations and hence 2D DFT can be obtained by two 1D DFT's applied first to the columns and then to the rows. The same does apply for the use of 1D FFT in obtaining 2D FFT.

1.2. Convolution Kernels

Convolution kernel h(n,m) is said to be separable if it can be written as $h(n,m)=h_1(n)h_2(m)$. In this case, 2D convolution can be implemented as two 1D convolutions. First, rows of f(n,m) are convolved with $h_1(n)$ and then columns are convolved by $h_2(m)$. Same idea does apply to Fourier domain processing. Zeros of separable filters are easily found due to the Fundamental Theorem of Algebra which guarantees that there are n zeros of a single variable polynomial whose order is n. Unfortunately such a theorem does not exist for 2D nonseparable filters since they are polynomials with two variables. Therefore analysis of nonseparable convolution kernels is not trivial as in the case of 1D or separable filters.

1.2.1 Gaussian Filter

Gaussian filter can be used for noise removal. It is effectively a lowpass filter. The coefficients of this filter are given in Fig. 2.

1	2	1
2	4	2
1	2	1

Fig. 2. Gaussian kernel.

1.2.2. Gradient Filter

An interesting type of filter is the gradient filter. This filter is especially useful when intensity variations along a certain axis are enhanced. An example of this type of filter is given in Fig. 3 where the enhancement is being done in 45^o axis.

0	-1	-1
1	0	-1
1	1	0

Fig. 3. Gradient filter.

1.2.3. Laplacian Filter

Laplacian filter is an omnidirectional gradient filter. Hence it is a highpass filter. It is usually used for edge detection. There are different variants of Laplace filter. Fig. 4. (a) shows the general form a Laplace filter while Fig. 4. (b) is a special case used for edge detection.

A	B	C
D	X	D
C	B	A

(a)

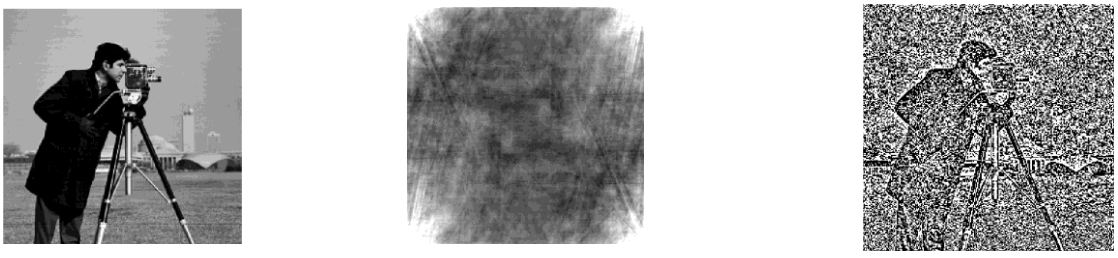
-1	-1	-1
-1	8	-1
-1	-1	-1

(b)

Fig. 4. Laplacian filter.

2. 2D Linear Phase Filters

A 2D image (or signal) can only be completely represented by using its DFT magnitude and phase together. In addition, it is also known that phase information is relatively more important than the magnitude information. In order to understand this, consider the image in Fig. 5. (a).



(a) (b) (c)

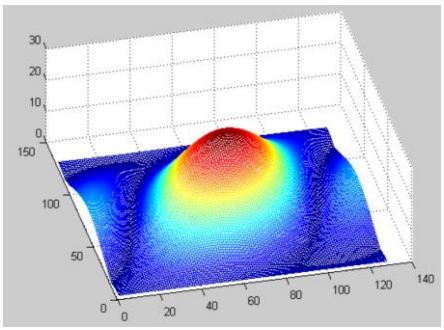
Fig. 5. (a) Original Image, (b) Image reconstructed by magnitude only, (c) Image reconstructed by phase only.

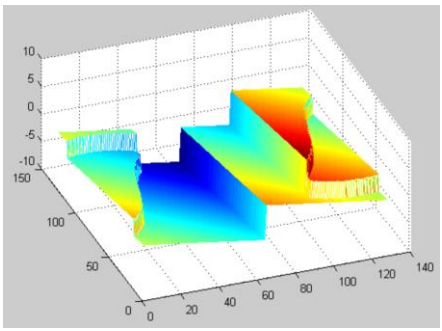
The original image 2D DFT is obtained and only the magnitude is extracted for reconstruction through 2D IDFT. The resulting image is shown in Fig. 5. (b). A similar operation is performed for the phase of 2D DFT by assuming the magnitude as one. Fig. 5. (c) shows the reconstructed image. As it is seen, phase information is more critical in image processing.
In order to preserve the phase information during 2D filtering, linear phase filter should be used. A 2D linear phase should satisfy the following condition, i.e.,

$$h(n_1,n_2)=h(N_1-1-n_1,N_2-1-n_2),\ \ 0\le n_1\le N_1-1,\ 0\le n_2\le N_2-1 \quad (5)$$

The following example is a 2D linear phase filter, with its magnitude and phase characteristics.

1	2	3
4	5	4
3	2	1





(a) (b) (c)

Fig. 6. (a) $h(n_1,n_2)$, (b) $|H(k_1,k_2)|$, (c) $\angle H(k_1,k_2)$.

3. Inverse Filtering

Inverse filtering or deconvolution is used to obtain the original image when its filtered version is given. Inverse filtering problem arises in different situations. A widely known case is about the Hubble Telescope. When Hubble telescope is stationed in Earth's orbit for transferring the star images, it is realized that the images are blurred due to malfunctioning lens. Hence the scientists should find a way to correct the filtering effect to obtain a sharper image. The solution was known for a long time namely the inverse filtering. In its simplest form, the filter that distorts the image is known. Unfortunately, it is unknown in many practical applications.
The inverse filtering problem is given in Fig. 7.

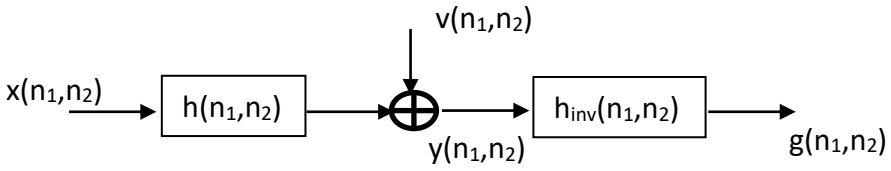


Fig. 7. Inverse filtering problem.

The input image $x(n_1,n_2)$ is filtered by $h(n_1,n_2)$ and corrupted by noise, $v(n_1,n_2)$. Given the output image, $y(n_1,n_2)$, the problem is to obtain an estimate of $x(n_1,n_2)$ by designing an inverse filter $h_{inv}(n_1,n_2)$. In the ideal case where $v(n_1,n_2)=0$ and 2D DFT $H(k_1,k_2)$ is invertible,

$$H_{inv}(k_1,k_2)=\frac{1}{H(k_1,k_2)} \qquad (6)$$

Note that H(k1,k2) is invertible if all of its poles and zeros are inside the unit circle (hence minimum-phase filter). Especially when there is noise and H(k1,k2) is small, there is noise amplification. It is possible to obtain optimum solution in MSE sense. The result is the Wiener inverse (or deconvolution) filter and it is given as,

$$H_{inv}^{Wiener}(k_1,k_2)=\frac{|H(k_1,k_2)|^2}{|H(k_1,k_2)|^2\,H(k_1,k_2)+\dfrac{1}{SNR(k_1,k_2)}} \qquad (7)$$

In (7), $SNR(k_1,k_2)$ is the signal-to-noise ratio for the k_1,k_2 frequency bins. Note that in the absence of this information, it should be estimated. Otherwise, a constant value for SNR results a suboptimum solution. Nevertheless $SNR(k_1,k_2)$=constant is a simple choice used in practice in order to have a rough idea about the performance of the inverse filtering.

2) ***Please include all the plots in your preliminary work!!!***

a) Write the following MATLAB code to plot the image A and its 2FFT magnitude.

- **Comment on** the magnitude of 2FFT of the image A.
- Why do we scale AFFT in line 5?
- Why do we use ***fftshift*** to plot the magnitude of 2FFT?

```
1 –  A=ones(81,81);
2 –  figure
3 –  imshow(A);
4 –  AFFT=fft2(A);
5 –  AFFT=AFFT/max(max(abs(AFFT)));
6 –  figure
7 –  imshow(fftshift(abs(AFFT)));
```

b) Write the following MATLAB code to plot the image B and its 2FFT magnitude.

- **Comment on** the magnitude of 2FFT of the image B.

```
10 –  B=zeros(81,81);
11 –  B(41,41)=1;
12 –  figure
13 –  imshow(B);
14 –  BFFT=fft2(B);
15 –  BFFT=BFFT/max(max(abs(BFFT)));
16 –  figure
17 –  imshow(fftshift(abs(BFFT)));
```

c) Write the following MATLAB code to plot the image C and its 2FFT magnitude.

- **Comment on** the magnitude of 2FFT of the image C.

```
C=zeros(81,81);
C(35:45,:)=1;
figure
imshow(C);
CFFT=fft2(C);
CFFT=CFFT/max(max(abs(CFFT)));
figure
imshow(fftshift(abs(CFFT)));
```

d) Write the following MATLAB code to plot the image D and its 2FFT magnitude.

- **Comment on** the magnitude of 2FFT of the image D.

```
D=zeros(81,81);
for dd=0:8
    D(:,10*dd+1)=1;
end
figure
imshow(D);
DFFT=fft2(D);
DFFT=DFFT/max(max(abs(DFFT)));
figure
imshow(fftshift(abs(DFFT)));
```

e) Write the following MATLAB code to filter the image D with the filter F1.

- What is the type of the filter F1, i.e, lowpass or highpass?
- **Comment on** the effect of the filter F1 on the image D.

```
F1=[ 0 1/5 0; 1/5 1/5 1/5; 0 1/5 0];
DF=filter2(F1,D);
figure
imshow(DF)
```

f) Write the following MATLAB code to plot the image E and its filtered version EF.

- What is the type of the filter F2, i.e, lowpass or highpass?
- What's the special name of the filter F2?
- **Comment on** the effect of the filter F2 on the image E.

```
47 –     E=zeros(81,81);
48 –     E(15:25, 20:40)=1;
49 –     for ee=50:70
50 –       E(ee,60+50-ee:60-50+ee)=1;
51 –     end
52 –     figure
53 –     imshow(E);
54
55 –     F2=[ -1 -1 -1; -1  8  -1;  -1  -1  -1];
56 –     EF=filter2(F2,E);
57 –     figure
58 –     imshow(EF)
```

g) Write the following MATLAB code to plot the image G and its filtered version GF.

- What's the special name of the filter F3?
- **Comment on** the effect of the filter F3 on the image G.

```
61 –     G=rand(81,81);
62 –     figure
63 –     imshow(G)
64
65 –     F3=[ 0 -1 -1; 1  0  -1;  1  1  0];
66 –     GF=filter2(F3,G);
67 –     figure
68 –     imshow(GF)
```

PART 1

MATLAB Programming Tasks

a) Write a ***"expriment7.m"*** file to load two images, ***'cameraman.tif'*** and ***'lena.png'***. Use ***'imshow'*** to display the images.

b) Take the 2D Fourier transform of cameraman in order to visualize the low and high frequency components. Use ***'fftshift'*** to center the DC component.

c) Show that 2D fft, ***'fft2'***, is equivalent to applying ***'fft'*** to rows and columns respectively.

d) Obtain the phase components of ***cameraman***. Assume that the magnitude components are one and use ***ifft2*** to reconstruct the image by using only the phase information as shown in Fig. 5. ***Comment on*** your findings. Try alternatives for the magnitude terms during the reconstruction. One alternative is to use random numbers, i.e. ***rand()***. Another alternative is to use the magnitude of ***lena*** image. Which one gives ***better reconstruction***?

e) Consider the following ***9x9 nonlinear phase lowpass kernel***.

```
h1=1/4248*[92    35    75    96    22    39    70    30    81;
           82    40    38    34   100    49    36     3     3;
           16    66    60    71    57    88    80     8    21;
           82    37    15    93    81    90    98    17    91;
           82     9     4    41    52    92    68    34    82;
           49    37    56    54    84     6    62    98    50;
           82     9    28    88     9    36    73    83    40;
            2    58     2    40    48    38    54    79    19;
           80    77     6    54    58    50    35    60    84;];
```

Design a ***9x9 separable lowpass linear phase filter***, h2=ha'*hb. Let

```
ha=[1 2 3 4 5 4 3 2 1];
hb=[1 1 2 2 3 2 2 1 1];
```

Plot the ***magnitude*** and ***phase*** characteristics of these filters as shown in Fig. 6. Filter the ***cameraman*** and ***lena*** images by these filters and ***comment on*** the phase distortions introduced by the nonlinear phase filter. Note that you may need appropriate scaling for better visualization. You need to implement the filters in two ways. One is convolution in time. The other is the frequency domain implementation by using fft2. Use ***tic-toc*** commands to measure the computation time for each type of filtering.

f) Consider the filter, ***h2***, in ***e)***. Filter the ***lena*** image with this filter and obtain the output ***y(n_1,n_2)*** as shown in Fig. 7 by adding random noise with a known variance. Display the resulting image with ***imshow*** command. Design the inverse filter as it is given in equation (7). You should use an appropriate FFT size to obtain linear convolution. Filter the output, $y(n_1,n_2)$ with the inverse filter. Display the result of the inverse filtering and ***comment on*** the deconvolution performance. Try different noise levels and comment on the reconstruction performance. ***Determine*** the least squares error in reconstruction. Try different images and comment on their reconstruction quality. Change the h2 filter (by changing ha and hb) and ***comment on*** the inverse filtering quality based on the filter characteristics. What is the effect of zeros of ***ha*** and ***hb*** on the reconstruction quality?

MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 7 Part 2. IMAGE PROCESSING, 2D FFT, FILTERING, EDGE DETECTION

Real-time Programming Tasks

In this part, you can use **Edge Detection with 2D Convolution.vi** which you can find in **"Find Examples"** item in the Help Menu. Copy this vi and modify its contents accordingly to implement the operations described in the MATLAB programming tasks.

The project structure in Experiment 7 is given in Figure 1.

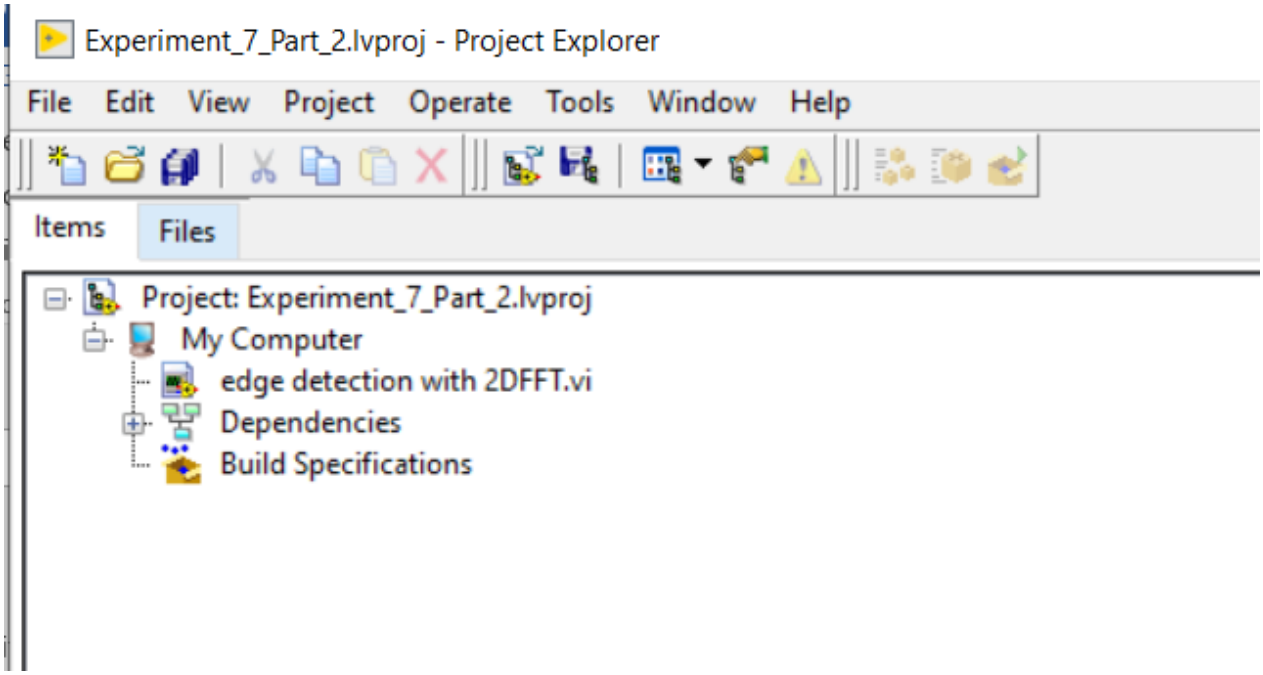


Figure 1. Project structure for Experiment 7.

A part of the front panel of Experiment 7 is given in Figure 2. Note that FFT vi in LabVIEW has an input which you can set as True to obtain "**fftshift**" effect in MATLAB. Also the front panel does not include certain parts that you need to implement and hence given only for guidance.

a) Modify the contents of Edge Detection with 2D Convolution.vi such that you have a front panel similar to Figure 2. A part of the Block Diagram after modification looks like as shown in Figure 3.

b) Obtain the 2D Fourier transform of **cameraman** in order to visualize the low and high frequency components. You can try other image sources such as "**Baboon**", "**Colombia**" and "**Couple**" in order to see the differences between frequency contents. **Comment** on each image and its frequency characteristics.

c) Obtain the phase components of cameraman. Assume that the magnitude components are one and use 2D ifft to reconstruct the image by using only the phase information as shown in Figure 5 (refer Experiment 7 Part 1 Manual). Comment on your findings. Try alternatives for the magnitude terms during the reconstruction. Can you obtain better reconstruction quality with alternative selections?

d) Increase the size of the Convolution Mask in Figure 2 to 9x9. You can easily do that by enlarging the box with mouse and entering numerical values. Repeat **part e)** in MATLAB Tasks. In this case you only need to implement the filtering using **"convolution.vi".**

e) Repeat **part f)** in **MATLAB** Tasks. In this case, you can use the available image sources instead of **Lena**. Try different images and comment on their reconstruction quality. Change the h2 filter and comment on the inverse filtering quality based on the filter characteristics.

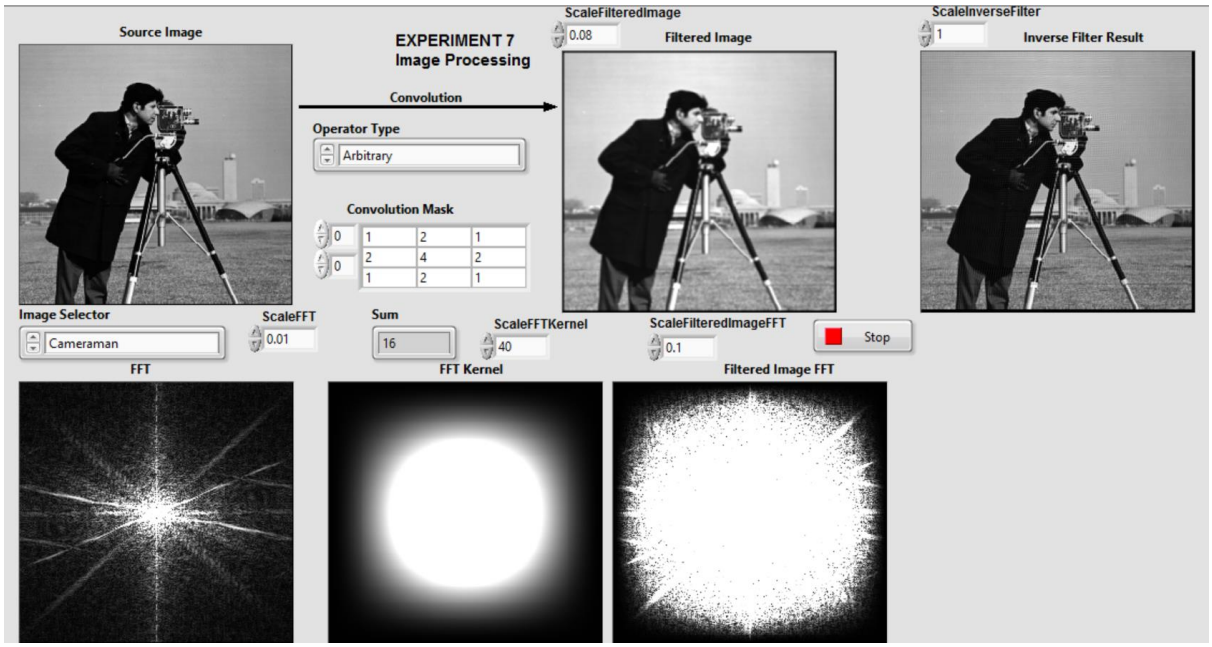


Figure 2. Front panel of Experiment 7.

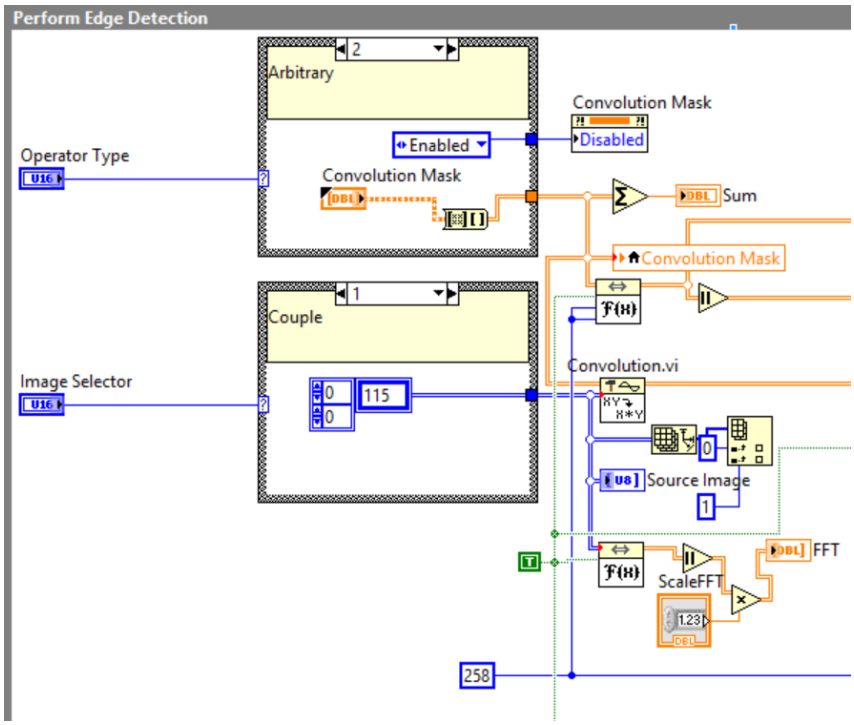


Figure 3. Part of the Block Diagram in Experiment 7.