

## 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/](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.

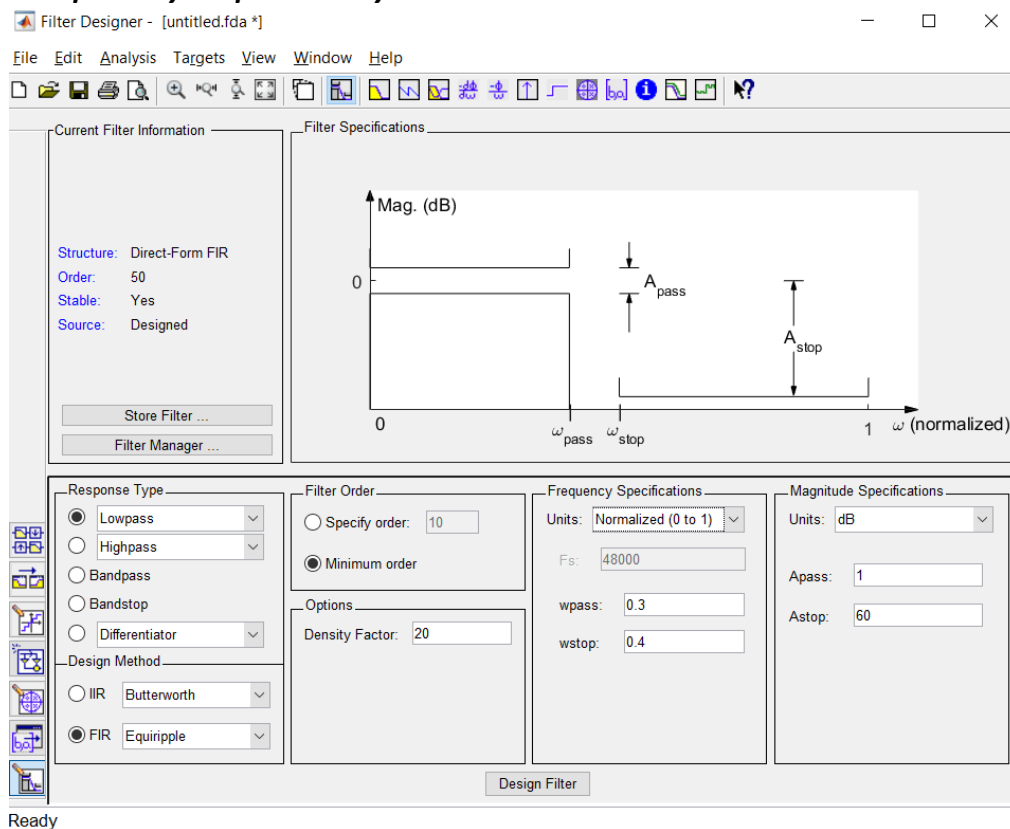
An alternative for computationally **efficient FIR filter** implementation is to design and use **"multiplierless"** filters. The filter coefficients for these FIR filters are powers of two which can be implemented by simple **"bit shift"** operation. Hence these filters can be implemented by

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  **$0.3\pi$**  and  **$0.4\pi$**  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  $\pi$  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 Chebyshev 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.**

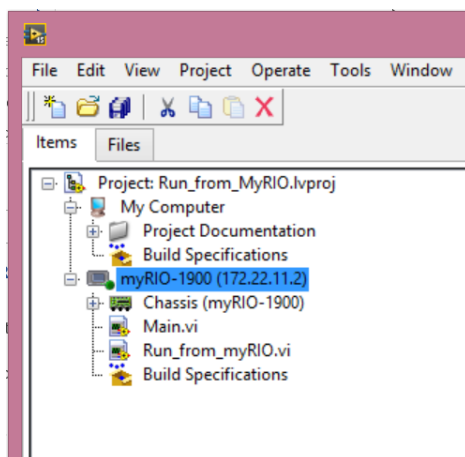
d) In this part, you need to write a MATLAB code to implement **first-order FIR and IIR lowpass filters**. The **input** to these filters should be a **triangular waveform** where the **amplitude is 1** and **frequency is 200Hz** with **sampling frequency 4000 samples/sec**. The first-order filter structures 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

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

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

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

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


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

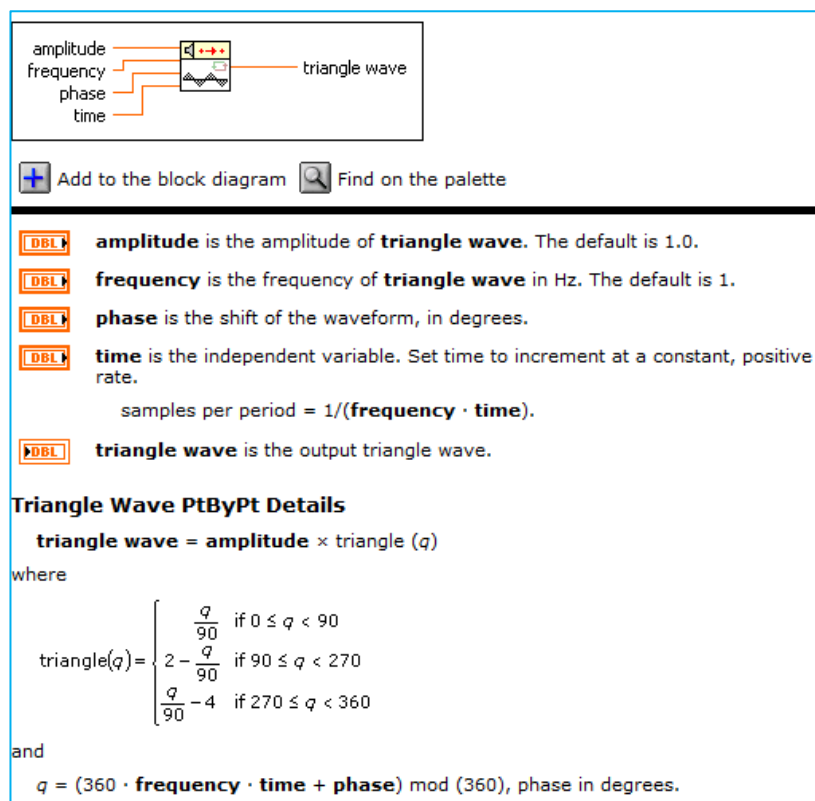
$$\frac{Y(z)}{X(z)} = H(z) = 1 - bz^{-1} \quad (3)$$

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,

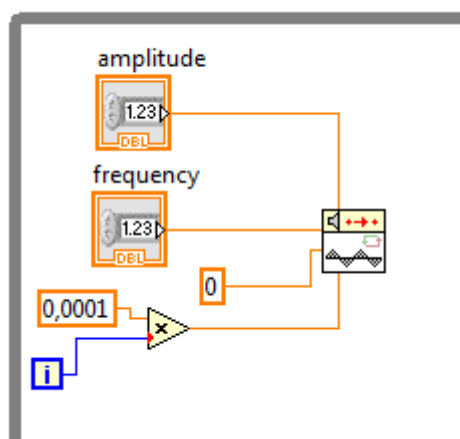
$$y[n] = x[n] - bx[n-1] \quad (4)$$

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.

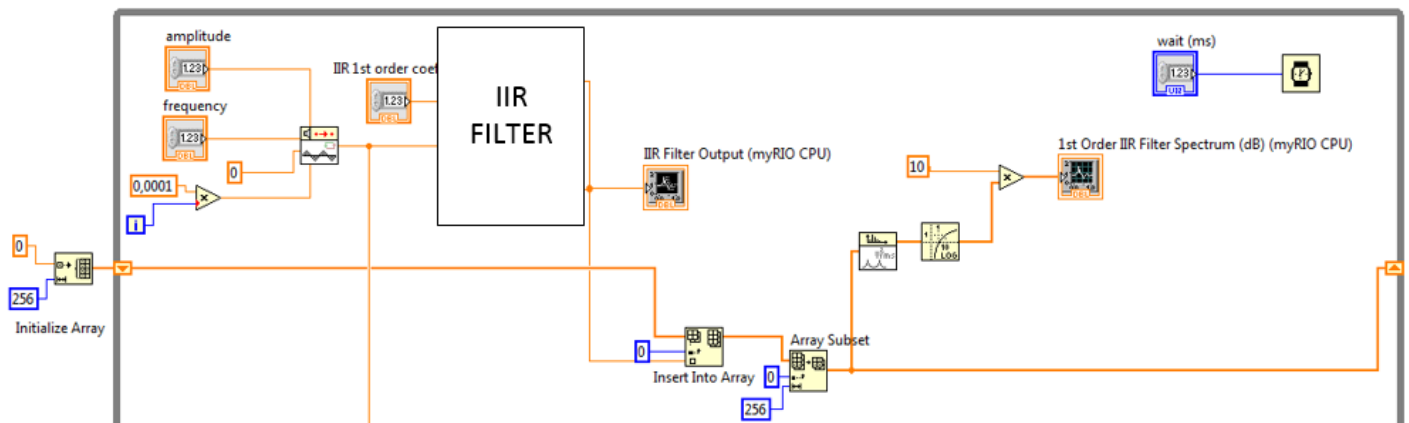
- You need to write your own code to compute the output samples of these filters using (2) and (4). The **inputs on the front panel** will be the **triangular wave amplitude, frequency, IIR and FIR filter coefficients ( $a$  and  $b$ )**.
- i) Generate a triangular waveform using built-in LabVIEW functions.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

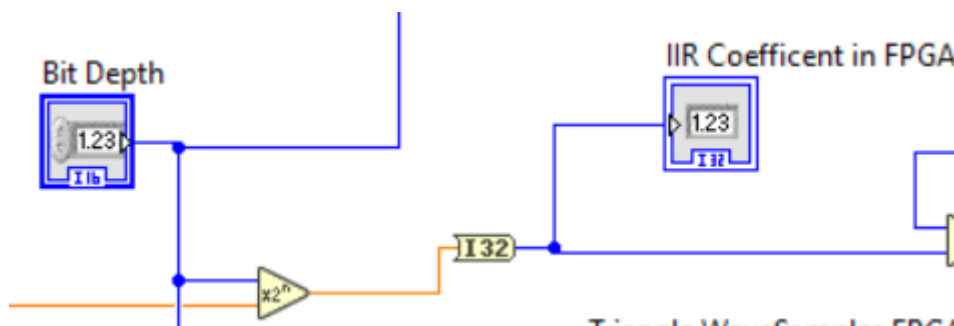


**Fig 5. Processing of the last 256 points.**

- 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  $2^{(\text{Bit Depth})}$  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,

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

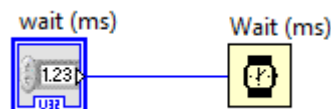
(5) is implemented in the FPGA by (6), i.e.,

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

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).

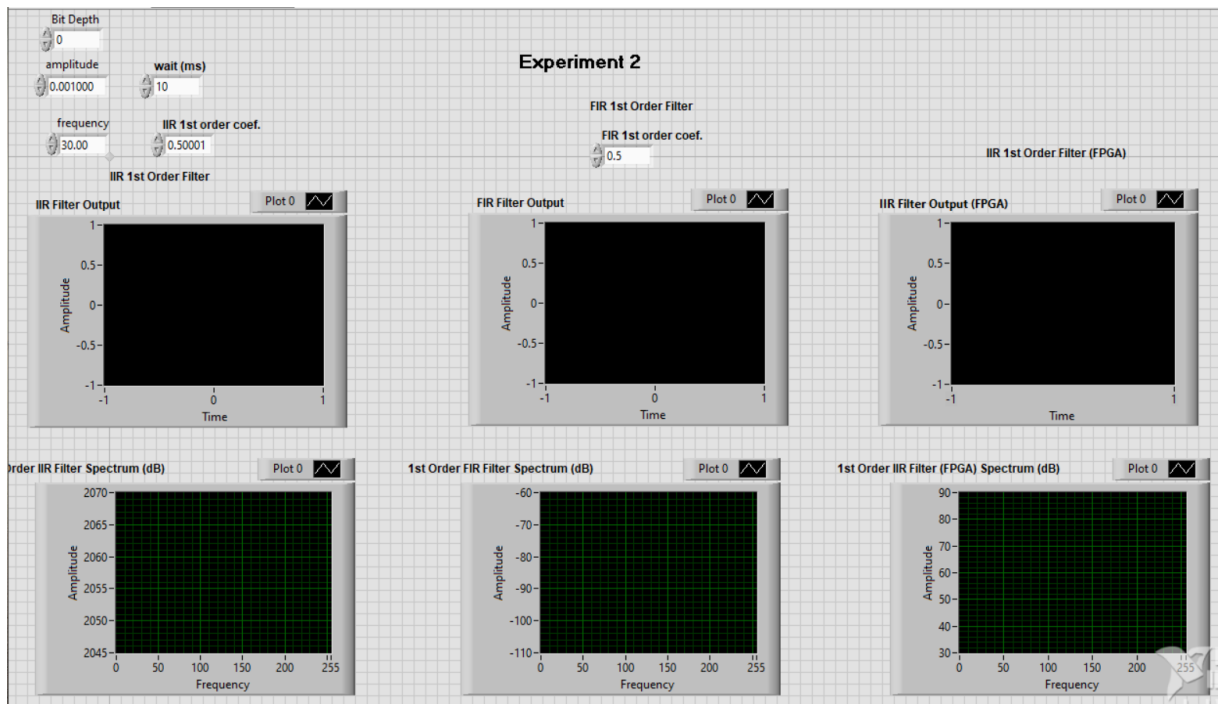
ii) You will also need to divide the final output by  **$2^{(Bit\ Depth)}$**  before visualization for scaling.

- 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.