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

In this experiment, simple real-time programming on an embedded platform, myRIO, is considered. 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 in myRIO CPU and FPGA.
In order to perform the experiment, the following programs should be installed in your computer;
1) LabVIEW 2016
2) LabVIEW Real-time Module 2016
3) LabVIEW myRIO Toolkit
4) LabVIEW FPGA Module

## II. Preliminary Work

1) Read the parts ***"1. Floating-Point Representation"*** and ***"2. Fixed-Point Representation"*** from the following web address:

   **http://www.ni.com/white-paper/3115/en/#toc1**

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***.

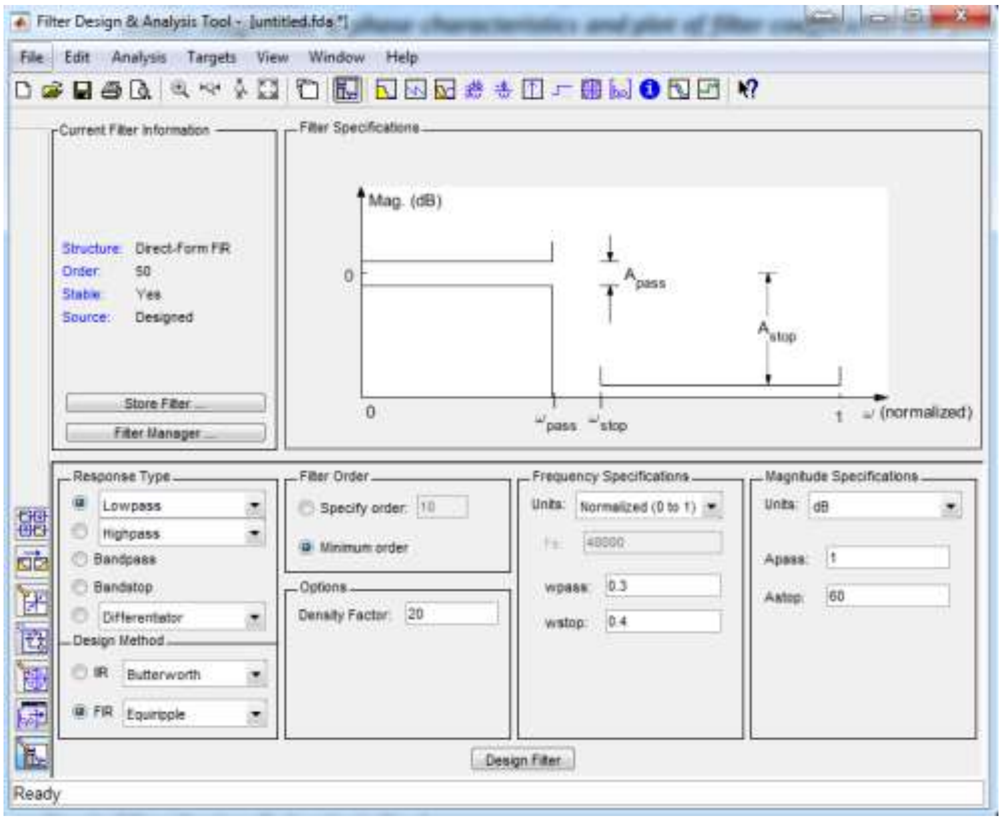
While multiplierless filters are advantageous, their frequency characteristics are inferior to the cases where the filter coefficients are represented by floating-point representation.

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

***<u>FIR and IIR Filter Implementation in MATLAB</u>***

**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 considering the magnitude characteristics 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***.



Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: Lowpass
Design Method: FIR Equiripple

Specify order: 10
Minimum order

Density Factor: 20

Units: Normalized (0 to 1)
Fs: 48000
wpass: 0.3
wstop: 0.4

Units: dB
Apass: 1
Astop: 60

Ready

***Fig. 1. Filter Design & Analysis Tool.***

**c)** Now design an ***IIR Chebyshev Type II*** filter with the same parameters. Write the order of filter and compare it with the previous FIR equiripple filter. In addition, **comment** on the placement of zeros and poles considering the magnitude characteristics 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 the frequency is **200 Hz** 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.

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

## III. Experimental Work

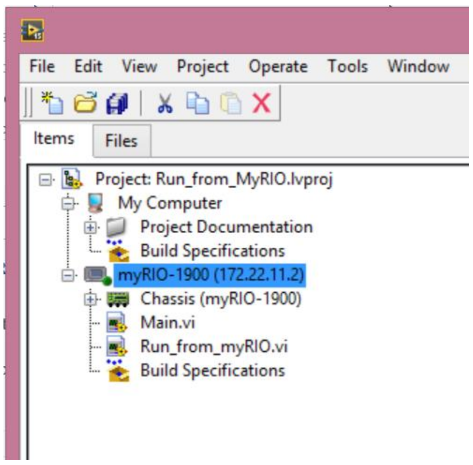
### 1. Simple Real-time Programming on MyRIO CPU

- 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.

- In the first part of this experiment, the steps for constructing a project for ***real-time processing on the myRIO CPU*** is outlined. Then, the required programming tasks are described. In the second part, ***real-time programming on myRIO FPGA*** is discussed and programming tasks are outlined.

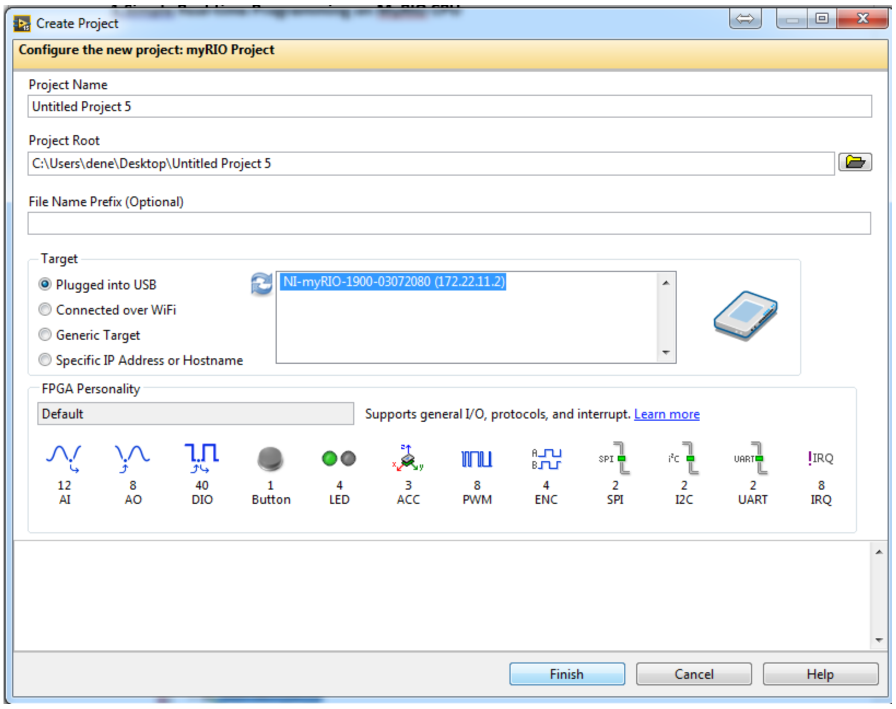
- The first step is to build a project on LabVIEW as shown in Fig. 3. First, create a ***myRIO project*** and select ***"Plugged into USB"*** as ***Target*** as shown in Fig. 4 and click ***Finish***.

- In order to bring Chassis under myRIO, ***right click on myRIO item*** in Project Explorer and select ***New → Targets and Devices*** as shown in Fig. 5. ***Expand myRIO Chassis*** item as in the Fig. 6 and select ***myRIO-1900*** and click ***OK***.



```
Project: Run_from_MyRIO.lvproj
    My Computer
        Project Documentation
        Build Specifications
    myRIO-1900 (172.22.11.2)
        Chassis (myRIO-1900)
        Main.vi
        Run_from_myRIO.vi
        Build Specifications
```

***Fig. 3. LabVIEW project for real-time programming in MyRIO.***


Create Project

Configure the new project: myRIO Project

Project Name: Untitled Project 5

Project Root: C:\Users\dene\Desktop\Untitled Project 5

File Name Prefix (Optional):

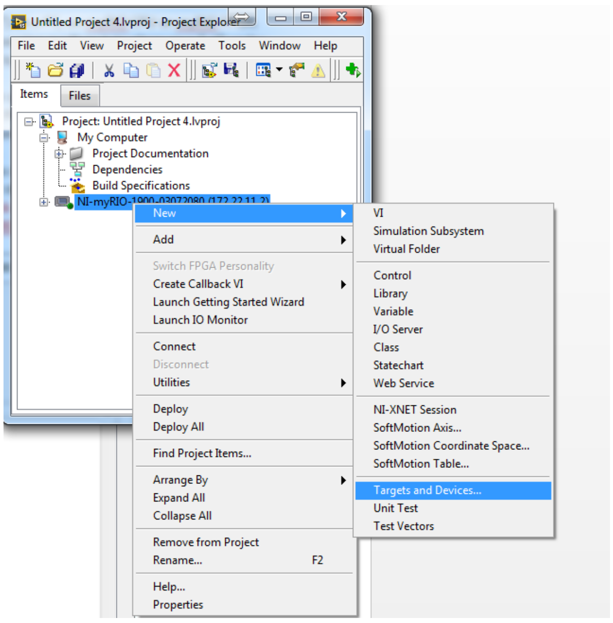
Target: [✓ Plugged into USB] Connected over WiFi / Generic Target / Specific IP Address or Hostname

NI-myRIO-1900-03072080 (172.22.11.2)

FPGA Personality: Default

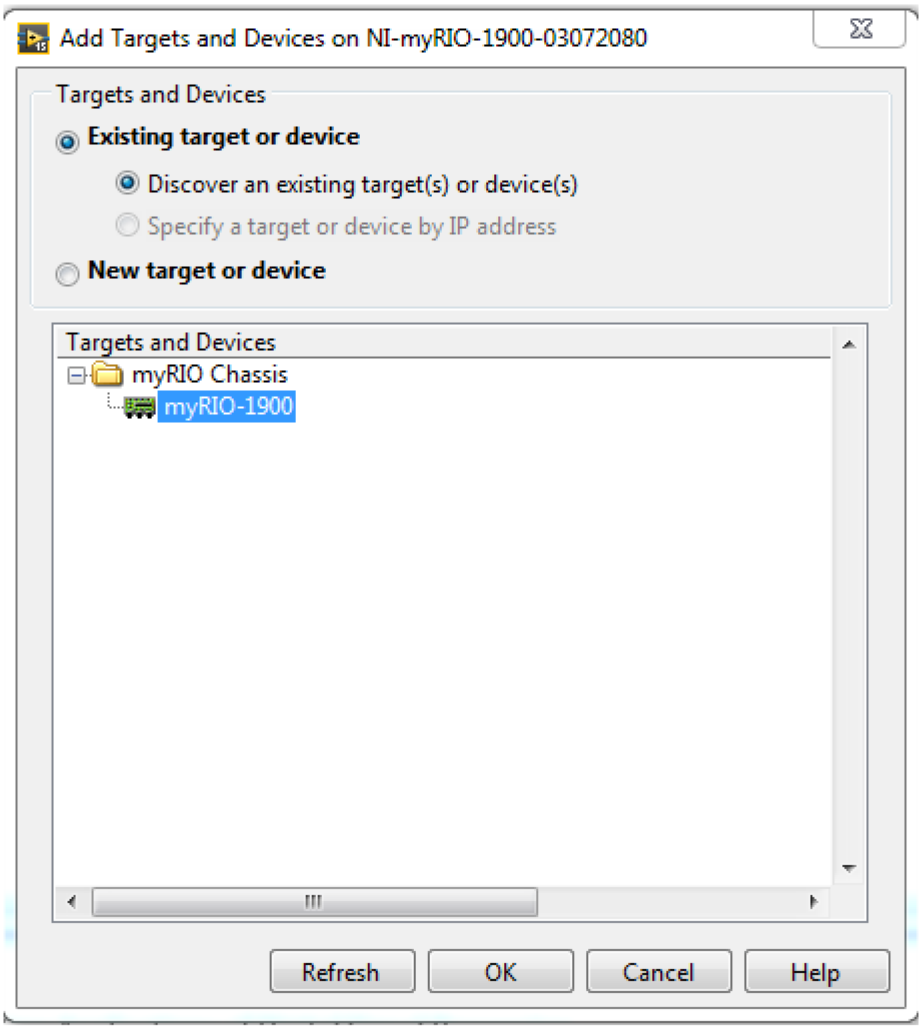
Supports general I/O, protocols, and interrupt. Learn more

*Fig. 4. Creating myRIO Project.*


Untitled Project 4.lvproj - Project Explorer

Project: Untitled Project 4.lvproj
My Computer
Project Documentation
Dependencies
Build Specifications
NI-myRIO-1900-03072080 (172.22.11.2)

*Fig. 5. Constructing Chassis under myRIO in the Project Explorer.*


Add Targets and Devices on NI-myRIO-1900-03072080

Targets and Devices

[✓ Existing target or device]

[✓ Discover an existing target(s) or device(s)]

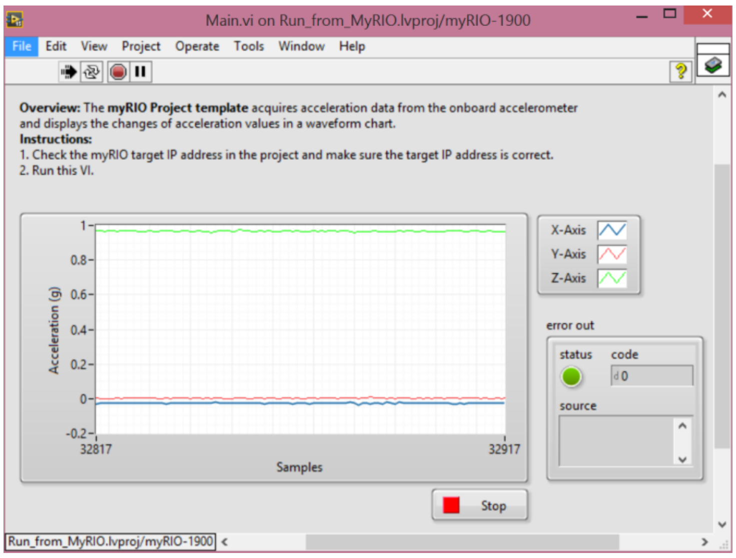
Specify a target or device by IP address

New target or device

Targets and Devices
- myRIO Chassis
  - myRIO-1900

*Fig. 6. Add Targets and Devices on myRIO.*

- Note that ***Main.vi*** is automatically inserted under myRIO-1900. This VI has a test program to check the sensors on myRIO and its Front Panel is given in Fig. 7.


Main.vi on Run_from_MyRIO.lvproj/myRIO-1900

**Overview:** The **myRIO Project template** acquires acceleration data from the onboard accelerometer and displays the changes of acceleration values in a waveform chart.
**Instructions:**
1. Check the myRIO target IP address in the project and make sure the target IP address is correct.
2. Run this VI.

error out: status ● code 0, source (empty)

*Fig. 7. Mani.vi Front Panel.*

- If you run this VI, you can observe the display as in Fig. 7.
- In this part, a simple program is constructed and ***run on myRIO***. In order to generate a new VI, add a new VI by ***right clicking on the myRIO item*** and select ***New → VI*** as shown in Fig. 8.

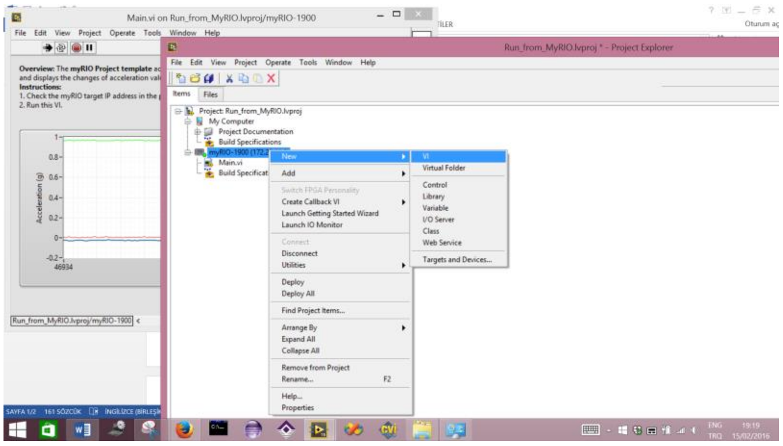


***Fig. 8. Adding a new VI under myRIO.***

- Once you open this new VI, both Front Panel and Block Diagram are observed. You can ***save*** this VI as ***"Run_from_myRIO.vi"***. Now, a simple addition operation can be programmed on the Block Diagram. So select the addition function from the functions menu as shown in Fig. 9.

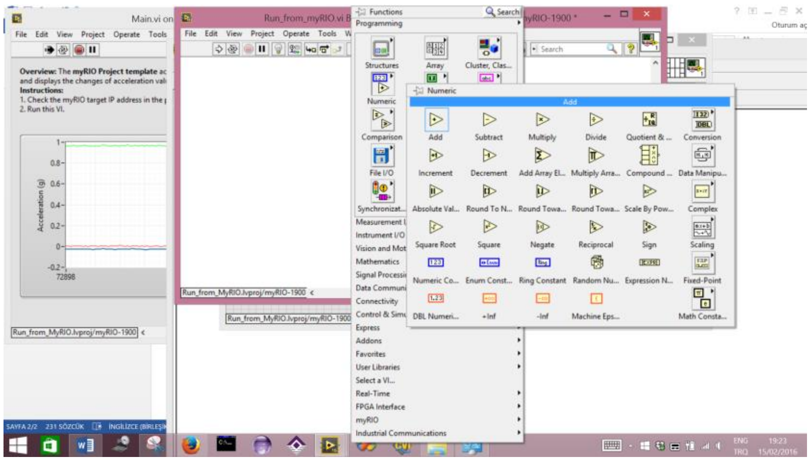


***Fig. 9. Addition function on Block Diagram.***

- Next, you need to add a control to one of the inputs of addition function as shown in Fig. 10.

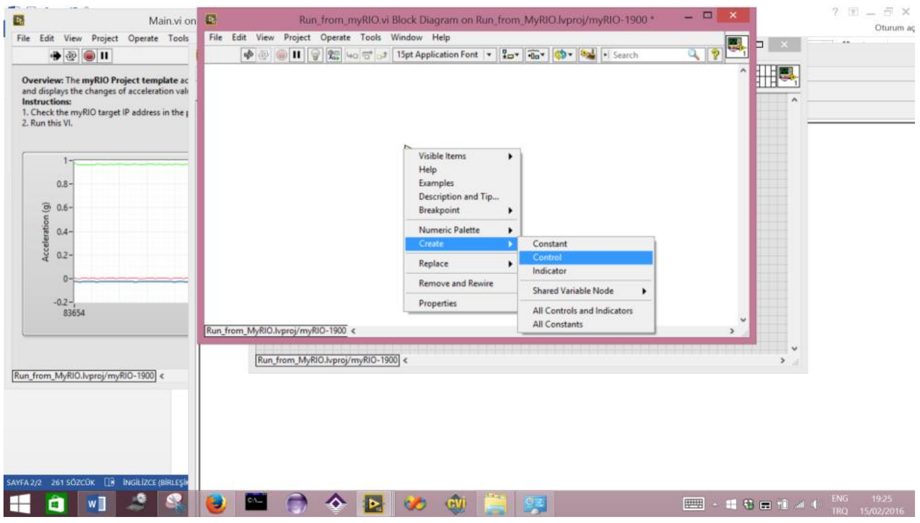


***Fig. 10. Adding a control to the addition function.***

- You also need to connect a ***constant*** to the other terminal of addition function. Set the constant as ***1***. The result of addition is connected to an indicator in order to show the resulting number on the Front Panel. If you do not include this program inside a while loop, it is executed only once. Hence it is better to use a while loop to observe the program run. The final program is shown in Fig. 11.

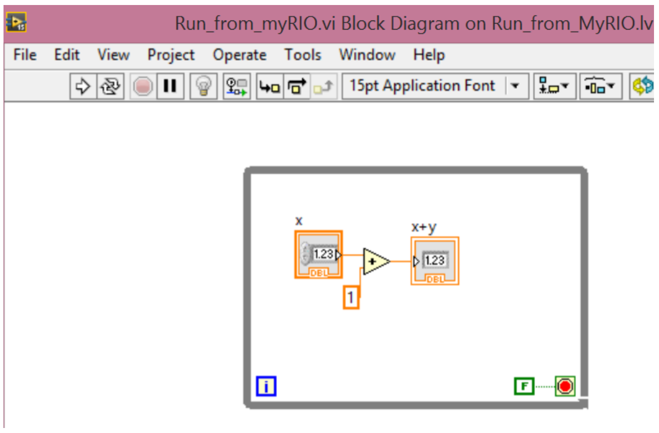

Run_from_myRIO.vi Block Diagram on Run_from_MyRIO.lv
File Edit View Project Operate Tools Window Help
15pt Application Font
x
x+y
1.23
1.23
1

***Fig 11. Final program for addition.***

- When we look at the Front Panel, two items are seen, namely ***the input as x and output as the x+y***. Run the program and enter any number to the input x to observe x+1. The program display is shown in Fig 12.

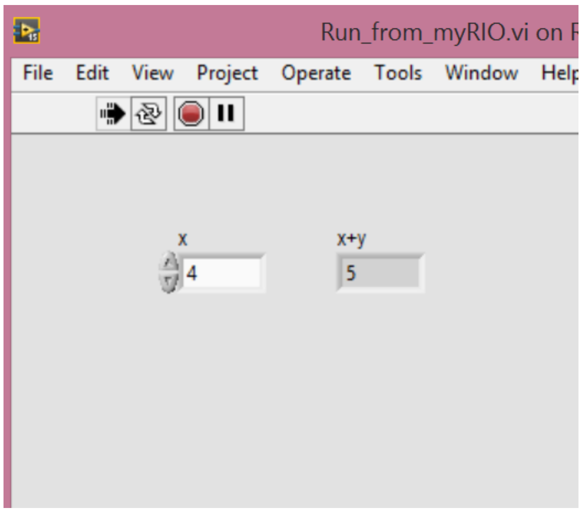


***Fig. 12. Real-time addition Front Panel.***

- Note that this program runs on ***myRIO CPU*** and hence it is a ***floating-point real-time implementation***.

**2. Programming Task 1**

**a)** You need to write your own VI in order to implement a ***subtraction function in real-time***. ***x is the input and x-y is the result where y is 1***. Generate your VI and run it to observe the output.

**b)** In this part, you need to implement ***first order IIR and FIR filters on myRIO CPU***. 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,

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

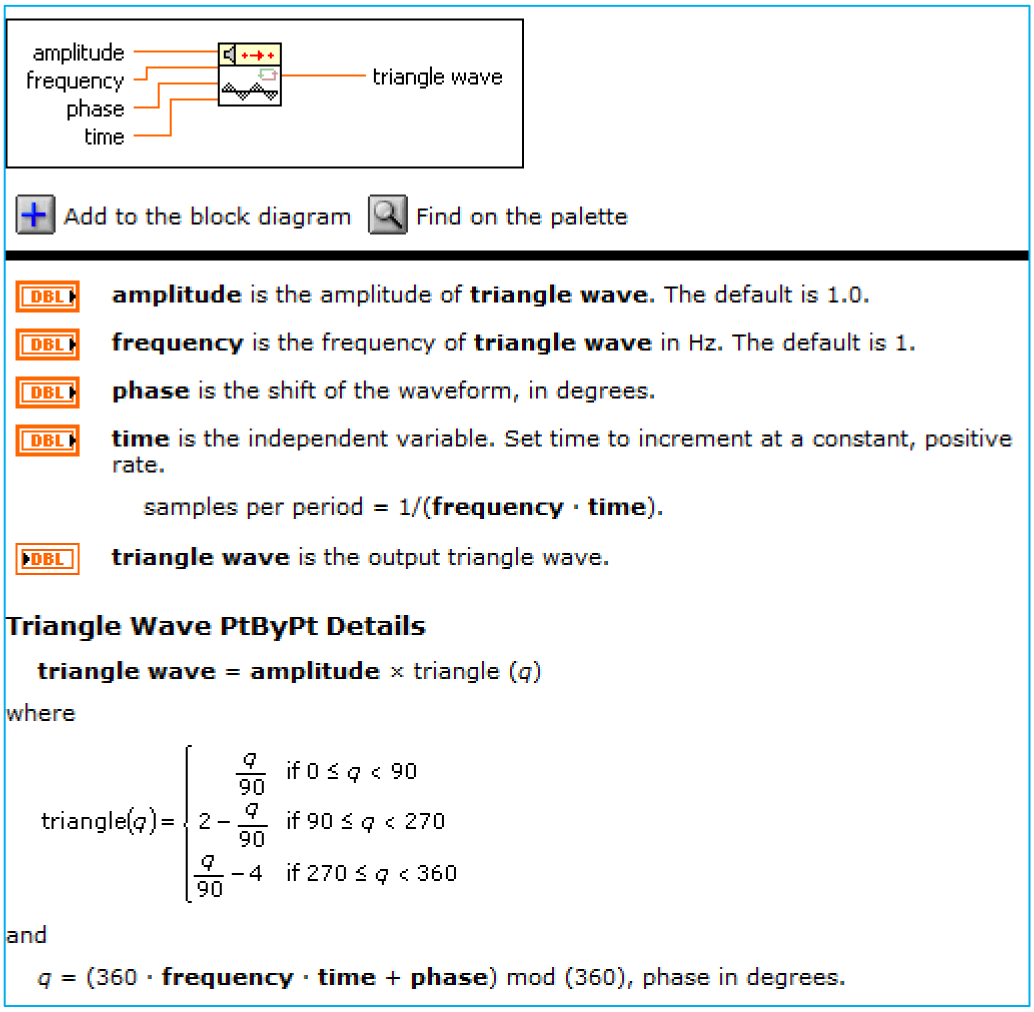
where b is the FIR filter coefficient which is given as the input from the front panel. The difference equation for the FIR filter is given as,

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

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 this 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. 13. 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. 14. So sampling rate is 10 kHz for the triangular wave.



**amplitude** is the amplitude of **triangle wave.** The default is 1.0.

**frequency** is the frequency of **triangle wave** in Hz. The default is 1.

**phase** is the shift of the waveform, in degrees.

**time** is the independent variable. Set time to increment at a constant, positive rate.

samples per period = 1/(**frequency** · **time**).

**triangle wave** is the output triangle wave.

**Triangle Wave PtByPt Details**

**triangle wave** = **amplitude** × triangle ($q$)

where

$$\text{triangle}(q) = \begin{cases} \frac{q}{90} & \text{if } 0 \le q < 90 \\ 2 - \frac{q}{90} & \text{if } 90 \le q < 270 \\ \frac{q}{90} - 4 & \text{if } 270 \le q < 360 \end{cases}$$

and

$q$ = (360 · **frequency** · **time** + **phase**) mod (360), phase in degrees.

***Fig. 13. Triangle Wave PtByPt Details.***

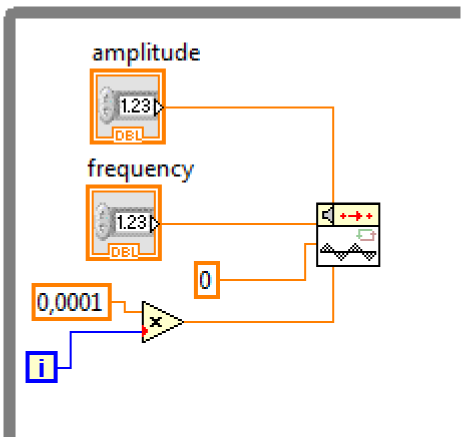


*Fig 14. Triangle waveform generation in a while loop.*

**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. 15.

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

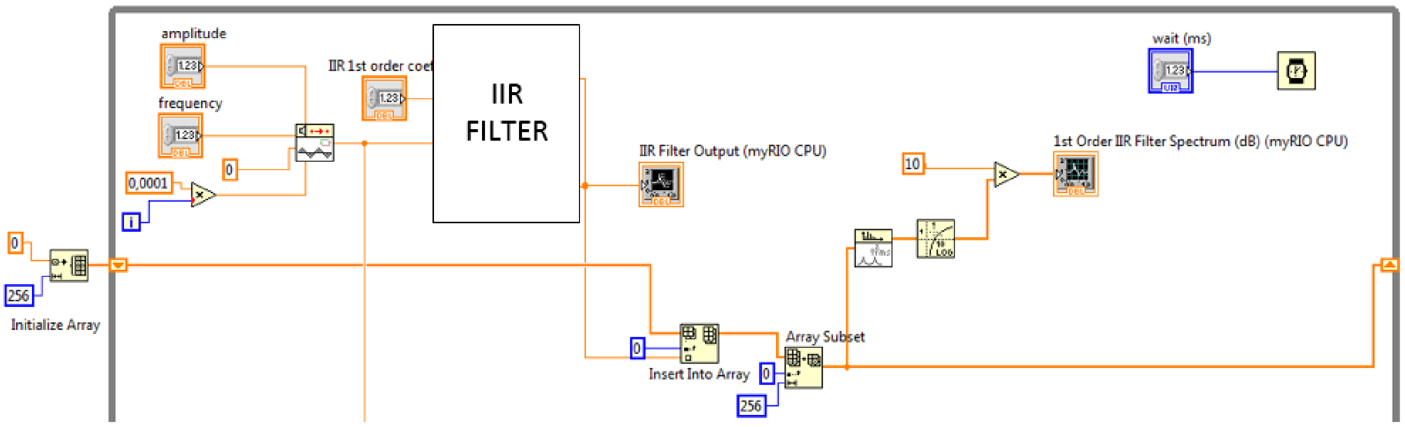


*Fig 15. Processing of the last 256 points.*

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

## 3. Simple Real-time Programming on myRIO FPGA

- As it pointed before, the programs (namely .vi files) run underneath either on computer, or on myRIO. In case of myRIO, there are two possibilities, namely myRIO CPU and myRIO FPGA. In order to run a program on myRIO FPGA you should place it under the FPGA target. Fig. 16 shows this process where a new VI is defined ***under the FPGA target***.

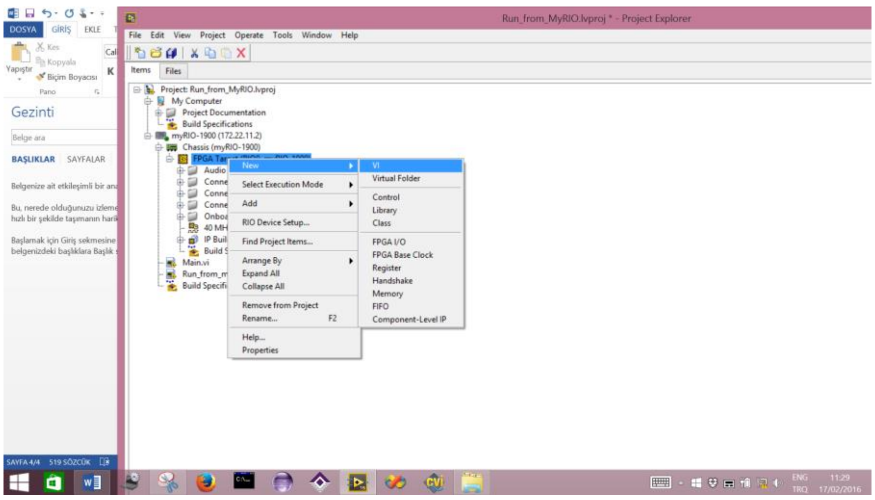


*Fig. 16. Placing a program under FPGA Target.*

- Once you place the VI under FPGA Target, you can access both the Front Panel and Block Diagram. In this experiment, a very simple programming task will be implemented. The task is to ***add and multiply two 16-bit number*** and display the results. In addition, the results are ***quantized to 8-bit*** representation and displayed in order to show the effect of quantization. In Fig. 17, the Block Diagram for this programming is shown.

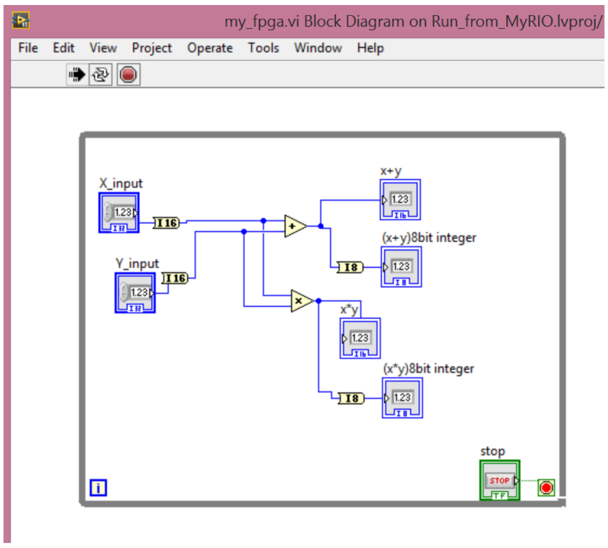
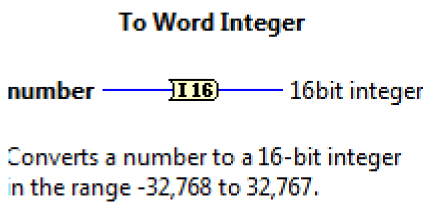

my_fpga.vi Block Diagram on Run_from_MyRIO.lvproj/
File Edit View Project Operate Tools Window Help
X_input
Y_input
I16
I16
x+y
(x+y)8bit integer
I8
x*y
(x*y)8bit integer
I8
stop
STOP

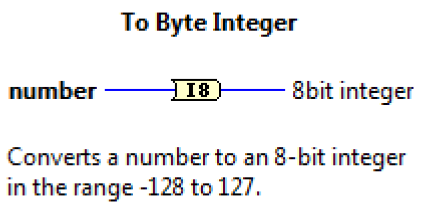
*Fig. 17. Block Diagram for the FPGA program.*

- As it is shown in this figure, two numbers ***X_input*** and ***Y_input*** are taken as the input and ***converted to 16-bit representation*** using ***"To Word Integer"*** function whose Context Help Window is shown in Fig. 18. Then they are added and multiplied respectively. The results are displayed on the Front Panel. Furthermore, the results are ***quantized with 8-bit*** representation using ***"To Byte Integer"*** function whose Context Help Window is shown in Fig. 19. Then the results are displayed on the Front Panel.


**To Word Integer**

Converts a number to a 16-bit integer
in the range -32,768 to 32,767.

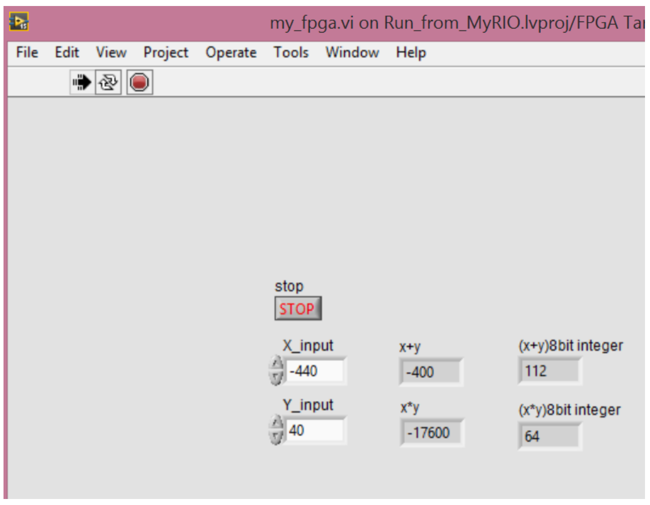
***Fig. 18. To Word Integer function.***


**To Byte Integer**

Converts a number to an 8-bit integer
in the range -128 to 127.

***Fig. 19. To Byte Integer function.***

- When you run this program, the code is first compiled and then run on the FPGA. After compilation, ***FPGA bit-file*** is generated and stored in your computer. When the program is run, you can enter any two numbers on the Front Panel and see the quantization effects. In Fig. 20, Front Panel of the program is shown.



X_input: -440
Y_input: 40
x+y: -400
x*y: -17600
(x+y)8bit integer: 112
(x*y)8bit integer: 64

***Fig. 20. Front Panel for the FPGA program.***

## 4. Programming Task 2

- In this part, **IIR filter** is implemented on ***myRIO FPGA***. We will enter the IIR filter coefficient from the Front Panel of the VI under myRIO CPU and generate triangle wave points in this VI of myRIO CPU. The filter coefficients will be transferred to myRIO FPGA through the procedure called ***"Front Panel Communications"***. When you need to transfer single data points as quickly as possible, you can use the "***Read/Write Control"*** function in the host VI (it is the VI under myRIO CPU in our case) to access the front panel controls and indicators of the FPGA VI.

- The second procedure for data transfer is ***Direct Memory Access (DMA) using FIFO's.*** When you do not need to transfer each data point immediately, you can wait for a large amount of data to accumulate and use a ***DMA FIFO*** to transfer the data efficiently. In this experiment, we will create two FIFO's one of which is used for transferring input triangle sequence from myRIO CPU to the FPGA. The second FIFO will be used to transfer the filter output sequence from the FPGA to the CPU.

**i)** Generate control variables on the FPGA VI where you will write the ***IIR filter coefficient*** and the ***tick counts*** from the control items in the Front Panel of VI under myRIO CPU. For this purpose, you can use ***Read/Write Control*** on the Block Diagram. Read/Write Control is an FPGA interface function where you can transfer values between myRIO CPU and FPGA. You should create 2 control variables on the FPGA, namely, ***IIR filter coefficient*** and the ***tick counts*** for synchronization purpose. The front panel for the FPGA VI should look like as in Fig. 21.

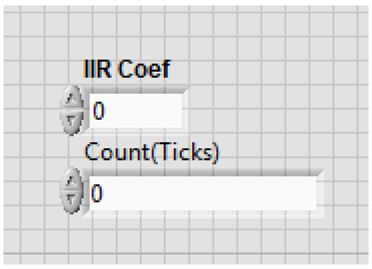
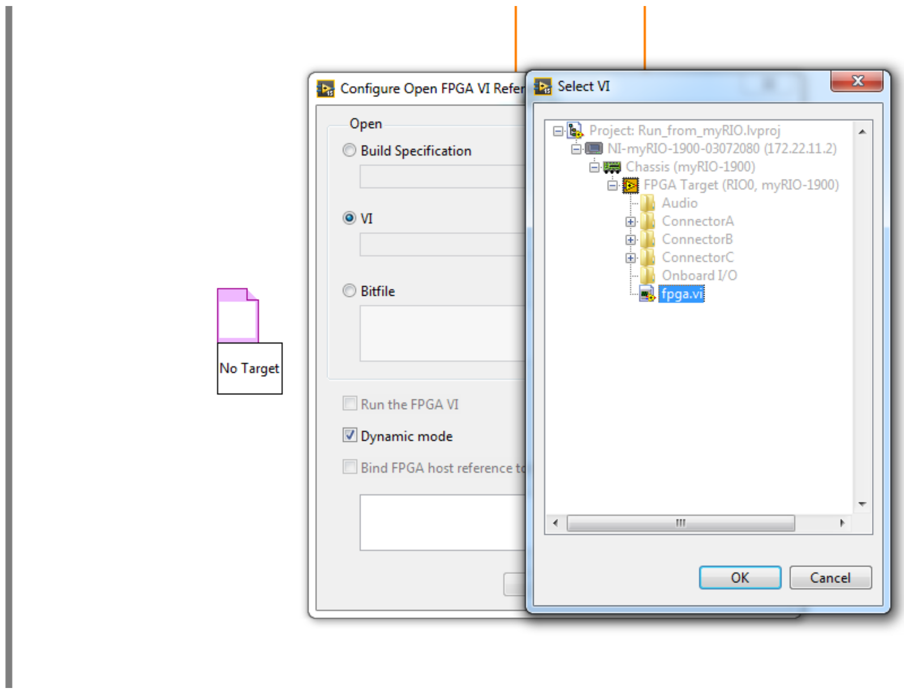


***Fig. 21. Front Panel for the FPGA program for implementing IIR filter.***

- The same control variables should also be created in the VI under myRIO CPU. Now, select ***Functions → FPGA Interface → Open FPGA VI Reference*** inside the while loop in the Block Diagram of the myRIO CPU VI. ***Double click*** on the node and select ***VI → your FPGA VI*** as shown in Fig. 22 and click OK. Place this node outside of the while loop as shown in Fig. 23. Then, select ***Functions → FPGA Interface → Read/Write Control*** and make wire connections as shown in Fig. 23.


Configure Open FPGA VI Refer

Open

Build Specification

[✓ VI]

Bitfile

Run the FPGA VI

[✓ Dynamic mode]

Bind FPGA host reference t

Select VI

Project: Run_from_myRIO.lvproj
NI-myRIO-1900-03072080 (172.22.11.2)
Chassis (myRIO-1900)
FPGA Target (RIO0, myRIO-1900)
Audio
ConnectorA
ConnectorB
ConnectorC
Onboard I/O
[✓ fpga.vi]

OK Cancel

*Fig. 22. Creating FPGA Interface.*

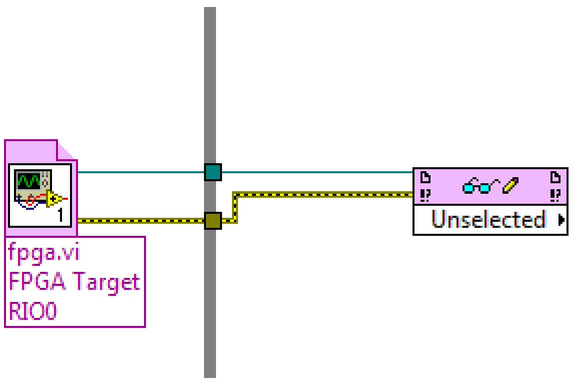


*Fig. 23. Connections between Open FPGA VI Reference and Read/Write Control.*

- Now, ***right click on Unselected*** item on Read/Write Control and select ***Controls → IIR Coef***. If you expand the node, you can see both of the controls as shown in Fig. 24.

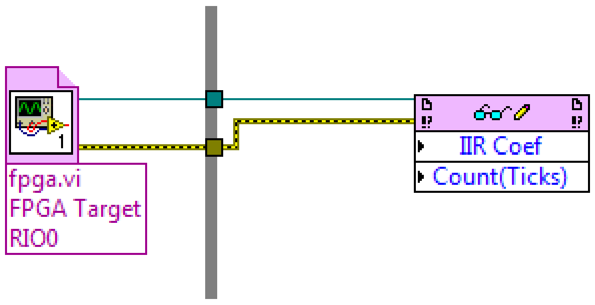


*Fig. 24. Creating Controls on Read/Write Control.*

- Define ***"Bit Depth"*** as a control in myRIO CPU 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.

- We need to transfer data in integer format since FPGA uses fixed-point arithmetic. For this purpose, we will multiply the controls to be transferred to the FPGA by ***2^(Bit Depth)*** and convert it to the ***32 bit integer*** as shown in Fig. 25. Here, we use ***Scale By Power Of 2*** and ***To Long Integer*** functions. Connect 32 bit integer controls to the input terminals of Read/Write Control as shown in Fig. 25. Note that, Bit Depth determines the number of bits to represent the numbers between -1 and 1 in integer format in the FPGA.

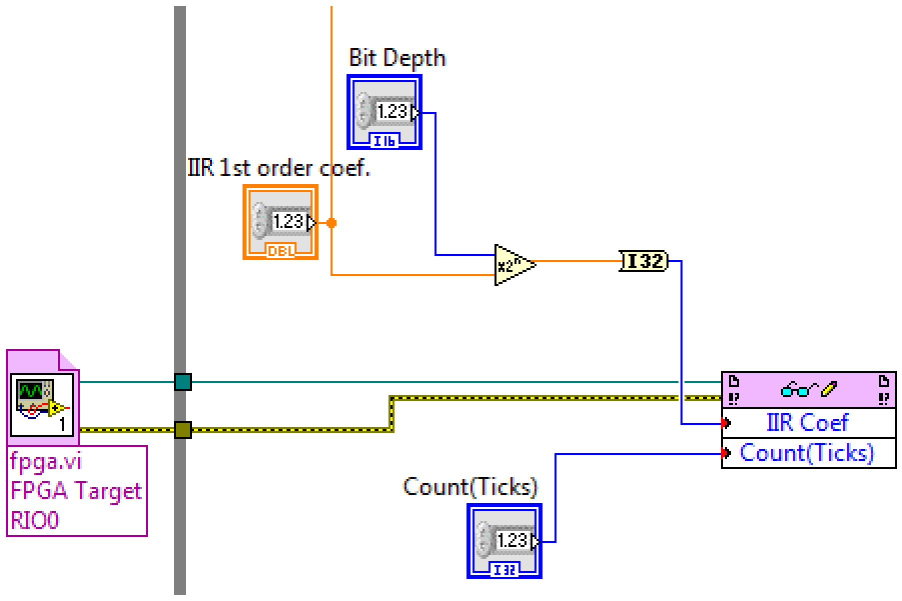
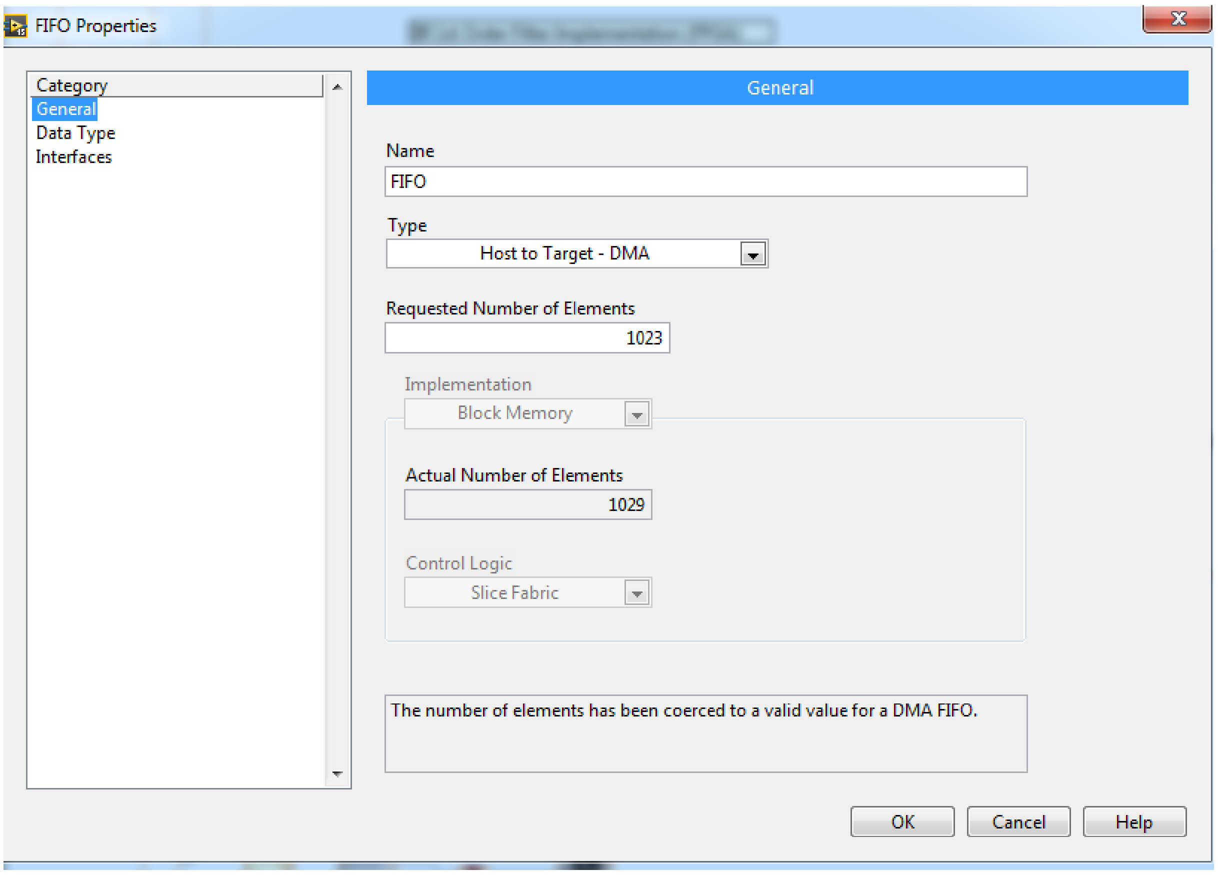


***Fig. 25. Bit Depth and Conversion to Long Integer of IIR Coef.***

**ii)** ***Triangular waveform samples*** are transferred to ***FPGA*** through ***DMA***. For this purpose, define a FIFO with I32 definition. For this, ***right click on FPGA Target*** and select ***New → FIFO*** in the ***Project Explorer***. The DMA will be from host (CPU) to target (FPGA). So, choose ***Host to Target – DMA*** as the type of FIFO as shown in Fig. 26. Set the ***data type of FIFO*** as *I32* as shown in Fig. 27. This FIFO is then read in FPGA to read the triangular wave samples. The filter outputs are also transferred to myRIO CPU through DMA. For this purpose, you need to create another FIFO for the IIR filter output. The filter outputs in FPGA are written at each iteration of the while loop in FPGA and then read in myRIO CPU program to display them in the front panel. The type of ***second FIFO*** is ***Target to Host-DMA and data type is I32***. At this point, two FIFO's, FIFO and FIFO2 should be seen in the Project Explorer as shown in Fig. 28.


FIFO Properties

Category: General (selected), Data Type, Interfaces

General

Name: FIFO

Type: Host to Target - DMA

Requested Number of Elements: 1023

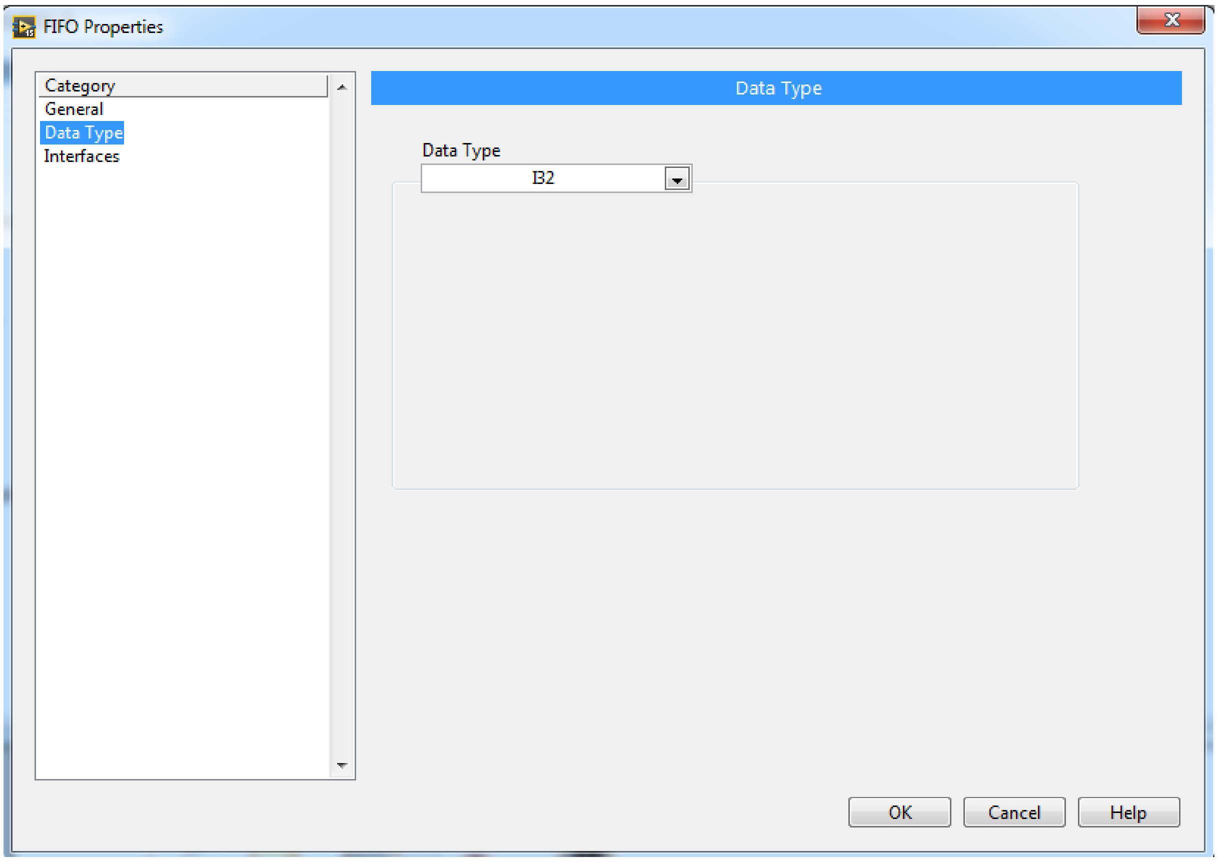
Implementation: Block Memory

Actual Number of Elements: 1029

Control Logic: Slice Fabric

The number of elements has been coerced to a valid value for a DMA FIFO.

***Fig. 26. FIFO Properties.***

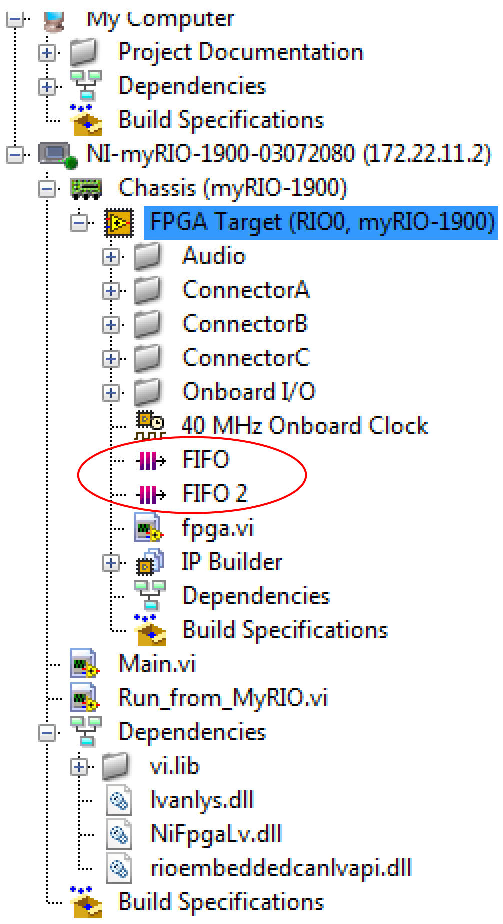

FIFO Properties

Category: General, Data Type (selected), Interfaces

Data Type

Data Type: I32

***Fig. 27. FIFO Data Type.***



```
My Computer
    Project Documentation
    Dependencies
    Build Specifications
NI-myRIO-1900-03072080 (172.22.11.2)
    Chassis (myRIO-1900)
        FPGA Target (RIO0, myRIO-1900)
            Audio
            ConnectorA
            ConnectorB
            ConnectorC
            Onboard I/O
            40 MHz Onboard Clock
            FIFO
            FIFO 2
            fpga.vi
            IP Builder
            Dependencies
            Build Specifications
    Main.vi
    Run_from_MyRIO.vi
    Dependencies
        vi.lib
        lvanlys.dll
        NiFpgaLv.dll
        rioembeddedcanlvapi.dll
    Build Specifications
```

***Fig. 28. Two FIFO's in the Project Explorer.***

- Now, we will write the triangle wave samples to FIFO. First, select ***Functions → FPGA Interface → Invoke Method*** and make connections to the ***Open FPGA VI Reference*** as shown in Fig. 29.

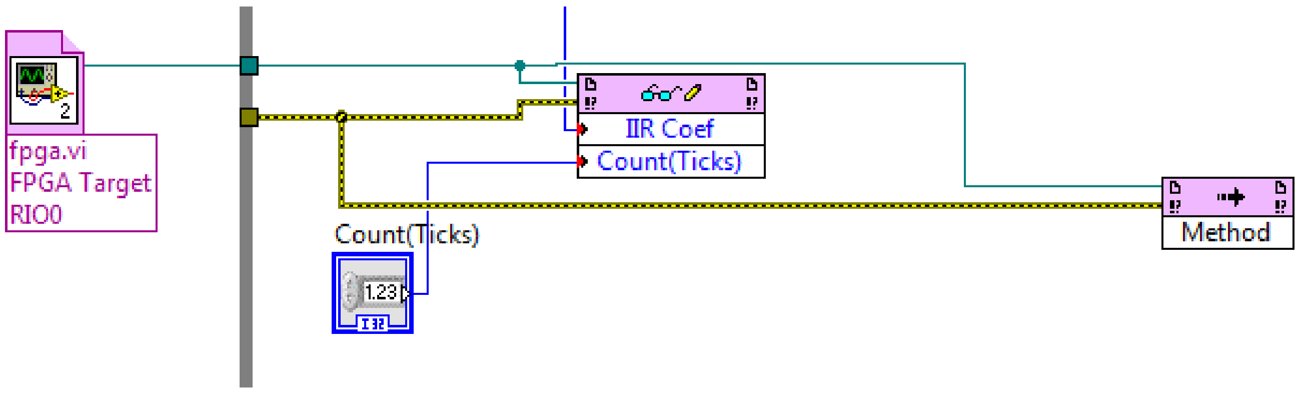


***Fig. 29. Connecting Invoke Method to the Open FPGA VI Reference.***

- ***Right click on Method*** item on Invoke Method node and select ***Method → FIFO → Write***. Now, the Invoke Method node should be seen as in Fig. 30.

- ***Timeout (ms)*** specifies the minimum number of milliseconds the Invoke Method function waits before timing out. The Invoke Method function times out if the host part of the FIFO does not contain enough space in which to write ***Data*** by the time the number of milliseconds you specify elapse. The default is 5000 milliseconds. Set this parameter to ***0*** as shown in Fig. 30.

- ***Empty Elements Remaining*** returns the number of empty elements remaining in the host memory part of the DMA FIFO. We will not use this terminal here.

- We will multiply the output of triangle wave generator by ***2^(Bit Depth)*** and convert it to the ***32 bit integer*** as shown in Fig. 30. Here, we use ***Scale By Power Of 2*** and ***To Long Integer*** functions as before. Additionally, we will use ***Build Array*** function as shown in Fig. 30 to convert the scalar into one element array since ***Data*** in Invoke Method accepts ***array*** data.

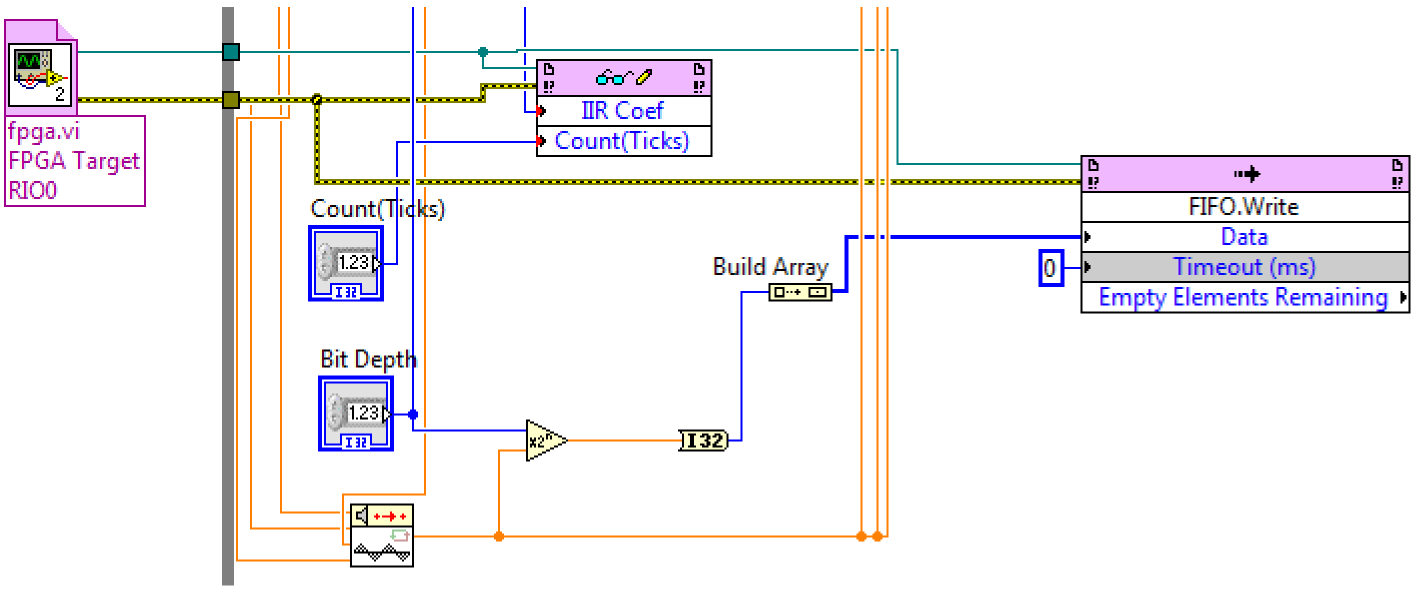


***Fig. 30. Writing triangle wave samples to FIFO.***

- After writing triangle wave samples, we can read the output of the IIR filter from FIFO2. The output will be written to FIFO2 by FPGA VI as we will show later. Place ***another Invoke Method*** function to the right side of ***FIFO.Write***. Make the connections as shown in Fig. 31 and ***right click on Method*** item. Select ***Method → FIFO 2 → Read***. The Invoke Method node for FIFO2 will be seen as in Fig. 31.

- ***Number of Elements*** determines the number of elements you read from the DMA FIFO. Connect a ***constant 1 and constant 0*** to ***Number of Elements and Timeout(ms)*** input terminals, respectively as shown in Fig. 31.

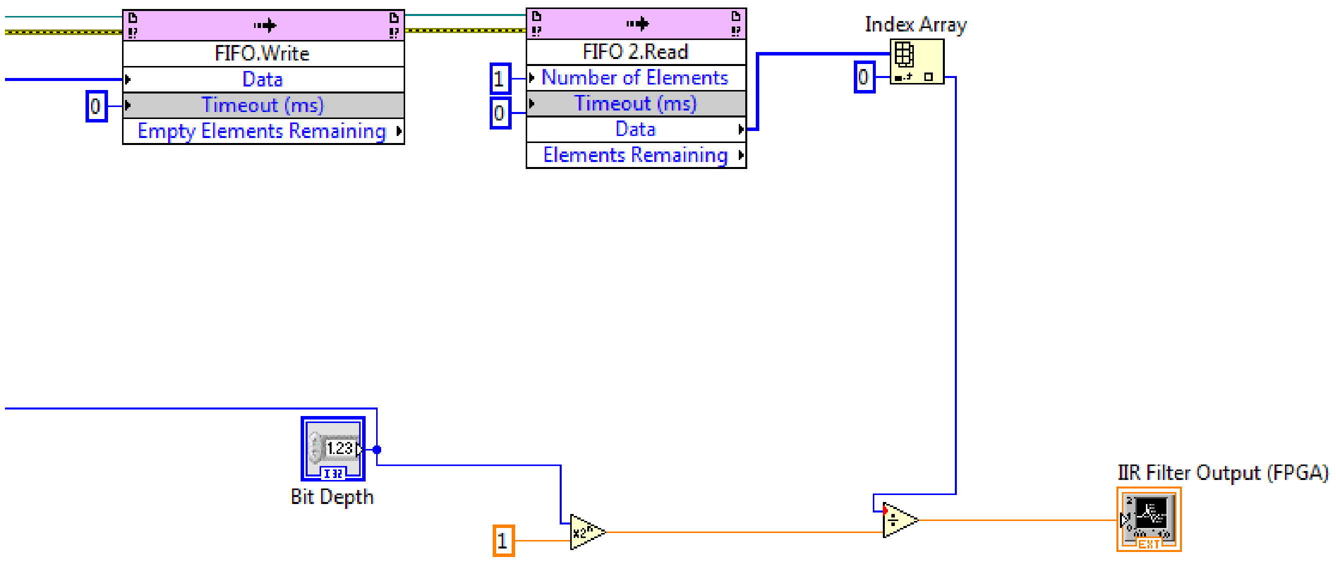


***Fig. 31. Reading output of IIR filter from FIFO2.***

- In Fig. 31, we use Index Array function to convert one element data array to the scalar. Then, we divide this value by ***2^(Bit Depth)*** in order to normalize the result. (Remember that we have multiplied the IIR filter coefficient and input sequence by ***2^(Bit Depth)*** before sending it to the FPGA).

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

- Now, let's construct the code in the FPGA VI. As you remember, there were two controls, ***IIR coef*** and ***Count(Ticks)***, in the FPGA VI. The reason of using Count(Ticks) is to control the speed of FPGA and provide synchronization with CPU. Construct the block diagram in Fig. 32 to read the input samples from FIFO, implement IIR filtering and writing the output of the filter to FIFO2.

- Change the timing setting of ***Wait*** function as ***"msec"*** by double clicking on Wait function in FPGA VI.

- Note that, we also use ***Bit Depth*** in the FPGA VI as a control. So, connect Bit Depth control in CPU VI to the ***Read/Write Control*** function as in Fig. 33.

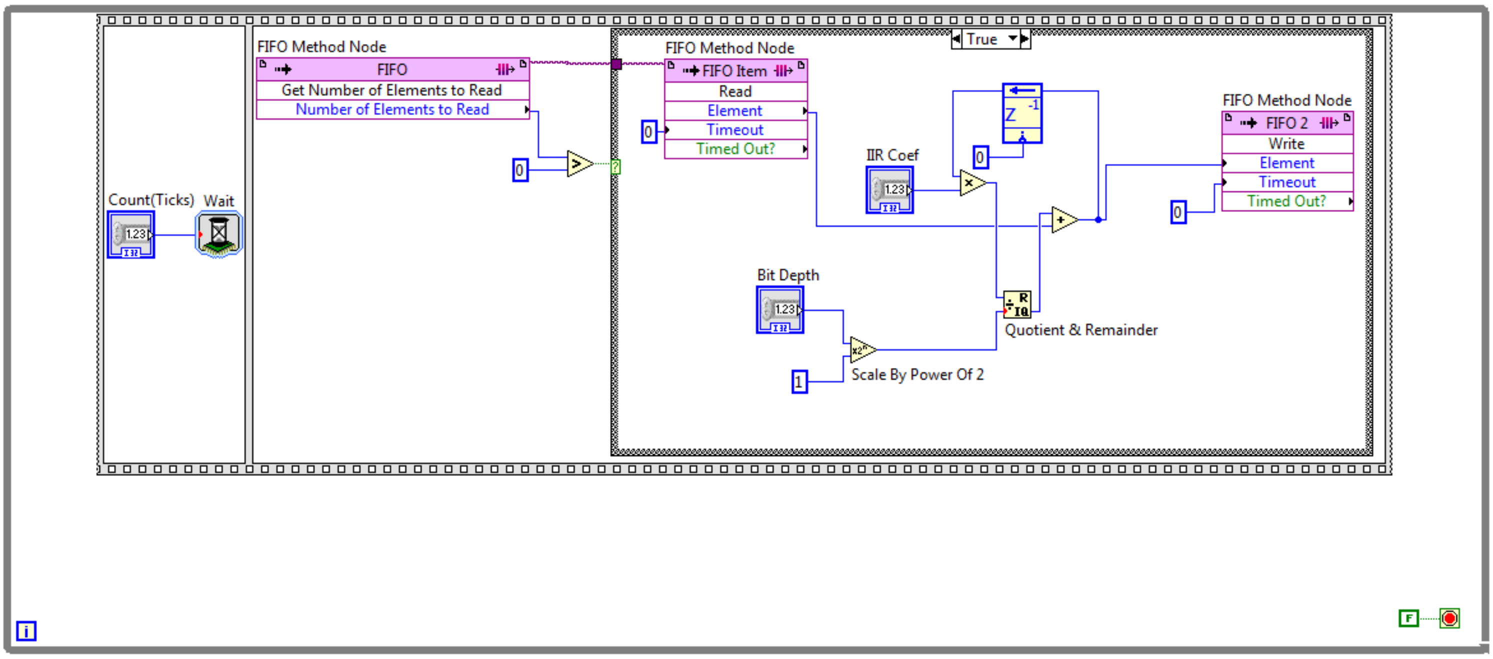


*Fig. 32. Block Diagram for FPGA VI.*

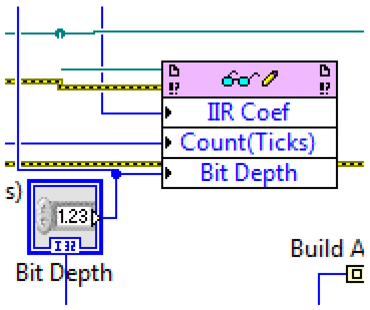


*Fig. 33. Connecting Bit Depth as control to Read/Write Control function in CPU VI.*

- You can add ***FIFO Method Node*** from ***Functions→Synchronization→FIFO→FIFO Method Node***.

- When you ***double click on the FIFO Method Node***, you can choose the FIFO you will use from ***Select FIFO***. The ***FIFO Method Node in the leftmost side*** of the Block Diagram in Fig. 32 is used to read the ***number of elements*** in the ***FIFO***. For this, you select ***FIFO*** and ***Select Method → Status → Get Number of Element to Read*** by double clicking on the node. ***The other FIFO Method Nodes*** are used to ***read elements from FIFO*** and ***write elements to FIFO2***. You can adjust their methods similarly.

- In the FPGA VI code, we ***divide the multiplication of IIR coef 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.,

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

where $2^{BitDepth} a$ and $2^{BitDepth} x[n]$ are rounded and sent to the FPGA. 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).

- Add a Wait function with a control inside the while loop in CPU VI as shown in Fig. 34.

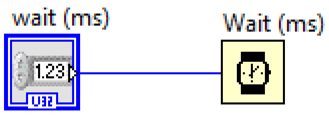
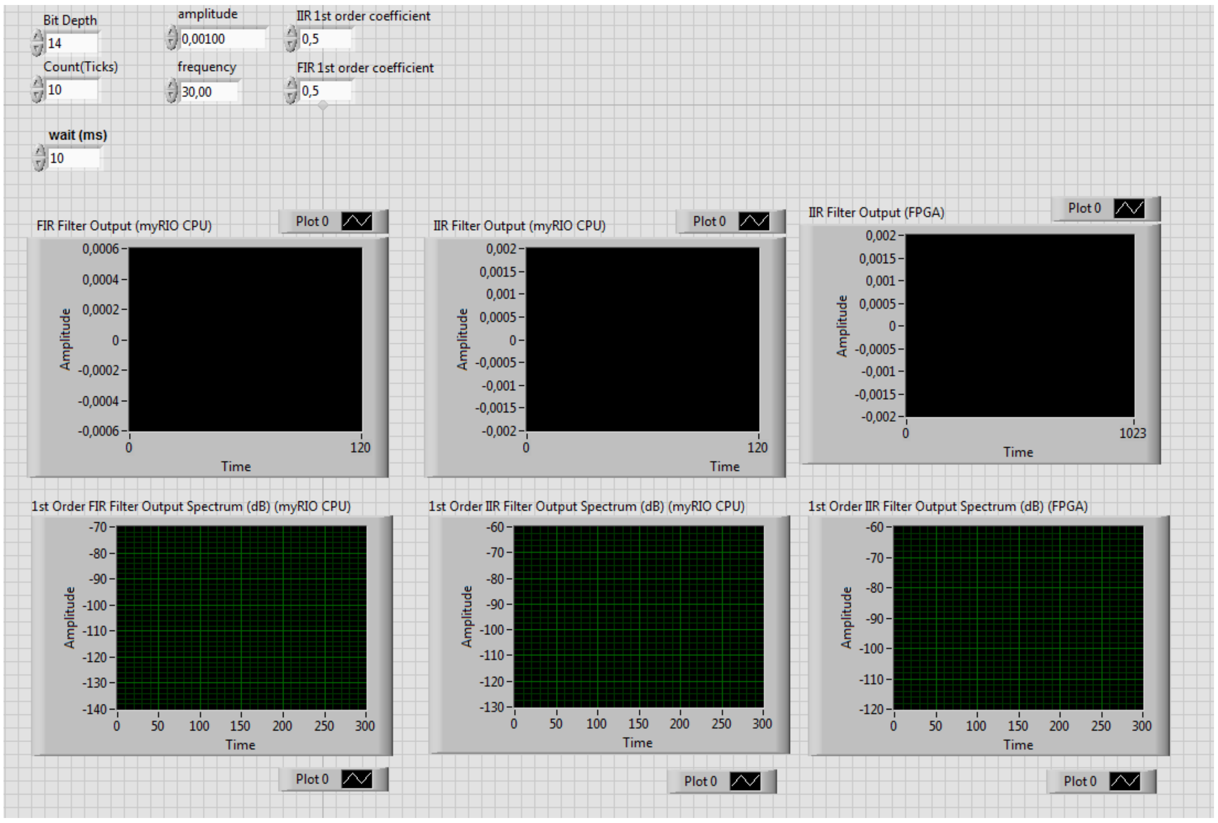


***Fig. 34. Wait function with a Control in CPU VI.***

- Run the FPGA VI to compile it.
- After compilation, you can run your CPU VI.
- Fig. 35 shows an example front panel for the required programming task.

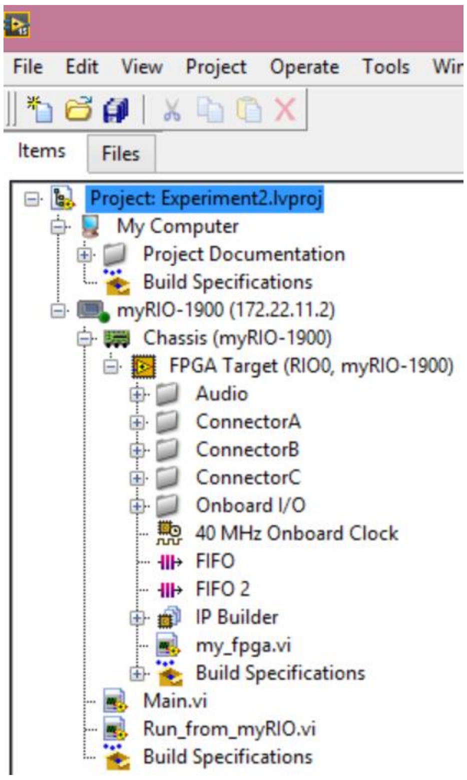


Bit Depth 14
Count(Ticks) 10
wait (ms) 10
amplitude 0,00100
frequency 30,00
IIR 1st order coefficient 0,5
FIR 1st order coefficient 0,5

FIR Filter Output (myRIO CPU)
IIR Filter Output (myRIO CPU)
IIR Filter Output (FPGA)
Plot 0
Amplitude
Time
0,0006
0,0004
0,0002
0
-0,0002
-0,0004
-0,0006
120
0,002
0,0015
0,001
0,0005
-0,0005
-0,001
-0,0015
-0,002
1023
1st Order FIR Filter Output Spectrum (dB) (myRIO CPU)
1st Order IIR Filter Output Spectrum (dB) (myRIO CPU)
1st Order IIR Filter Output Spectrum (dB) (FPGA)
-60
-70
-80
-90
-100
-110
-120
-130
-140
0
50
100
150
200
250
300

*Fig. 35. Example front panel for the required programming task in Experiment 2.*

- Figure 36 shows the complete project view for Experiment 2.



File Edit View Project Operate Tools Wir

Items Files

```
Project: Experiment2.lvproj
    My Computer
        Project Documentation
        Build Specifications
    myRIO-1900 (172.22.11.2)
        Chassis (myRIO-1900)
            FPGA Target (RIO0, myRIO-1900)
                Audio
                ConnectorA
                ConnectorB
                ConnectorC
                Onboard I/O
                40 MHz Onboard Clock
                FIFO
                FIFO 2
                IP Builder
                my_fpga.vi
                Build Specifications
        Main.vi
        Run_from_myRIO.vi
        Build Specifications
```

*Fig.36. Project Explorer Window for the Experiment 2.*

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

iv) 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.

v) 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?

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

vii) Now, ***increase Bit Depth*** from ***14*** to ***17*** 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.

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

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

## 5. Experiment 2 Bonus Part (20pts):

Implement a second order FIR filter in Fourier domain using FFT in a block-by-block manner. This implementation will be on ***myRIO CPU***. Each input block is obtained and processed using the filter coefficients in Fourier domain. You can use 256-point FFT in your implementation. The filter output is displayed on the front panel in real-time.