

# IRIO DESIGN RULES

RULES TO DESIGN ADVANCED DATA ACQUISITION  
SYSTEMS USING LABVIEW FOR FPGA AND RIO DEVICES

INSTRUMENTATION AND APPLIED ACOUSTICS - RESEARCH GROUP



GRUPO DE INVESTIGACIÓN EN  
**INSTRUMENTACIÓN Y  
ACÚSTICA APLICADA**



### **Worldwide Technical Support and Product Information**

Web: [www.i2a2.upm.es](http://www.i2a2.upm.es)

Support: [irio@i2a2.upm.es](mailto:irio@i2a2.upm.es)

### **I2A2 Research Group – Technical University of Madrid**

UPM Campus Sur,

Carretera de Valencia, km 7, 28031 Madrid

Phone: +34 91 3364696

Fax: +34 91 3364696

## Table of contents

<b>1</b>	<b>SCOPE .....</b>	<b>7</b>
1.1	Identification.....	7
1.2	Document Overview.....	7
1.3	Acronyms .....	7
<b>2</b>	<b>REFERENCED DOCUMENTS .....</b>	<b>9</b>
2.1	References .....	9
<b>3</b>	<b>METHODOLOGY.....</b>	<b>10</b>
3.1	Introduction.....	10
<b>4</b>	<b>CREATING A LABVIEW PROJECT FOR A RIO DEVICE.....</b>	<b>12</b>
4.1	Creating a new LabVIEW project .....	12
<b>5</b>	<b>DESIGN RULES FOR FLEXRIO .....</b>	<b>15</b>
5.1	Platform identification.....	15
5.2	Mandatory resources for a FlexRIO design. ....	15
5.3	Analog Signal Data acquisition profile.....	19
5.3.1	Mandatory resources for data acquisition profile. ....	19
5.3.2	Data format in the DMA for Data acquisition profile.....	22
5.3.2.1	PXIe 7966R / 5761R.....	23
5.3.2.2	PXIe 7961R / 6581R.....	23
5.3.2.3	PXIe 7961R or PXIe7966R without adapter module.....	23
5.3.2.4	Format A: DAQ samples.....	23
5.3.2.5	Format B (TBD).....	25
5.3.3	Optional resources.....	25
5.3.3.1	Analog inputs .....	25
5.3.3.2	Auxiliary analog inputs .....	26
5.3.3.3	Analog Output.....	27
5.3.3.4	Auxiliary analog output .....	27
5.3.3.5	Digital input/output .....	27
5.3.3.6	Signal generator .....	32
5.3.4	LabVIEW for FPGA VI implementation for data acquisition profile .....	33
5.3.4.1	Rules to be applied when designing for LabVIEW/FPGA for FlexRIO data acquisition profile .....	36
5.3.5	Summary of resources used for Data acquisition profile.....	42
5.4	Image acquisition profile .....	45
5.4.1	Mandatory resources for Image acquisition profile .....	45
5.4.2	Data format in the DMA for Image Acquisition profile .....	50
5.4.2.1	PXIe 7966R / NI1483 .....	50
5.4.3	Summary of resources used for image acquisition profile.....	51

<b>5.5</b>	<b>Analog Signal Data acquisition profile (data to GPU).....</b>	<b>55</b>
5.5.1	Mandatory resources for data acquisition profile (data to GPU). ....	56
<b>5.6</b>	<b>Image acquisition profile (data to GPU) .....</b>	<b>60</b>
5.6.1	Mandatory resources for Image acquisition profile .....	60
<b>6</b>	<b>DESIGN RULES FOR CRIO .....</b>	<b>62</b>
<b>6.1</b>	<b>Platform identification.....</b>	<b>62</b>
<b>6.2</b>	<b>Mandatory resources for a cRIO design.....</b>	<b>62</b>
<b>6.3</b>	<b>Analog Signal Data acquisition profile (DMA-based) .....</b>	<b>66</b>
6.3.1	Mandatory resources for data acquisition profile .....	66
6.3.2	Data format in the DMA for Data acquisition profile.....	68
6.3.2.1	NI9159/NI9205 .....	68
6.3.2.2	Format A: DAQ samples.....	68
6.3.2.3	Format B: (TBD).....	70
6.3.3	Optional resources.....	70
6.3.3.1	Analog inputs .....	71
6.3.3.2	Auxiliary analog inputs .....	71
6.3.3.3	Analog Output.....	72
6.3.3.4	Auxiliary analog output .....	72
6.3.3.5	Digital input/output .....	72
6.3.3.6	Signal generator .....	72
6.3.4	Summary of resources for cRIO DAQ profile .....	73
<b>6.4</b>	<b>Point by Point acquisition profile .....</b>	<b>77</b>
6.4.1	Mandatory resources for point by point I/O profile .....	77
6.4.2	Optional resources.....	77
6.4.2.1	Analog inputs .....	77
6.4.2.2	Auxiliary analog inputs .....	77
6.4.2.3	Analog Output.....	78
6.4.2.4	Auxiliary analog output .....	78
6.4.2.5	Digital input/output .....	78
6.4.2.6	Signal Generator .....	78
6.4.3	Summary of resources for cRIO Point by Point profile.....	78
<b>6.5</b>	<b>cRIO Examples.....</b>	<b>81</b>
6.5.1	cRIO Basic requirements for the examples provided .....	81
6.5.2	Module Identification in the Chassis .....	81
6.5.3	Module Description and Signal Interconnections.....	82
6.5.3.1	NI9205 Analog Input Module.....	82
6.5.3.2	NI9264 Analog Output Connection .....	83
6.5.3.3	NI9401 Digital Input/Output.....	85
6.5.3.4	NI9477 Digital sinking output module .....	87
6.5.3.5	NI9426 Sourcing Digital Input Module .....	88

6.5.3.6	NI9425 Sinking Digital Input Module .....	89
6.5.3.7	NI9476 Sourcing Digital Output Module .....	90
6.5.4	System General Description.....	91
6.5.4.1	General Block Diagram .....	91
6.5.4.2	State Machine.....	91
6.5.4.3	Operation State: I/O Acquisition Loop .....	92
6.5.4.4	System Management: Host HMI.....	92
6.5.5	Point by Point DAQ Profile Example .....	93
6.5.5.1	Objective .....	93
6.5.5.2	cRIO Hardware Elements Used .....	93
6.5.5.3	Signal Connection .....	94
6.5.5.4	Mandatory Resources for Point by Point I/O Profile .....	94
6.5.5.5	Optional Resources .....	94
6.5.5.6	LabVIEW Implementation for a cRIO Point by Point DAQ .....	95
6.5.5.7	Host HMI Program.....	97
6.5.6	Analog Signal DAQ Profile (DMA based) Example.....	99
6.5.6.1	Objective .....	99
6.5.6.2	cRIO Hardware Elements Used .....	99
6.5.6.3	Signal Connection .....	100
6.5.6.4	Mandatory Resources for Analog Signal DAQ Profile .....	100
6.5.6.5	Optional Resources .....	100
6.5.6.6	LabVIEW Implementation for a cRIO Analog Signal DAQ Profile .....	101
6.5.6.7	Host HMI Program.....	103
<b>7</b>	<b>LABVIEW FOR FPGA TEMPLATES .....</b>	<b>107</b>
7.1.1	Location of the templates in GitHub repository.....	107
7.1.2	LabVIEW template directory structure .....	107
<b>7.2</b>	<b>FlexRIO templates .....</b>	<b>107</b>
<b>7.3</b>	<b>cRIO templates .....</b>	<b>108</b>
<b>8</b>	<b>USING THE LABVIEW TEMPLATES.....</b>	<b>110</b>
<b>8.1</b>	<b>Overview .....</b>	<b>110</b>
<b>8.2</b>	<b>Templates.....</b>	<b>110</b>
8.2.1	LabVIEW template browser description.....	110
8.2.2	Folder Libs .....	111
8.2.3	Target Clocks .....	111
8.2.4	DMAs to Host .....	111
8.2.5	NI Adapter module.....	112
8.2.6	Build specifications.....	112
8.2.7	LabVIEW template VI.....	112
8.2.7.1	Control panel.....	112
8.2.7.2	Block diagram .....	114

**9 NI FPGA INTERFACE C API GENERATOR .....119**  
**9.1 Executing the application .....119**  
**9.2 Header file generated.....120**

# 1 SCOPE

## 1.1 Identification

This document contains the description of the different design rules that must be followed by LabVIEW for FPGA designers when developing applications for the RIO devices. All the designs for FlexRIO and cRIO, meeting the design rules explained in this document, can be used by the software layers IRIO Library, NI-RIO EPICS Device Driver and IRIO-NDS. These design rules allow:

- The development of FPGA-based data acquisition applications for analog and digital signals using FlexRIO and cRIO platforms
- The development of FPGA-based image data acquisition applications using cameralink cameras using FlexRIO platform.
- The integration of the monitoring part in a FPGA-based control application using cRIO

## 1.2 Document Overview

This document summarises the design rules to be followed by the LabVIEW for FPGA designers. The document contains a basic description of the methodology to be used, and the design rules for cRIO and FlexRIO platforms. Additionally, examples and templates are explained.

## 1.3 Acronyms

CPU	Central Processing Unit
DAQ	Data Acquisition
FlexRIO	Flexible Reconfigurable Input/Output
FPGA	Field Programmable Gate Array
FPSC	Fast Plant System Controller
I&C	Instrumentation and Control
I/O	Input and Output
NI	National Instrument
PCDH	Plant Control Design Handbook
PCI	Peripheral Component Interconnect – computer bus standard
PCI Express	Peripheral Component Interconnect Express

PICMG	PCI Industrial Computer Manufacturers Group
PXI	PCI Extensions for Instrumentation
PXI Express	An evolution of PXI using PCI Express technologies
OS	Operating system
RIO	Reconfigurable input/output



## 2 REFERENCED DOCUMENTS

### 2.1 References

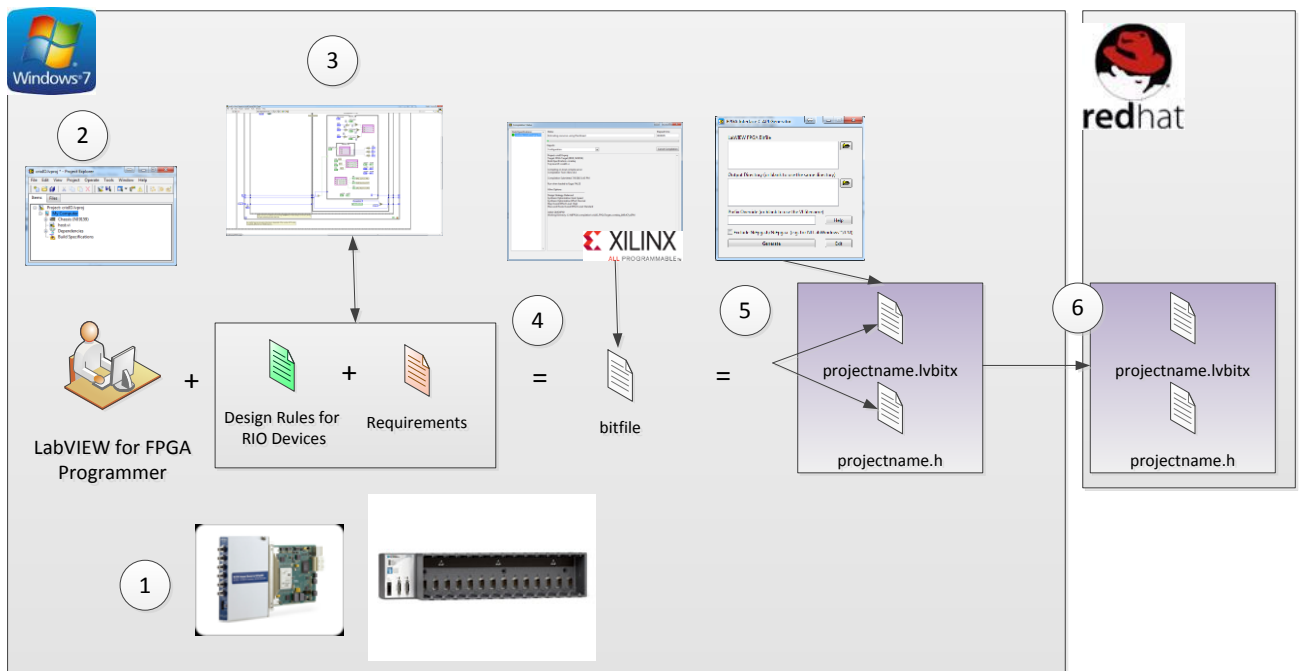
- [RD1] NI FlexRIO FPGA Module Specifications (<http://www.ni.com/pdf/manuals/372525d.pdf>).
- [RD2] NI FlexRIO Help. Edition Date: June 2010. Part Number: 372614D-01.  
<http://www.ni.com/pdf/manuals/372614d.zip>
- [RD3] NI LabVIEW for CompactRIO. <http://www.ni.com/pdf/products/us/fullcriodevguide.pdf>
- [RD4] Developer's Guide LabVIEW TM FPGA Course Manual. Course Software Version 2009. August 2009 Edition. Part Number 372510C-01.
- [RD5] The NI LabVIEW High-Performance FPGA Developer's Guide.  
<http://www.ni.com/tutorial/14600/en/>
- [RD6] [http://zone.ni.com/reference/en-XX/help/370984T-01/criodevicehelp/module\\_ids/](http://zone.ni.com/reference/en-XX/help/370984T-01/criodevicehelp/module_ids/)
- [RD7] DDS Waveform Generation Reference Design for LabVIEW FPGA  
<http://www.ni.com/example/31066/en/>

## 3 METHODOLOGY

### 3.1 Introduction

This section explains the different steps that a developer must follow to build data acquisition applications using National Instruments RIO devices: FlexRIO and cRIO. These RIO devices have an FPGA chip that controls the DAQ process and interchange data with a host using a communication bus (PCIe). To program the FPGA on RIO devices, you need to follow these steps (see Fig. 1):

1. Select the NI FlexRIO device [RD2] with the adapter module, or a cRIO system [RD3].
2. Create a LabVIEW project [RD1]. Section 4 provides more details about this and LabVIEW project templates are available to be downloaded from GitHub repository to simplify the development.
3. Write a VI using the specific rules described in this document and the recommendations described in [RD4]. Sections 5 and 6 describe the rules to be used when creating a VI from FlexRIO and cRIO respectively.
4. Compile the VI and obtain the bitfile using LabVIEW for FPGA tools. This tool calls directly to the XILINX compiler simplifying the process to obtain the bitfile. Once you complete successfully this step you can test and debug your LabVIEW for FPGA application in the windows computer.
5. Before starting the test in a Linux machine, generate the header file of the design using the “FPGA Interface C API Generator” software application. Section 8 provides more details about this step.
6. Move (copy) the header file and bit file obtained to the Linux machine and start testing your applications in the Redhat environment.



**Fig. 1: Design flow for RIO devices**

## 4 CREATING A LABVIEW PROJECT FOR A RIO DEVICE

The first step using a RIO device is to develop a project in LabVIEW for FPGA. This software only runs in Windows OS. This software will provide all the tools to configure/program the FPGAs. The FPGA is configured using a “bitfile” generated by the software environment. To start the LabVIEW development environment, execute the LabVIEW program using the Windows panel Fig. 2.

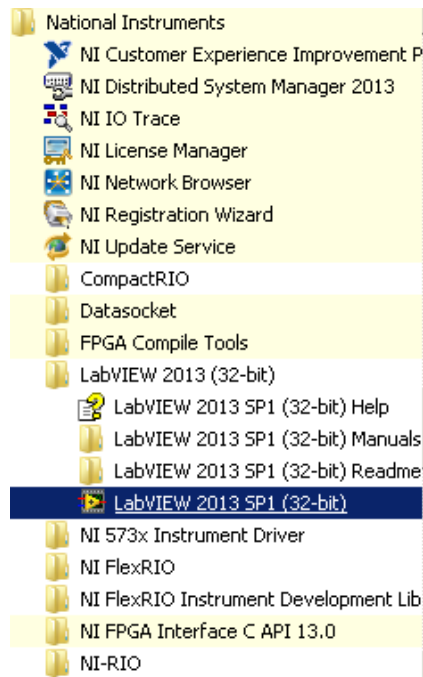
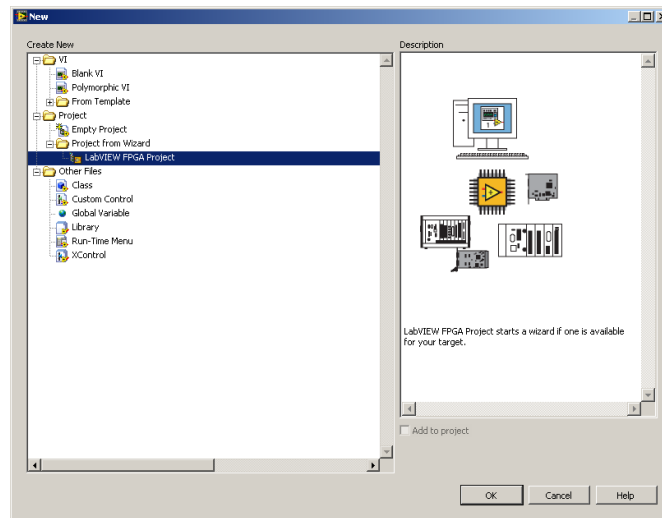


Fig. 2 LabVIEW (32-bit) programs access.

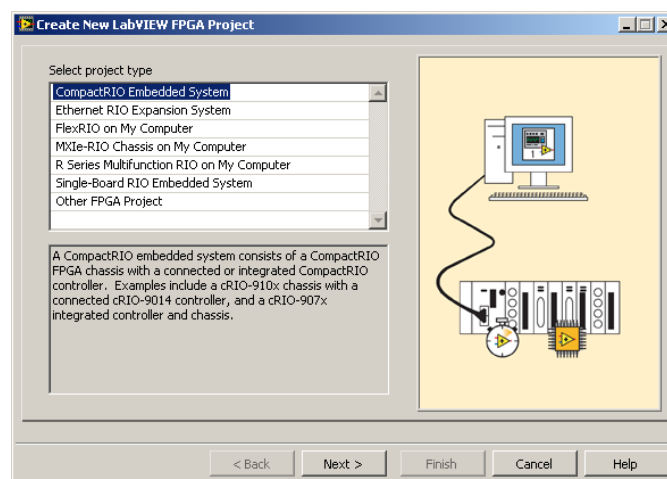
### 4.1 Creating a new LabVIEW project

Once LabVIEW is started, the first step is to create a new LabVIEW project containing the target (the FPGA card). Select “File->New” and a window will be displayed. Select Project->Project from Wizard->LabVIEW FPGA Project (see Fig. 3).



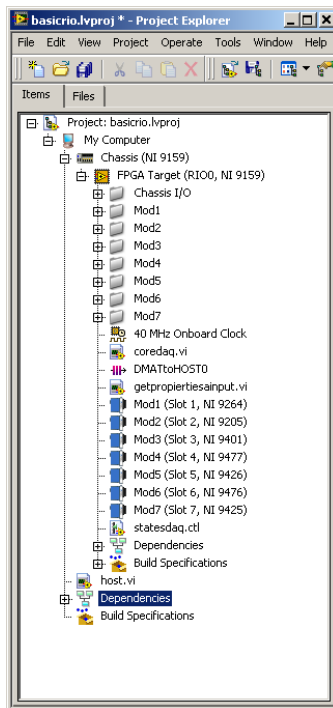
**Fig. 3: Selecting the wizard for LabVIEW for FPGA.**

Now, you need to select the target. You can choose *MXIe-RIO chassis on My Computer* for NI9159-based systems or *FlexRIO on My Computer* for FlexRIO bundles (see Fig. 4).



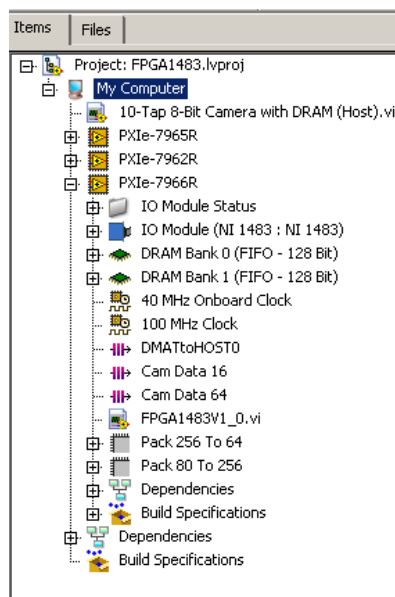
**Fig. 4: Selecting the RIO platform**

In the case of compactRIO system you can choose the chassis model (NI9159 in this case) and later the cRIO modules allocated in them. This last step should be completed depending on your final configuration (select the modules needed). In the case of FlexRIO select the FlexRIO device (7961R or 7966R). Once you complete this step you will find a LabVIEW project already prepared for your development (Fig. 5).



**Fig. 5 LabVIEW project with a FPGA target**

Fig. 5 shows a project example with a compactRIO system NI 9159. This cRIO [RD3] system is connected in the development computer using a PCIe with an MXIe interface. The NI9159 contains an FPGA (Virtex 5) and 7 input/output modules connected. Detailed information on how to use LabVIEW with FPGAs is available in [RD6]. Fig. 6 shows an example of a project containing a PXIe7966R with a NI1483 adapter module.



**Fig. 6: Elements used in a LabVIEW project using a RIO device**

In this point you are ready to start your development using these design rules. These design rules apply only to the VI to be developed for the FPGA. The implementations of VIs for host computer have to follow the standard procedures when developing code for LabVIEW for Windows.

## 5 DESIGN RULES FOR FLEXRIO

The FPGA VI must contain a set of terminals that are mandatory independently of the application. These terminals are presented in Fig. 7. Different colours are used in order to identify clearly the different functionality. Some terminals need a default value because they will be read when the FPGA is not running yet. Independently of this the FPGA code description, the developed system has to meet additional rules described later in this document.



**[Important]:** The FlexRIO [RD2] design rules have been defined considering that the main objectives are: a) the acquisition and processing of analog signals using the NI5761R adapter module; b) the acquisition and generation of digital patterns using the NI6581R; c) the implementation of a frame grabber using the NI1483.

### 5.1 Platform identification

**Platform:** Enum U8 indicator register. This element is the register used to identify the hardware platform in use. The values for this terminal are shown in Table 1. The value of this terminal is read by the software driver when the bitfile has been downloaded to the FPGA but is not running.

Table 1: Values for Platform indicator

Platform	Value
FlexRIO	0
cRIO	1
R-Series	2



**Warning. R Series is contemplated but is not currently supported.**

### 5.2 Mandatory resources for a FlexRIO design.

Fig. 7 shows the basic resources to be added in a LabVIEW for FPGA design for FlexRIO. The meaning and functionality of the different terminals are explained below.

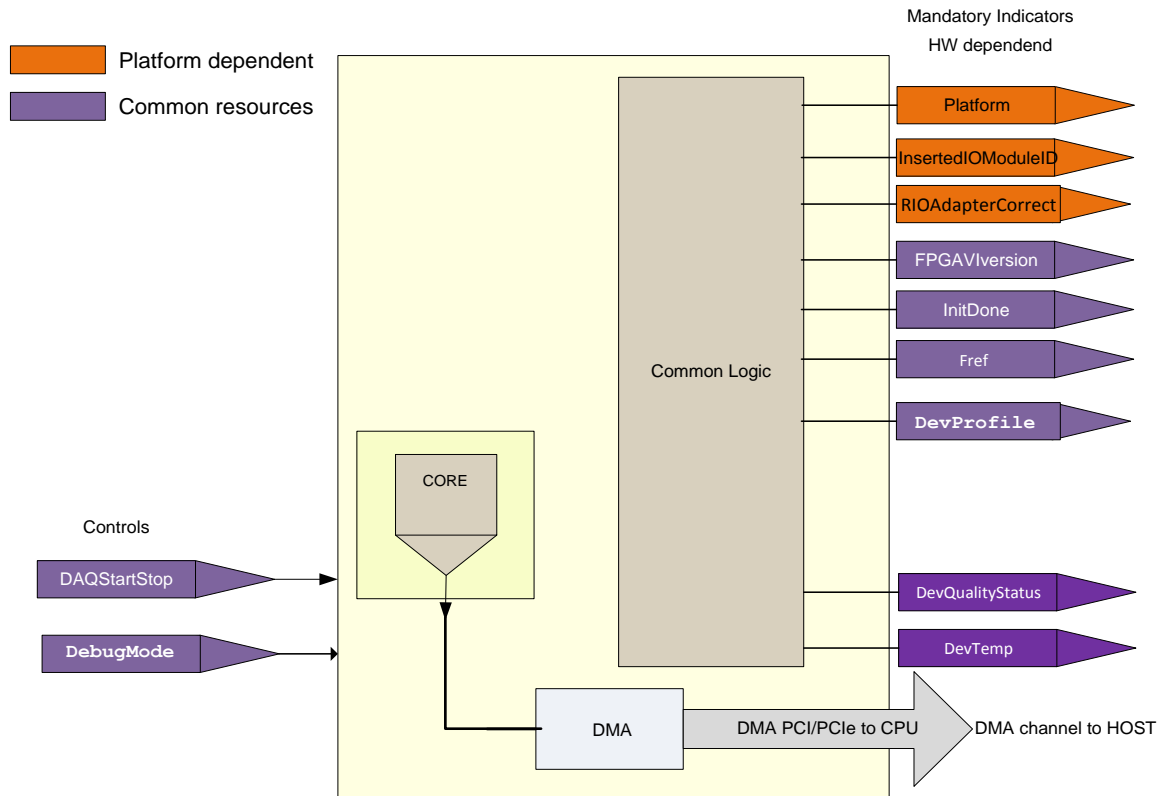


Fig. 7: Common terminals in the VI for FlexRIO

**FPGAVersion:** U8 array indicator. This indicator contains the version of the VI, which has to be checked by the software driver. The array is composed by 2 elements, the first one includes the major version “MM”, and the second one the minor version “mm”.

**InitDone:** Boolean register indicator. This indicator is used to signal that the FPGA and adapter module are correctly initialised. Zero means that the FGPA is not ready, and 1 means that the FPGA is ready. The designer should define when the FPGA and the I/O elements are ready to work checking the information provided by the module manufacturer. The FPGA designer has to follow the specific steps defined for the adapter module to perform the initialization.

Table 2: Values for Boolean InitDone indicator

Initdone	Value
Correct	True
Incorrect	False

**InsertedIOModuleID:** U32 indicator. This indicator contains the identification of the inserted module as defined in the LabVIEW/FPGA design. The values for this indicator are shown in Table 3.



**Table 3: Values for ExpectedIOModuleID indicator**

Module	ExpectedIOModuleID
1483	0x109374C9
5761	0x109374C6
6581	0x10937418

**RIOAdapterCorrect:** Boolean indicator. This indicator signals if the detected module matches the expected one.

**Fref:** U32 indicator. The indicator contains the reference clock in Hz used for the sampling rate acquisition.

**DevQualityStatus:** U8 indicator. This indicator signals about the possible errors in the signal conditioning systems if available or other possible situations related with the quality of data acquisition process.

**DevTemp:** I16 indicator. This indicator contains the temperature of the RIO's FPGA.

**DevProfile:** U8 indicator. This indicator is used to determine the kind of application implemented in the FPGA. If DevProfile is equal to 0 the implementation contains a design for analog input waveform oriented data acquisition. Then, the resources defined for this profile are mandatory. For this profile waveform output generation, digital and analog point by point I/O are optional. If DevProfile is equals to 1 the resources for Image profile are mandatory. Table 5 and Table 6 summarize the mandatory and optional resources. Profiles 2 and 3 require a fast controller with a NVIDIA/FlexRIO bundle installed (NVIDIA GPU with FlexRIO).

**Table 4: Values for DevProfile indicator**

DevProfile	Info	Data acquired are sent to
0	Data acquisition	CPU Memory
1	Image acquisition	CPU Memory
2	Data acquisition	NVIDIA GPU Memory
3	Image acquisition	NVIDIA GPU Memory

**Table 5: Resources for data acquisition profile (FlexRIO)**

Resources		Info
	<b>Common</b>	Mandatory
	<b>Data acquisition (HOST or GPU)</b>	Mandatory
<b>Analogs</b>	<b>Analog Input</b>	Optional

Resources		Info
	<b>Analog Output</b>	Optional
<b>Aux Analogs</b>	<b>Aux Analog Input</b>	Optional
	<b>Aux Analog Output</b>	Optional
<b>Digitals</b>	<b>Digital Output</b>	Optional
	<b>Digital Input</b>	Optional
<b>Aux Digitals</b>	<b>Aux Digital Output</b>	Optional
	<b>Aux Digital Input</b>	Optional
	<b>DDS Waveform Generation</b>	Optional

**Table 6: Resources for image acquisition profile (FlexRIO)**

Resources		Info
	<b>Common</b>	Mandatory
	<b>Image acquisition (HOST or GPU)</b>	Mandatory
	<b>Serial communication</b>	Mandatory
<b>Aux Analogs</b>	<b>Aux Analog Input</b>	Optional
	<b>Aux Analog Output</b>	Optional
<b>Digitals</b>	<b>Digital Output</b>	Optional
	<b>Digital Input</b>	Optional
<b>Aux Digitals</b>	<b>Aux Digital Input</b>	Optional
	<b>Aux Digital Output</b>	Optional

**DAQStartStop:** Control register Boolean type. This element is the register used to start and stop the data acquisition/generation in the RIO device. This terminal will start data acquisition/generation process in all the FPGA resources.

**DebugMode:** Control register Boolean type. This element is the register used to simulate the data acquired by the device. The behaviour of the simulation mode is defined by the developer.

## 5.3 Analog Signal Data acquisition profile

### 5.3.1 Mandatory resources for data acquisition profile.

Fig. 8 summarizes the different resources needed to implement the data acquisition profile (defined here as coreDAQ). This profile sends data to host memory. These resources are:

**DMATtoHOSTNCh:** U16 array indicator. This indicator has the information about the number of DMA channels implemented (the array size) and channels allocated in the different DMAs. The values of the different array elements are the number of channels. A group is defined as the set of channels included in one DMA.

**DMATtoHOSTFrameType:** U8 array indicator. This array must have the same dimension and size that DMATtoHostNCh. Every element in the array contains the data format used for the DMA data. The possible values for FlexRIO are: Format A and Format B.

Table 7: Possible values for an element in DMATtoHOSTFrameType array

DMATtoHostFrameType[index]	Info
0	Format A
1	Format B

**DMATtoHOSTSampleSize:** U8 array indicator. This array has the same dimension size that DMATtoHOSTNCh. Each element of the array contains the number of bytes used per sample. In a specific design all the channels included in DMA must have the same value of this parameter for all channels. Table 8 presents the valid values.

Table 8: Valid sample size in bytes.

DMATtoHOSTSampleSize [index]	Info
0	Not valid
1	1 sample is one byte
2	1 sample is 2 bytes
4	1 sample is 4 bytes
8	1 sample is 8 bytes



**[Example for PXIe7966R/NI5761]:** DMATtoHOSTNCh[2]={1,1} This means that we have one DMA with one channel, and another DMA with another one. One possible case is to acquire the signal with the first DMA and the FFT with the second one. DMATtoHOSTFrameType [2]={0, 0}, this is the same frame

type for both DMAs. **DMATtoHOSTSampleSize [2]={2,8}**, the signal samples have two bytes and the FFT is estimated with 32 bits for real part and 32 bits for imaginary part. Problem: If results are longer than 64 bits packaging will be required. This will complicate the design.

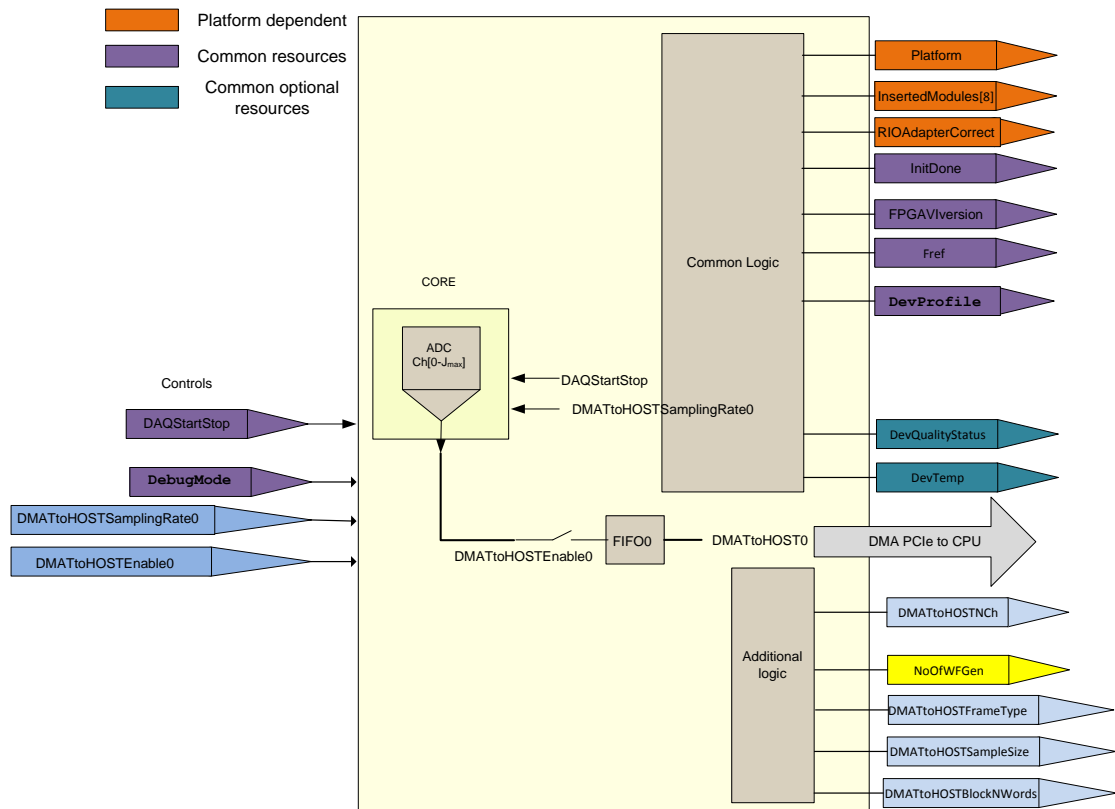
**DMATtoHOSTBlockNWords<n>**: U16 array. This array has the same dimension and size than the previous ones. Each element contains the length of the block used in the data acquisition. This terminal informs to the software layer about the frame length. A frame is a set of samples of the different channels involved. The length of the block is defined as S.

**DMATtoHOST<n>**: This element is a target to host FIFO FPGA memory which means that this memory is inside the FPGA. This memory is different of the DRAM memory externally located close to the FPGA (this depends on the FlexRIO used). This FIFO is always a 64-bit-wide FIFO connected to a DMA channel to send data to the HOST. The maximum number of FIFO DMAs is 16 for FlexRIO devices. Each DMA channel will send data acquired from a set of channels. We define and call this a DMA group.

**DMATtoHOSTSamplingRate<n>**: Control register. U16. There must be as many “SamplingRate” controls as DMAs used to pass acquired data to the CPU. The data acquired will be packaged into groups of channels and then flow through each DMA. See point 5.3.2 to understand how information is formatted. The label used must be enumerated from 0 to  $I_{\max}-1$ .  $I_{\max}$  is the maximum number of DMA channels available for the FlexRIO device (16). If the design includes more than one DMA, there will be a set of controls that we can define as **SamplingRate[0.. $I_{\max}-1$ ]**. These terminals control the sampling frequency from DMA group 0 to  $I_{\max}-1$ .

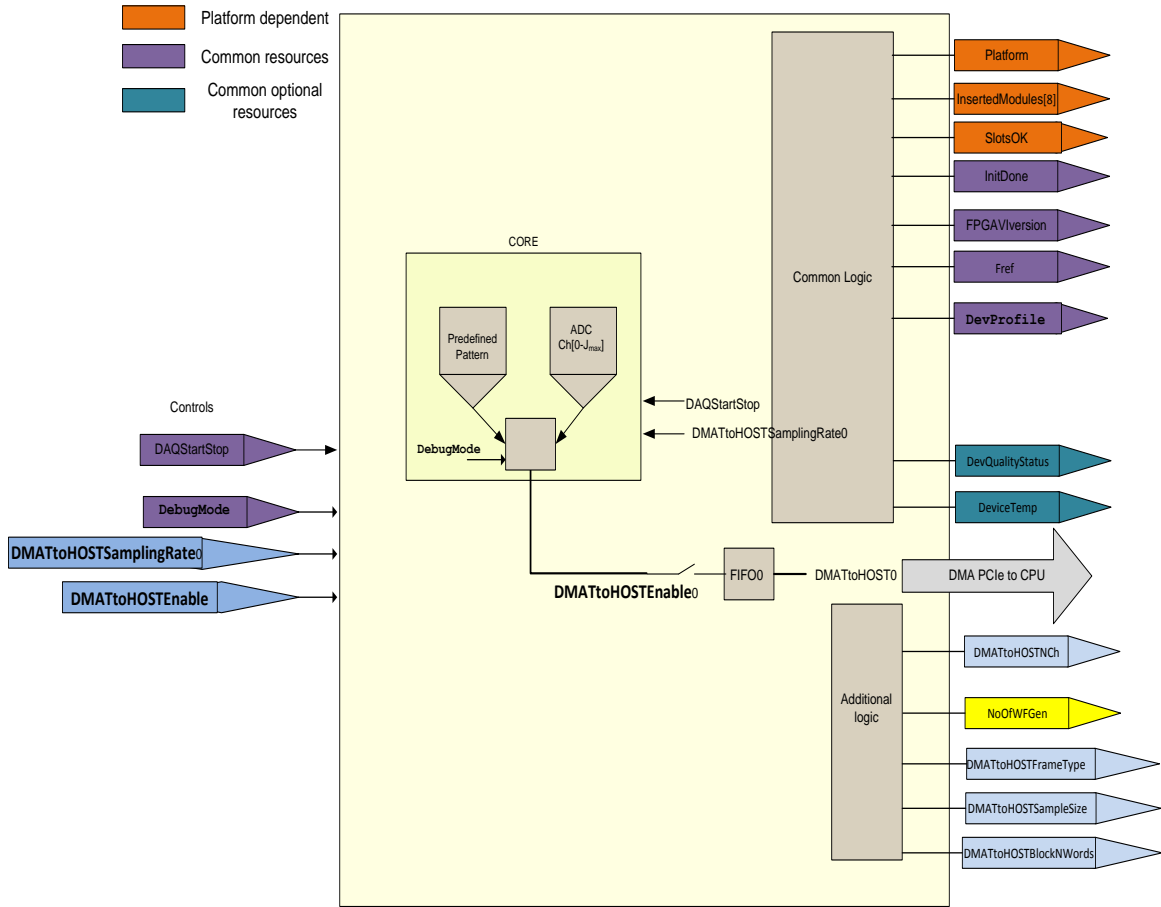
**DMATtoHOSToverflows**: U16 indicator. Each bit of this indicator will show the status of each of the device’s possible DMAs. The status will be either Correct (0) or Overflow (1).

**DMATtoHOSTEnable<n>**: Boolean register control. There are as many GroupEnable controls (**GroupEnable[0.. $I_{\max}-1$ ]**) as there are DMA groups. The data of the group are acquired if this control is set to true.



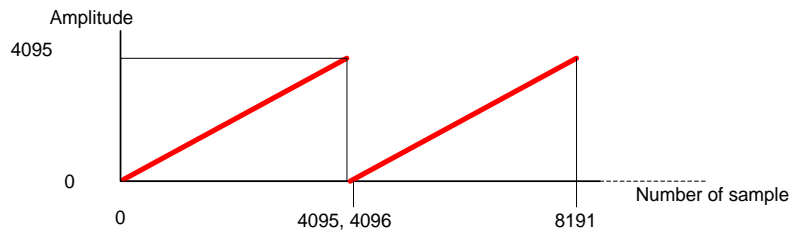
**Fig. 8: coreDAQ. Definition of minimum elements for implementing a data acquisition system in a RIO device**

The DAQ profile can be improved for supporting the debugging functionalities. Fig. 9 shows the idea of debugging the DAQ system using predefined tests patterns.



**Fig. 9: Adding Ramp Pattern Simulation to the design.**

For instance a simple hardware can be added to generate a periodic ramp signal in the FPGA to simulate the acquisition and allow the user to test the design. The pattern could follow, for instance, a ramp shape with a maximum value equals to the number of elements in a block (Fig. 10).



**Fig. 10: Ramp pattern generated by the FPGA.**

To change the operation of coreDAQ to simulation mode, we use a control register referred to as `DebugMode`

### 5.3.2 Data format in the DMA for Data acquisition profile.

The data acquisition profile is oriented to the acquisition of analog input channels and it supports different formats in the data stream sent to the HOST using the DMA [RD5]. The organization of

the information has a strong dependency with the features available in the different bundles (FlexRIO plus adapter module).

#### **5.3.2.1     *PXle 7966R / 5761R***

This bundle provides 4 analog input channels with 16 bits of resolution with a sampling rate up to 250MS/s. The user can use different combinations from 1 to 4 channels. Anyway, the DMA will be always organized using U64 words. The data frame will be organized in a block of size S element. This bundle also provide 8 DIO at a maximum sampling/update rate of 500kHz, the format for this DMA can be also Format A or B.

#### **5.3.2.2     *PXle 7961R / 6581R***

This bundle provides 54 DIO at a maximum sampling/update rate of 100MHz. The format for DMA will be also format A or B.

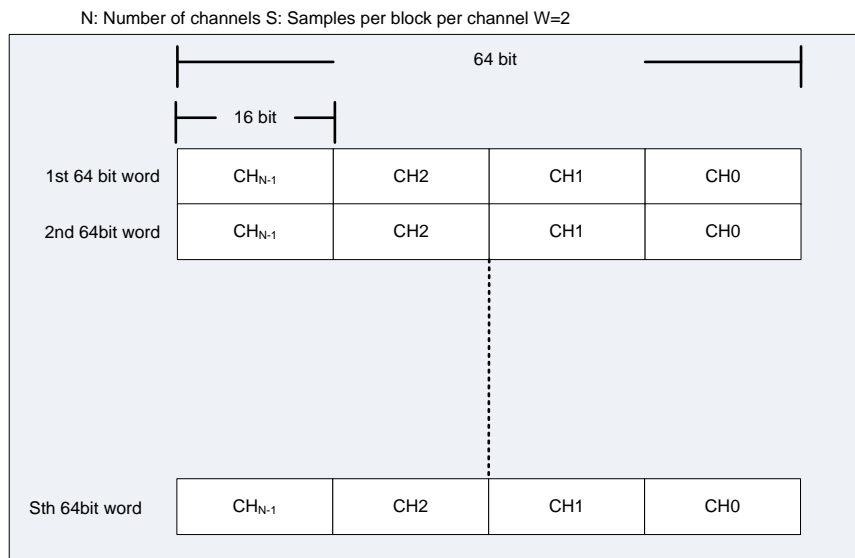
#### **5.3.2.3     *PXle 7961R or PXle7966R without adapter module***

For debugging and testing purposes it is possible to define a design without an adapter module. In this case the user also has to apply the format A or B.

#### **5.3.2.4     *Format A: DAQ samples***

The data in the DMA must be formatted according to the following rules:

- The number of channels N is variable between 1 and 32. N is configured in the FPGA for every DMA using the corresponding `DMATtoHOSTNCh[i]` element.
- W: Bytes used per sample. W=2 for instance for 5761R. All channels in the DMA use the same W. The valid values for bytes used per samples are 1, 2, 4 or 8. W is specified in the `DMATtoHOSTSampleSize` array.
- S is the number of samples S in a block. Every block has a length of U64 data with S samples (the number of channels included is defined with N). S must be an integer number multiple of  $N*W/4$ . This value is specified using the `DMATtoHOSTBlockNWords` array.
- The acquired data must be always encapsulated in 64-bit words of the DMA FIFO.



**Fig. 11: Data organization in the DMA. Example for N=4**



**Examples. Using two channels in NI5761R. N=2. W=2 S=1024.**

In this example the block is S=1024 U64 words. This means that there are 2048 samples per channel in a Block.  $N*W*S/4=1024$ .



**Examples. Using one channel in NI5761R. N=1. W=2 S=1024.**

In this example the block is S=1024 U64 words. This means that there are 4096 samples per channel in a Block.  $N*W*S/4=512$ .



**Examples. Using three channels in NI5761R. N=3. W=2 S=?**

In this example the block is S=1024 U64 words. This means that there are 4096 samples per channel in a Block.  $N*W*S/4 = k$ , S have to be a multiple of 6. For instance, 600. The number of samples per channel will be 800.



**Examples. Using 8 Digital inputs in NI6581R. N=1. W=1 S=1024.**

For NI6581R the minimum number of channels to read is 8. This implies the use of 1 byte and W=1. In this example the block is S=1024 U64 words. This means that there are 1024 samples per digital line grouped in a port of eight bits.



**Examples. Using 54 Digital inputs in NI6581R. N=7. W=1 S=?**

The acquisition of the 54 digital input lines implies the use of 7 bytes. Here the number of bytes per sample is 1. 1 byte means 8 Digital input lines. For instance, S=700

**Examples. Using 8 Digital inputs in NI6581R. N=1. W=1 S=1024.**

For NI6581R the minimum number of channels to read is 8. This implies the use of 1 byte and W=1. In this example the block is S=1024 U64 words. This means that there are 1024 samples per digital line grouped in a port of eight bits.



**Warning. The correct organization of DMA data frame is responsibility of the designer. This must follow the LabVIEW for FPGA [RD5]**

### 5.3.2.5 *Format B (TBD)*

Format B is not defined yet but it will allow to define a format including the timestamp of the data acquisition device.

## 5.3.3 Optional resources

### 5.3.3.1 *Analog inputs*

**AI<x>:** I32 indicator. The FPGA LabVIEW programmer can add read-only registers (indicators) with the last sample acquired from an analog input (see Fig. 12). This indicator will be updated at the sampling rate programmed for the channel. The nomenclature for naming the indicator will be "AI" followed by the number of the channel. The maximum number of AI terminals is 4.  $x \in [0,3]$ .

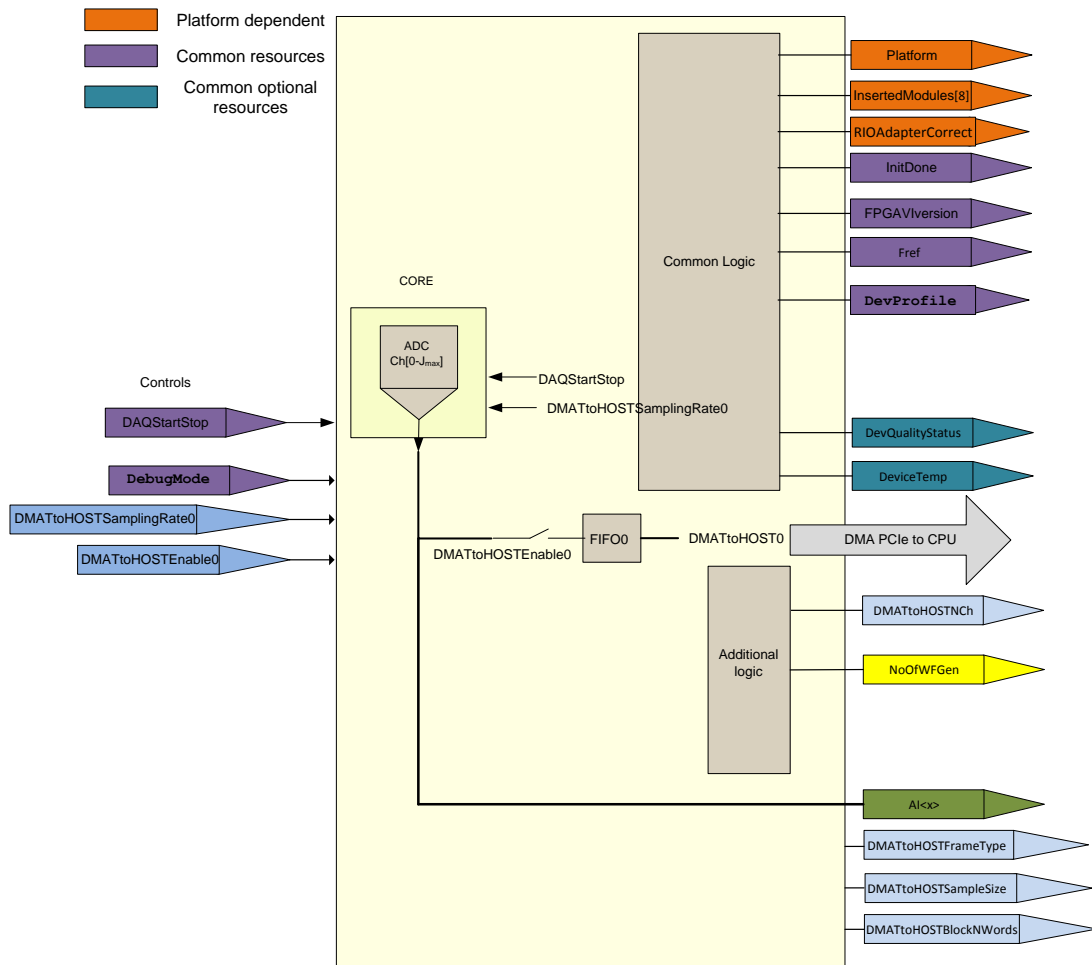


Fig. 12: Example of how to add an analog input terminal. X can be a value in the range 0 .. 3.

### 5.3.3.2 Auxiliary analog inputs

**auxAI<x>**: I32 indicator. The FPGA designer can include indicators identified as auxAI<x> with LabVIEW I32 data type representing any internal variable in the FPGA. These are the user defined registers and therefore the functionality is totally defined by the designer. For instance (Fig. 13), you can acquire one sample from adapter module analog input channels (I16), operate the data and connect the result to an I32 terminal labelled as auxAI0. The maximum number of auxAI is 16.

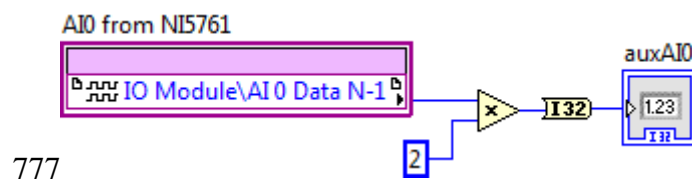


Fig. 13: Example of auxiliary analog input terminal

### 5.3.3.3 Analog Output

**AO<x>**: I32 indicator. The number of analog outputs available in FlexRIO technology is limited to two. The identification of the terminals (controls) used to drive this analog output must be AO0 and AO1. These terminals will be connected to the physical I/O nodes available in the adapter module.



**Warning. Analog outputs are not included in any of the adapter modules currently supported in the catalogue.**

The update of the output is validated using a terminal defined as AOEnable<n>. This enables or disables the signal generation: 0 means that output is always zero and 1 means that output is updated. If you add to the design an AO<x> terminal the corresponding AOEnable<x> is mandatory.



**Warning. The update of analog outputs is validated with the AOEnable terminal.**

### 5.3.3.4 Auxiliary analog output

**auxAO<x>**: I32 control. The FPGA designer can include controls identified as auxAO<x> with LabVIEW I32 data type representing any internal variable in the FPGA. These are the user defined registers and therefore the functionality is totally defined by the designed. For instance Fig. 14 shows an example of how to multiply the input value acquired from AI0 channel by the value available in the terminal auxAO0. The maximum number of auxAO is 16.

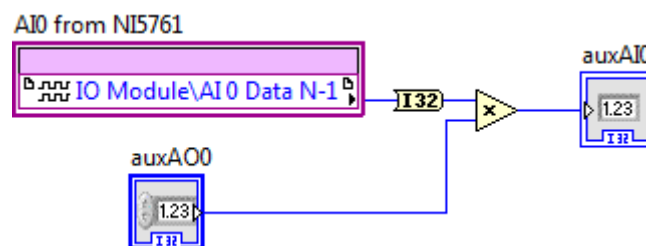


Fig. 14: Example of auxiliary analog output terminal

### 5.3.3.5 Digital input/output

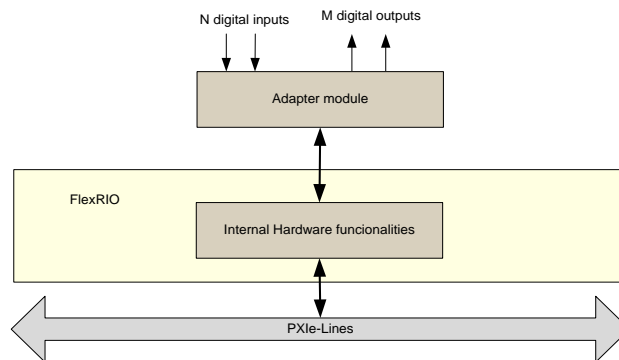
The FlexRIO bundle has a lot of physical digital input/outputs. Some of them are available in the adapter module's connector and others in the PXI/PXIe bus. Additionally, the user can add any digital input/output terminals to the design (see Fig. 15) only available inside the FPGA.



**Digital I/O in NI6581.** NI6581 adapter module can be configured to work on port mode or in line mode. In this point we are considering that the user wants to manage the digital lines individually independently of the configuration of NI658.

**Digital I/O in NI1483.** NI1483 provides only 4 I/O lines that can be independently configured as input or output.

**Digital I/O in NI5761.** NI5761 provides 8 bidirectional lines named AUXI/O[0..7].



**Fig. 15: Digital input/output.**

**DO<n>:** Boolean control. The FPGA designer can include controls identified as DO<n>. These controls will be connected physically to digital outputs in a FlexRIO adapter module. This is valid for 5761R, 6581R and NI1483 (see Fig. 16, Fig. 17 and Fig. 18). The designer has to check the technical details provided by the manufacturer in the product specifications.



**Fig. 16: Example of connection for DO0 for NI6581 Adapter module.**

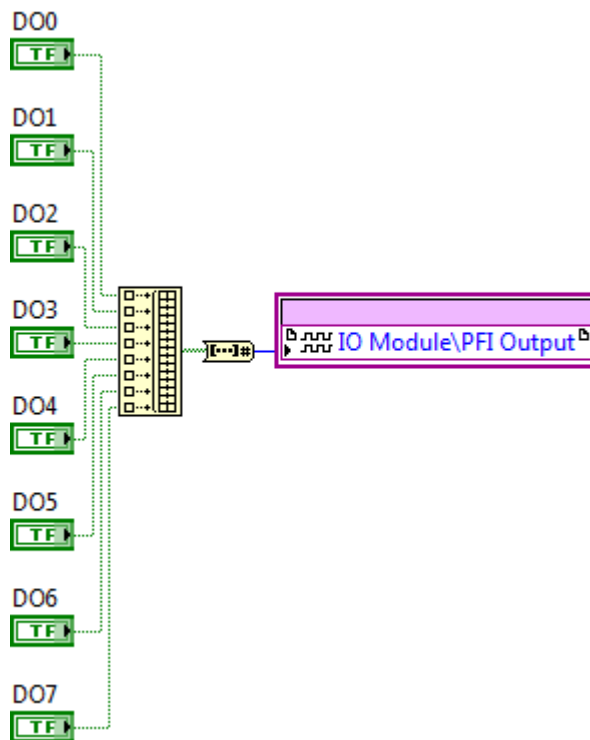


Fig. 17: connection of eight Digital outputs in NI5761R.

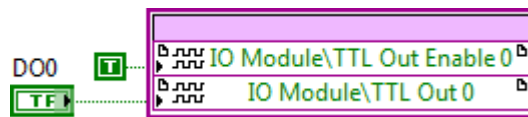


Fig. 18: Connection of one digital output in NI1483R.

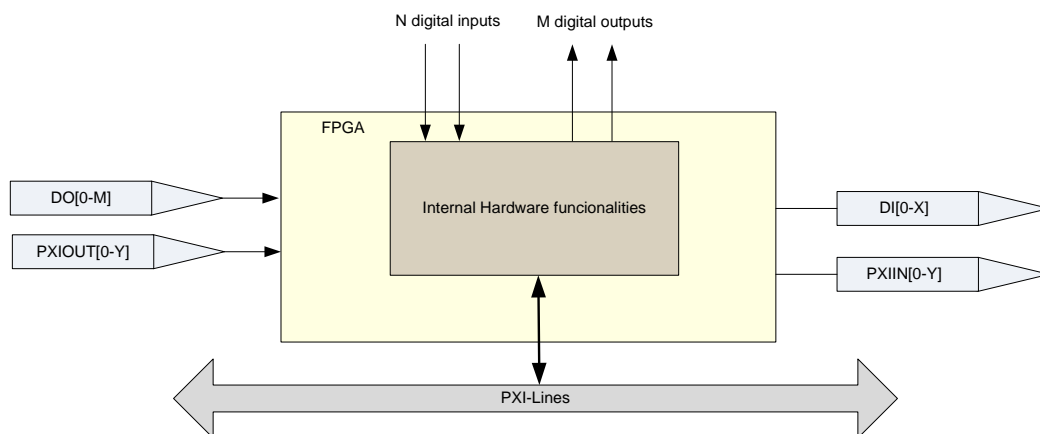
**DI<n>:** Boolean indicator. The FPGA designer can include indicators identified as DI<n>. These indicators will be connected to physical digital inputs in a FlexRIO adapter module. Valid for 5761R, 6581R and NI1483. The designer has to check the technical details provided by the manufacturer in the product specifications.

**auxDO<n>:** Boolean control. The FPGA designer can include controls identified as DO<n>. These controls will be connected to internal FPGA signals. Valid for 5761R and 6581R

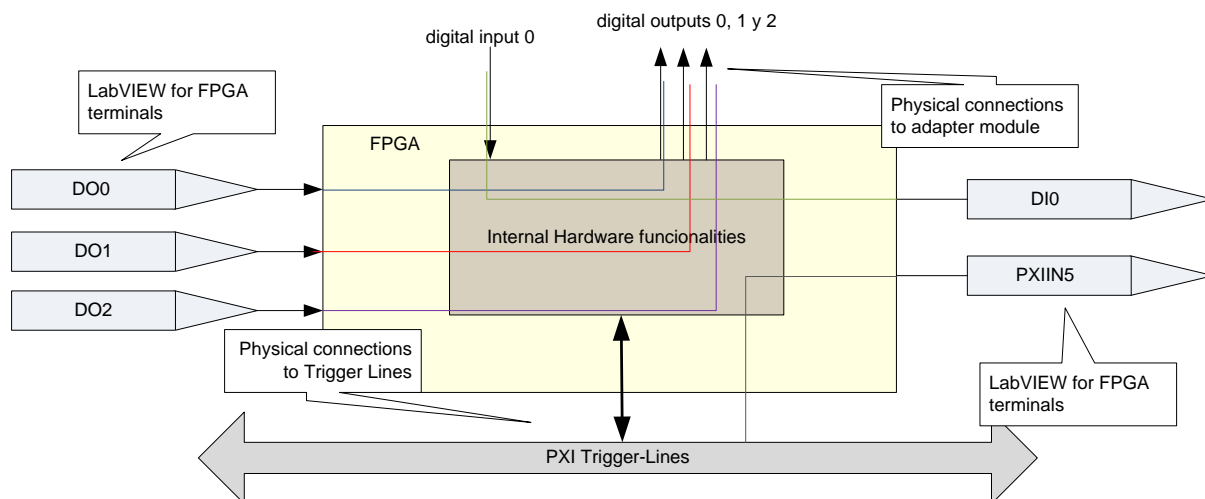
**auxDI<n>:** Boolean indicator. The FPGA designer can include indicators identified as auxDI<n>. These indicators will be connected to internal FPGA signals.

#### 5.3.3.5.1 Examples of using Digital I/O

The digital lines can be used in various ways: they can be connected to other digital lines. Fig. 19 to Fig. 23 show some digital input/output use cases.

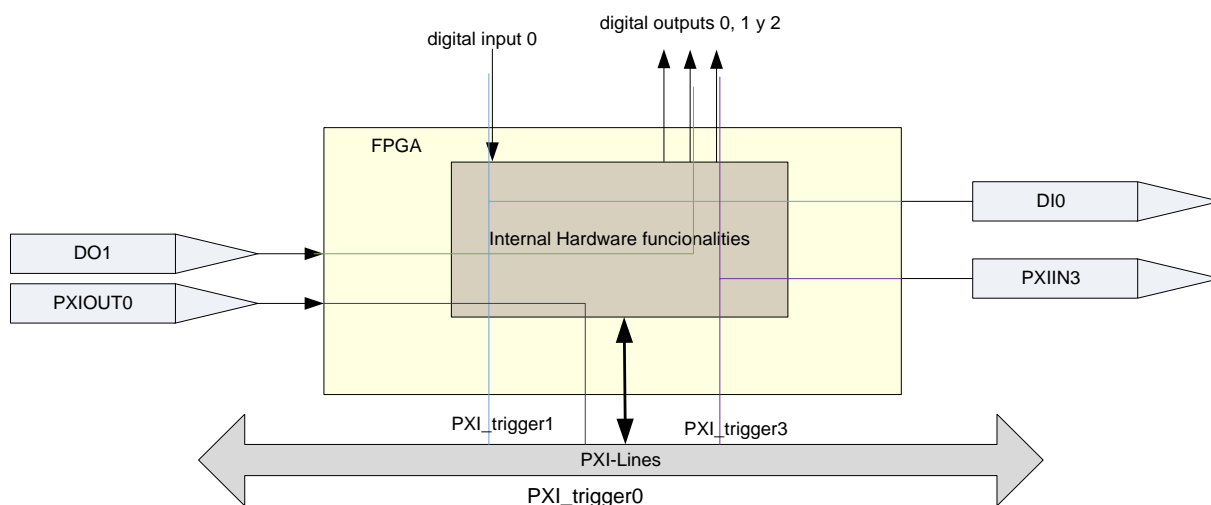


**Fig. 19: Use of digital input/output. DO range from 0 to 95. PXIOUT can range from 0 to 7. PXIIN can range from 0 to 7. DI can range from 0 to 95.**



**Fig. 20: Example 1 of digital I/O applications.**

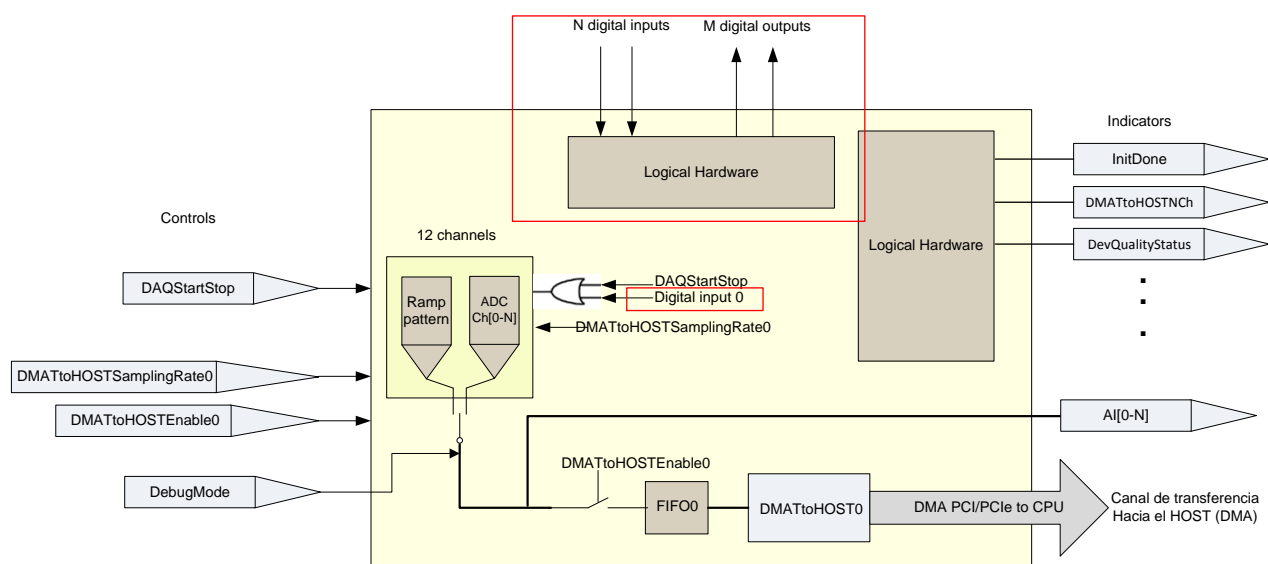
In Example 1, writing to control terminals named DO0, DO1 and DO2 will have direct impact on the states of digital output lines 0, 1 and 2, respectively. The activation and deactivation of the digital input line zero will be registered by terminal indicator named DI0. The activation and deactivation of PXI\_trigger line 5 will be registered by terminal indicator identified as PXIIN5.



**Fig. 21: Example 2 of digital I/O applications**

In this example (Example 2), the digital input line number 0 passes through the internal FPGA logic of the hardware and drives the line PXI\_trigger1. Additionally, the software can monitor the line using the terminal indicator DI0. With the terminal DO1, the user can control some internal hardware functionalities and other digital output lines. The register PXIOUT0 can enable or disable some functionality and drive the PXI\_trigger0 line. The digital output line 0 is activated by the FPGA when an internal event occurs. The values received in PXI\_trigger3 can be routed to a specific output line and monitored with a register.

A more real example is the control of the start and stop of data acquisition in coreDAQ using a digital line. Fig. 22 shows how to add this functionality to the previous coreDAQ example.



**Fig. 22: Example 3 of digital I/O applications**

Finally, Fig. 23 shows a LabVIEW for FPGA example using digital input/output lines in NI5781 adapter module.

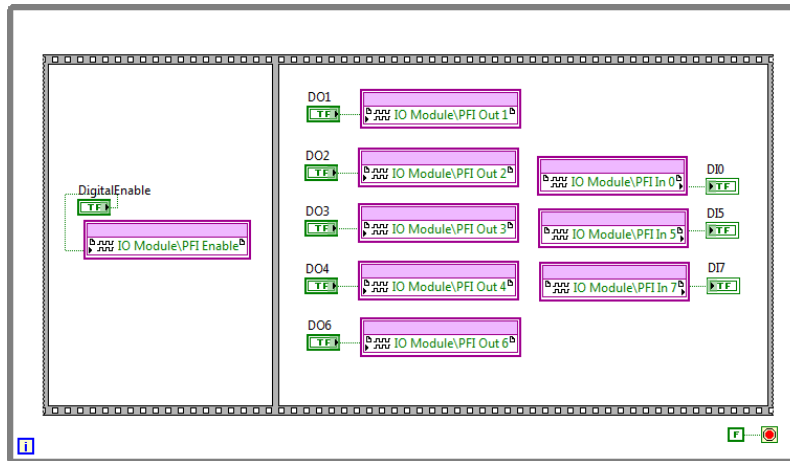


Fig. 23: LabVIEW implementation of Digital I/O using NI5781.



### Use of digital signals.

The aim of using digital signals is to provide the designer different degrees of freedom to customize the data acquisition system. Special triggers, trigger qualifiers can be implemented using these digital lines.

#### 5.3.3.6 Signal generator

**SGNo:** U8 indicator. This indicator is initialised with the number of waveform generators included in the design. Value zero means no signal generator implemented. The values allowed are from 0 to 2 (see Fig. 24).

In the RIO device, the user can add an element to implement signal generation using the analog outputs. The templates provide a signal generator implemented with direct digital synthesis (DDS) technique. In this method, the FPGA contains a memory with a predefined pattern. The details of the implementation are explained in the document [RD7]. The terminals available to use this block are described in Table 9.

Table 9: Terminals used by the signal generator

LabVIEW Terminal Name	Type	Functionality	Notes
SGFreq<n>	U32, Control	Frequency of the signal to be generated	The desired frequency(freq) in Hertz and the terminal the value following this equation $\text{Terminal value} = \frac{\text{freq} \times \text{LoopRate}}{\text{Fref} \times 2^{32}}$
SGAmp<n>	U16,	Amplitude of the signal to be	The value to be written in the terminal must be 0 to 4095.



LabVIEW Terminal Name	Type	Functionality	Notes
	Control	generated	
<b>SGPhase&lt;n&gt;</b>	U32, Control	Phase control for the signal	The terminal contains the phase shift
<b>SGSignalType&lt;n&gt;</b>	U8, Control	Signal type among DC, Sine, Triangular and Square	Enumerated value to select the signal needed
<b>SGUpdateRate&lt;n&gt;</b>	U32, control	Update rate frequency used for signal generation	The FPGA terminal needs a value that is the division of $F_{ref}/\{\text{desired AOUpdateRate}\}$ . For instance if the AOUpdateRate desired (PV value) is 50MS/S and the Fref frequency in the FPGA is 100MHz, the value to be written in AOUpdateRate terminal must be 2. This relationship is defined as the “looprate” for the signal generation
<b>SGFref&lt;n&gt;</b>	U32, Indicator	Defines the reference frequency used by the signal generator	Fref for signal generation

### 5.3.4 LabVIEW for FPGA VI implementation for data acquisition profile

The hardware architecture implemented in the FPGA will be composed of a specific hardware devoted to data acquisition applications. This hardware is referenced in the document as coreDAQ. Additional elements can be included based on the decisions taken by the designer and the resources available in both the FPGA itself and the RIO device used. The use of this coreDAQ has some limitations that must be considered. The implementation of a general-purpose DAQ device with multiple functionalities is not possible in a RIO device because the resources available in the device are limited. The insertion of the coreDAQ in the design is mandatory when you are using the data acquisition profile (DevProfile=1).

The LabVIEW/FPGA VI implementing the solution is explained in the following paragraphs. The first part of the VI has to include the necessary elements to perform the description of the design and initialize all the resources. Fig. 24 displays the LabVIEW code used to implement the basic initialization of the terminals.

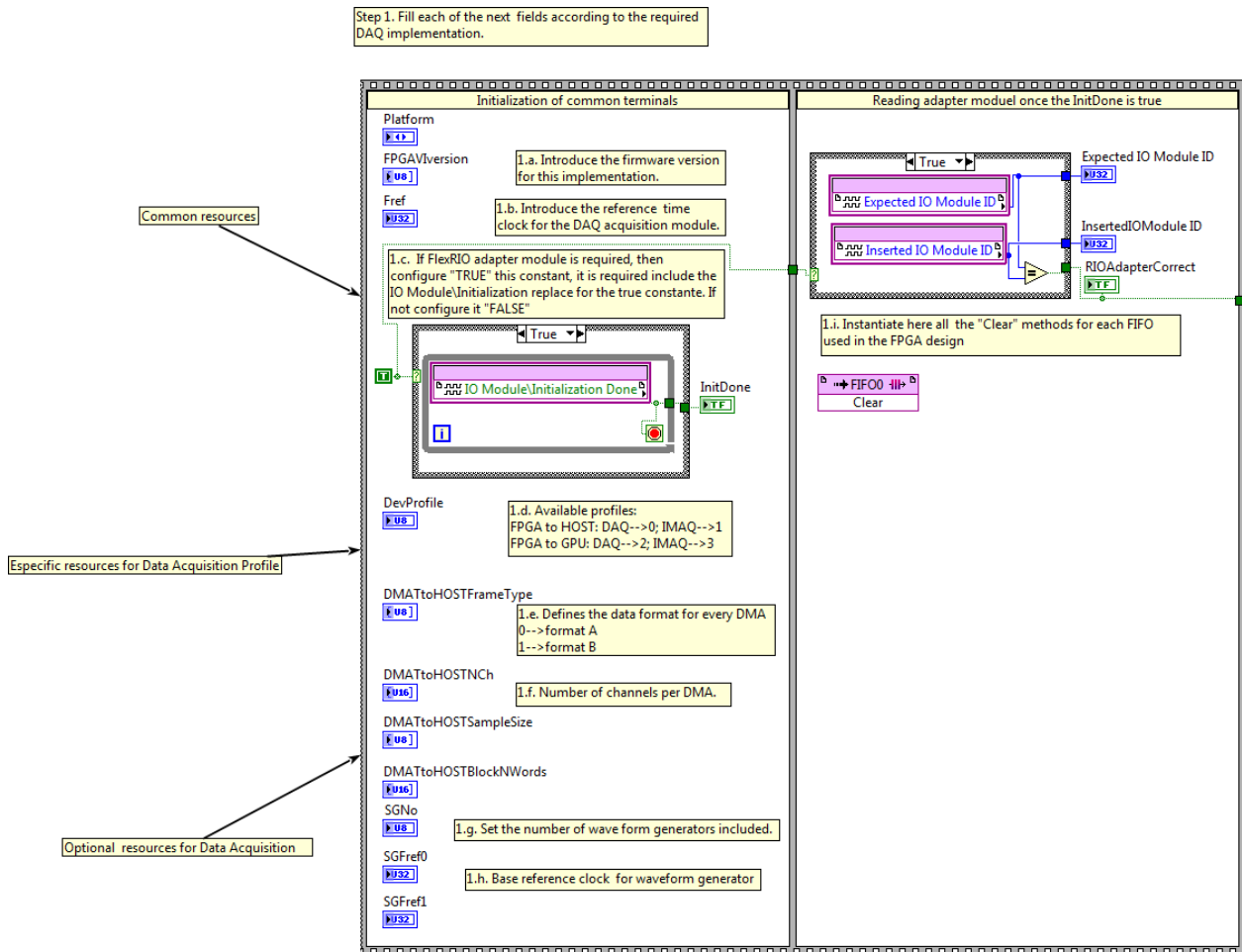


Fig. 24: Initialization examples of some mandatory terminals in LabVIEW/FPGA. Part a



**Default values for terminals:** The information provided in some terminals by the designer is essential for the software layer using the FPGA resources. For instance terminal Platform defined the hardware used in the implementation in LabVIEW you need to add an indicator in the VI Front Panel with the label Platform. Then you need to select FlexRIO and select Edit->Make Current Values Defaults. This terminal is read by the software layer before the FPGA is running; therefore, the only method to define a default value is this.

This code initializes the FPGA resources and verifies the correct installation of the adapter module. Once the initialization finalizes, the InitDone terminal is set to true. The implementation of the coreDAQ is represented in Fig. 25. This shows the basic template provided as pattern.

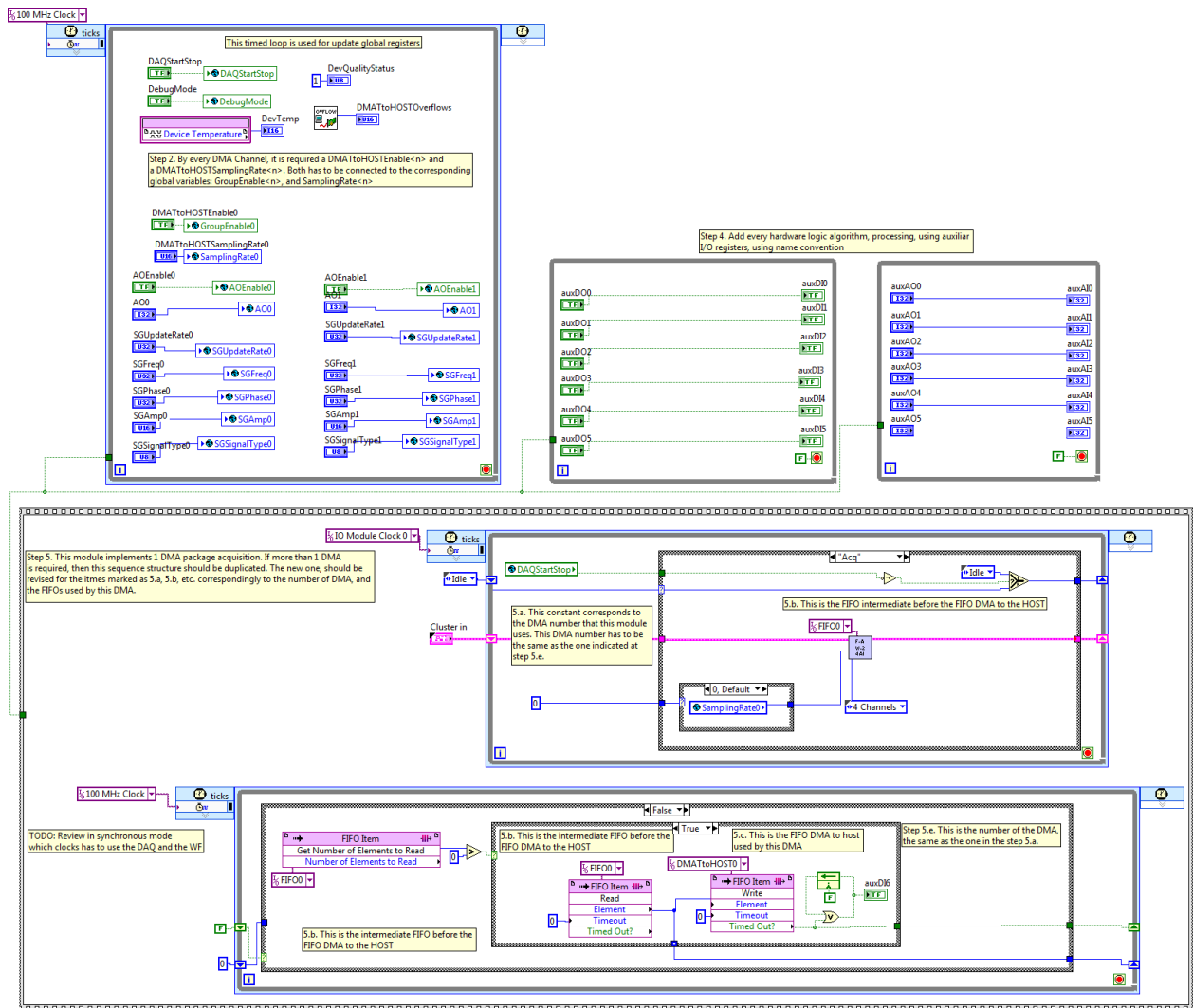


Fig. 25: Implementation of coreDAQ.

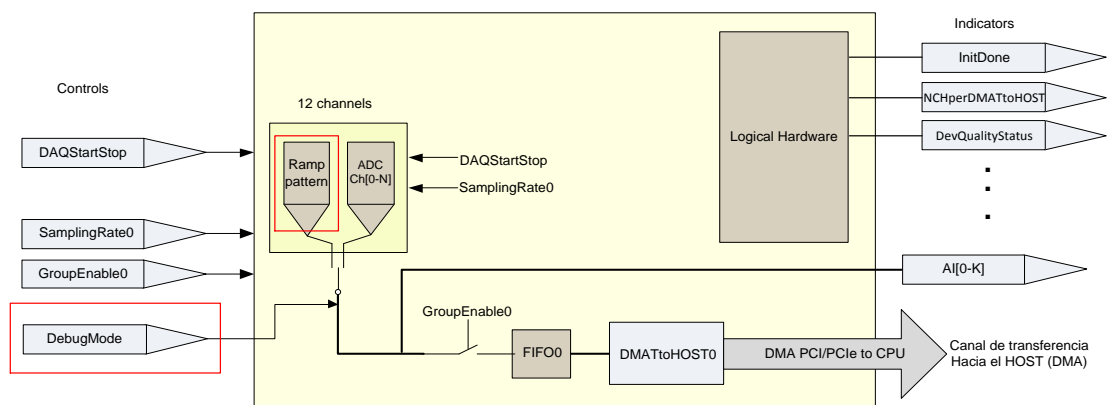


Fig. 26: Adding Ramp Pattern Simulation to the design.

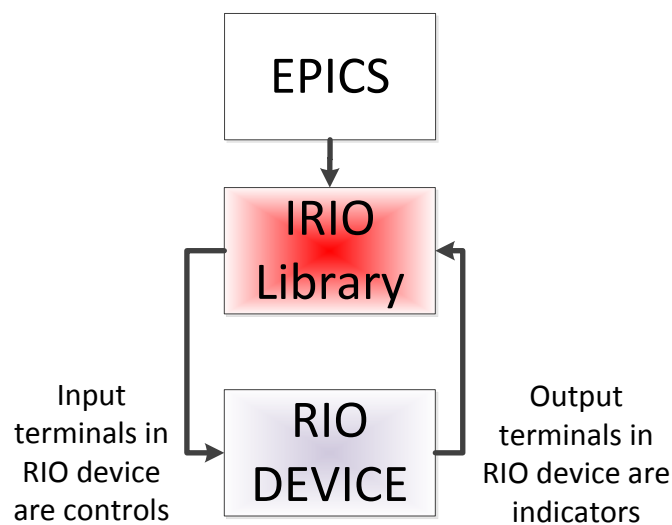
#### 5.3.4.1 *Rules to be applied when designing for LabVIEW/FPGA for FlexRIO data acquisition profile*



**[Important]:** This point is important if you are trying to modify the templates provided.

The user can change and modify the patterns provided in order to implement its specific application. If you change the code you need to observe the following main assumptions:

1. The configuration of coreDAQ will be controlled writing and reading registers. In LabVIEW for FPGA terminology, these registers are controls (for writing) and indicators (for reading). These FPGA terminals will be used by upper software layers; see Fig. 27. The IRIO layer using the NI-RIO Linux Device Driver tries to search these terminals using their names (label). If you do not use the correct labels the driver will not work correctly (labels are case sensitive).



**Fig. 27: Software applications read and write from/to RIO devices using a software interface.**

2. The data types supported for these terminals are summarized in Table 10.

**Table 10: LabVIEW for FPGA Data types summary.**

LabVIEW data Type	Supported	Scalar/Array	FIFOs (Target to HOST, Host to target)	Notes
<b>Boolean</b>	Control and indicator	Supported	Supported	
<b>I8</b>	Control and	Supported	Supported	

LabVIEW data Type	Supported	Scalar/Array	FIFOs (Target to HOST, Host to target)	Notes
	indicator			
<b>U8</b>	Control and indicator	Supported	Supported	
<b>I16</b>	Control and indicator	Supported	Supported	
<b>U16</b>	Control and indicator	Supported	Supported	
<b>I32</b>	Control and indicator	Supported	Supported	
<b>U32</b>	Control and indicator	Supported	Supported	
<b>I64</b>	Control and indicator	Supported	Supported	
<b>U64</b>	Control and indicator	Supported	Supported	
<b>FXP</b>	Control and indicator	Not supported	Not supported	Not supported by NI-RIO Linux Device Driver
<b>SGL</b>	Control and indicator	Not supported	Not supported	Not supported by NI-RIO Linux Device Driver
<b>DBL</b>	Control and indicator	Not supported	Not supported	Not supported by LabVIEW/FPGA
<b>EXT</b>	Control and indicator	Not supported	Not supported	Not supported by LabVIEW/FPGA
<b>CXT/CDB/CSG</b>	Control and indicator	Not supported	Not supported	Not supported by LabVIEW/FPGA
<b>Clusters</b>	Control and indicator	Not supported	Not supported	Not supported by NI-RIO Linux Device Driver



**Warning.** The designer must use the detailed data types and the label names.

3. There will be a **U8** indicator named **Platform** initialized to zero (FlexRIO platform), Platform=0.
4. There will be a **U8** array of 2 elements, named **FPGAVersion**, which is initialised with the VI version MM.mm. The first element will include the major version “MM”, and the second element will have the minor version, “mm”. For instance, FPGAVersion[2]={2,12} means V2.12 (see Fig. 28).

The screenshot shows the 'Platform' tab of a configuration window. It contains the following elements:

- FPGAVersion**: A U8 array with two input fields, both containing the value '1'.
- Expected IO Module ID**: A text input field containing the value '0'.
- InsertedIOModule ID**: A text input field containing the value '0'.
- InitDone**: A green circular indicator light.
- RIOAdapterCorrect**: A green circular indicator light.
- Fref**: A text input field containing the value '125000000'.
- DevQualityStatus**: A text input field containing the value '0'.
- DevTemp**: A text input field containing the value '-10244'.
- DevProfile**: A text input field containing the value 'x0'.
- DAQStartStop**: A green oval button with a right-pointing arrow.
- DebugMode**: A green oval button with a right-pointing arrow.

**Fig. 28: Versioning the bitfile.**

5. There will be an **U32** indicator named **ExpectedIOModuleID**. If the FPGA does not have an adapter module connected, then the programmer will initialise it with the value 0; if not, it will take the value from an I/O Modules Status element called Expected IO Module ID (see Fig. 24 and Fig. 28).
6. There will be a **Boolean** indicator named **RIOAdapterCorrect**. If the FPGA has an adapter module, it will be necessary to check whether the connected adapter module is the correct one as specified in a particular bitfile version. The IO Module Status element of the FPGA offers the Inserted IO Module ID element, which can be used for comparison with the Expected IO Module ID. Thus, it can be determined whether the RIO adapter is the correct one for the bitfile downloaded (see Fig. 24 and Fig. 28).

7. The designer has to decide when the FPGA is ready to work. This is done using the **InitDone** terminal. The default value is false (initialization value) and the user sets to true whether the initialization is complete and correct.

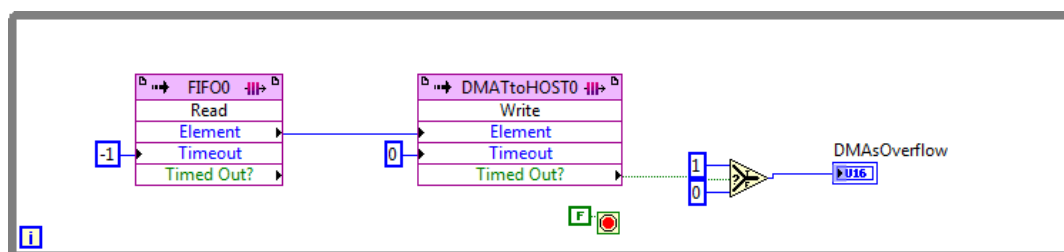


**Warning. The initialization is different for each adapter module. Check manufacturer information to confirm what is the best method to initialize the adapter module.**

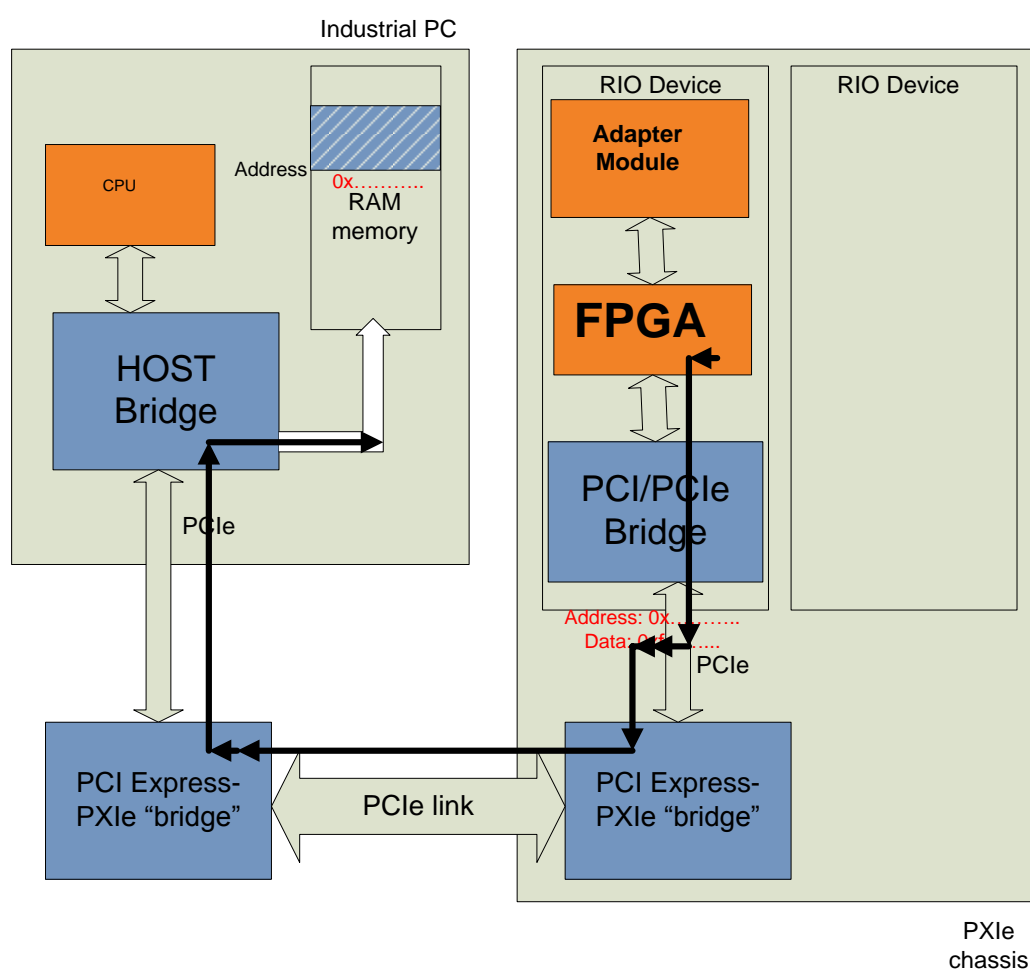
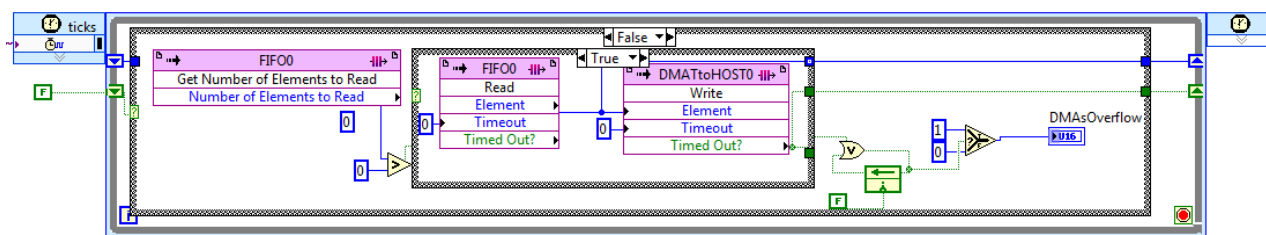
8. There will be a **U32** indicator named **Fref**, which will be initialised with the reference clock value for sampling purposes. The sampling frequency may vary, but it is always a divisor of the reference clock (see Fig. 24 and Fig. 28).
9. There will be a set of indicators that should be initialised with the corresponding values explained above. These indicators are **DevQualityStatus** and **DevTemp**,
10. DMA transfers are used to move large amounts of data, waveforms and/or images. Fig. 29 represents an example of the implementation of DMA movement in LabVIEW for a FPGA. There is a while loop (infinite loop) extracting the data from a FIFO memory and sending them to the HOST using the DMA. In LabVIEW for FPGA, this loop is implemented in the hardware using a 40 MHz clock. Therefore, if you need sampling rates above 40 MS/s, you must change the code (see Fig. 30). This code allows checking if data is available in the FIFO and moving it using DMA checking DMA overflow.



**[DMA concept]:** RIO devices with PCI and PCIe interfaces have bus master capability, which means that when the RIO device must move data to the host, it takes control of the PCI/PCIe bus to move the data directly to the host RAM memory. This functionality is traditionally known as direct memory access (DMA). The advantage of DMA is that it avoids the intervention of the CPU in data transference and therefore increases the performance in the data acquisition process (see Fig. 31).



**Fig. 29: Example of a LabVIEW/FPGA node to move data from the FPGA to the HOST.**



**Fig. 31: Representation of DMA in a PCIe architecture. DMA is implemented with bus master capabilities of PCIe devices.**

11. There should be at least one data acquisition channel moving the data from the FPGA to the host through DMA.
12. The use of DMAs is exclusively for the transfer of data acquired by the FPGA. These data are typically the samples of a set of channels of an adapter module connected to the FPGA. Fig. 32 shows how to obtain the ADC codes from channels AI0 and AI1, how to pack the data in a U64 word, and how to write the data to a FIFO memory. Remember that DMA data organization must follow the predefined format.



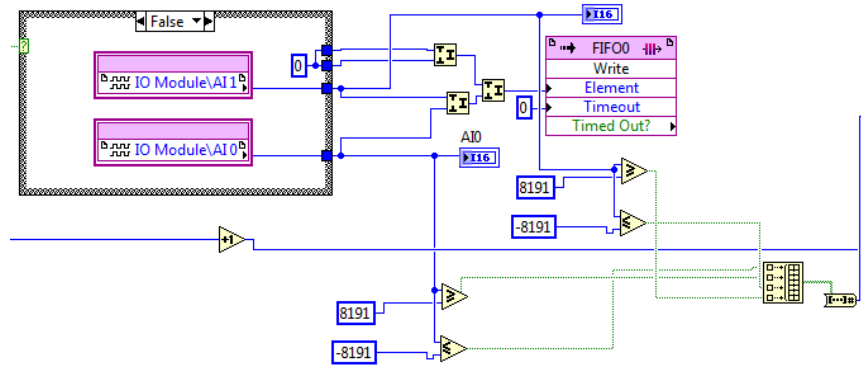


Fig. 32: LabVIEW code for reading analog input channels

13. A set of data acquisition channels is sent using DMAs. All the channels that are grouped in a DMA must have the same sampling frequency. Therefore, there will be as many sampling rate controls as DMAs (see Fig. 33).

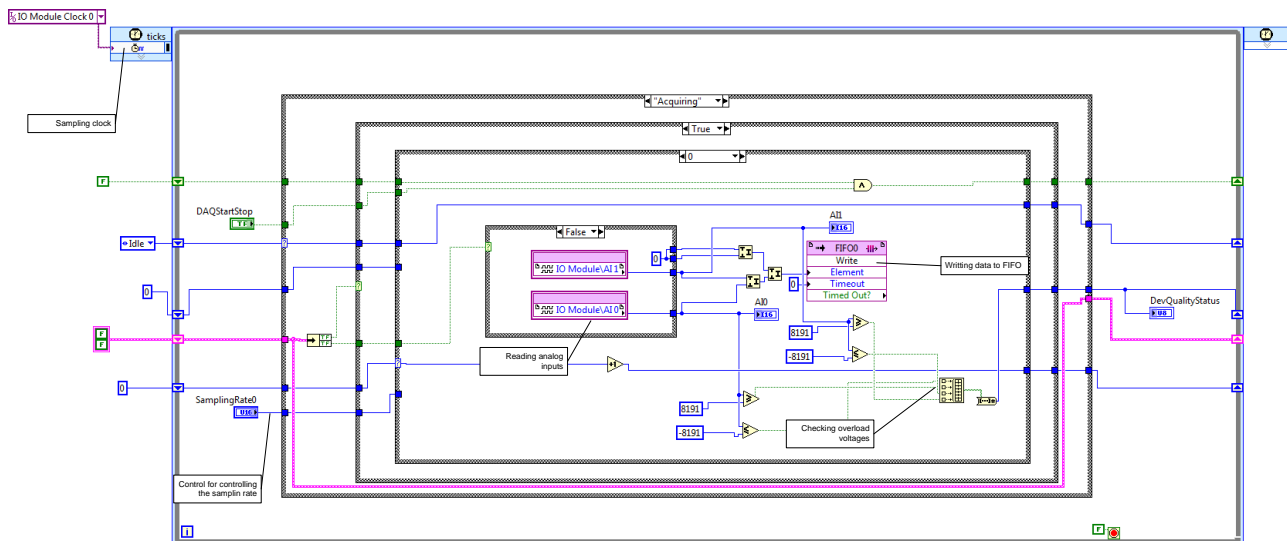


Fig. 33: Example of the implementation of data acquisition with LabVIEW for FPGA

14. A control to start and stop the acquisition by software is needed. The designer can include additional digital lines and/or state machines to meet its specific needs. For instance, a trigger can be implemented using a PXI trigger line. When the user starts the data acquisition the state machine transits to waiting for trigger state and then when trigger raises, the data acquisition starts.
15. There will be an **I16** array indicator named **DMATtoHOSTNCh**. The number of elements in this array (array size) must match the number of DMAs in use. Each element of the array must be initialised with the number of channels included in each DMA. For instance, a design using 3 DMA groups must contain an array defined as **DMATtoHOSTNCh [3] =**

{3,4,2}, which indicates that DMA 0 uses 3 channels, DMA 1 uses 4 channels, and DMA 2 uses 2 channels. This rule is a limitation because the user cannot change the number of elements included in the DMA stream during execution. Fig. 33 shows an example of how to define only one DMA channel with 2 channels in the stream.

16. The sampling frequency control is performed through a register. The number of these controls will be the same as the number of DMA groups. The value written to this register is the division factor to be applied to the reference clock to obtain the desired sampling rate. The reference clock is the clock used by the RIO device to perform its internal operations. Typical values for the reference clock are 40 MHz, 50 MHz, 100 MHz, and 200 MHz. For instance, module NI5761 uses a reference clock of 100 MHz.



Fig. 34: Sampling frequency control.

### 5.3.5 Summary of resources used for Data acquisition profile

Table 11 summarizes the terminals (control and indicators) used by data acquisition profile in FlexRIO platform. The templates for the bundles PXIe-7961R/NI6581 and PXIe-7966R/NI5761 (see paragraph 7.2) have been implemented using these terminals.

Table 11: Summary of the terminals used in data acquisition profile

Terminal Name	Data type	Type	Detail	Information	Values	Initialize before Run?
Platform	U8	Indicator	This terminal defines the form factor used in the FPGA implementation	Mandatory	0- FlexRIO 1- cRIO 2- R Series	YES
<b>Common Terminals for FlexRIO</b>						
FPGAVIversion	Array U8	Indicator	Contains the VI version, 2 elements. One for MM major version, and the next one mm minor version. MM.mm	Mandatory	For instance 1.1 FPGAVIversion[0]=1 FPGAVIversion[1]=1	YES
InitDone	Boolean	Indicator	This terminal must be set to true when the FPGA is	Mandatory	True=OK False=NOK	N/A

Terminal Name	Data type	Type	Detail	Information	Values	Initialize before Run?
			initialized			
RIOAdapterCorrect	Boolean	Indicator	Boolean indicating if the adapter module is the correct for the application	Mandatory	Defined by NI	NO
InsertedIOModuleID	U32	Indicator	Contains the Module ID of the corresponding module	Mandatory	Defined by NI	NO
Fref	U32	Indicator	Contains the Reference clock of the FPGA for sampling rate	Mandatory		YES
DevQualityStatus	U8	Indicator	This indicator will show the status of the acquisition	Mandatory		NO
DevTemp	I16	Indicator	This indicator will show the temperature of the FPGA	Mandatory		NO
Devprofile	U8	Indicator	This indicator defines the implementation in the FPGA (DAQ, Image, etc.)	Mandatory		NO
DAQStartStop	Boolean	Control	This terminal must be set to true to start data acquisition	Mandatory		NO
<b>Specific Terminals for data acquisition profile</b>						
DMATtoHOSTNCh	U16, array	Indicator	Describes the number of DMAs implemented in the FPGA. The array must be initializes with the number of channels available in	Mandatory		YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialize before Run?
			each DMA.			
DMATtoHOSTFrameType	U8, array	Indicator	Describes the frame type used in the DMA frame	Mandatory		YES
DMATtoHOSTSampleSize	U8, array	Indicator	Size in bytes for the channel sample	Mandatory		YES
DMATtoHOSTBlockNWords	U16, array	Indicator	Length of the block used for each DMA			YES
DebugMode	Boolean	Control	If debug is true the FPGA will simulate the acquired data. Otherwise, physical signals are acquired	Mandatory		
DMATtoHOST<n>	FIFO	DMA to HOST	FIFO memory in the FPGA	Mandatory	n={0 .. 16}	N/A
DMATtoHOSTSamplingRate<n>	U16	Control	Integer number obtained as Sampling rate/Fref	Mandatory	n={0 .. 16}	NO
DMATtoHOSTEnable<n>	Boolean	Control	Enable or disable write to DMA FIFO	Mandatory	n={0 .. 16}	YES
DMATtoHOSTOverflows	U16	Indicator	Status of the different DMA FIFO	Mandatory		YES
<b>Optional Resources</b>						
AI<n>	I32	Indicator	Digital sample for channel <n>	Optional	n={0 .. 3} 3 because of hardware limitation	
auxAI<n>	I32	Indicator	Auxiliary internal FPGA variables	Optional	n={0 .. 15}	
auxAO<n>	I16	Control	Auxiliary internal FPGA variables	Optional	n={0 .. 15}	

Terminal Name	Data type	Type	Detail	Information	Values	Initialize before Run?
DI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 7} NI5761R  n={0 .. 53} NI6581R	
DO<n>	Boolean	Control	Digital line	Optional	n={0 .. 7} NI5761R  n={0 .. 53} NI6581R	
auxDI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 15}	
auxDO<n>	Boolean	Control	Digital line	Optional	n={0 .. 15}	
SGNo	U8	Control	Number of waveform generators	Optional	0 .. 2	YES
SGSignalType<n>	U8	Control	Signal shape to be generated	Optional	n={1..2}	
SGAmp<n>	U32	Control	DSS accumulator increment	Optional	n={1..2}	
SGFreq<n>	U32	Control	Phase control	Optional	n={1..2}	
SGPhase<n>	U32	Control	Phase control	Optional	n={1..2}	
SGUpdateRate<n>	U32	Control	Update rate	Optional	n={1..2}	
SGFref<n>	U32	Indicator	Reference frequency	Optional	n={1..2}	

## 5.4 Image acquisition profile

### 5.4.1 Mandatory resources for Image acquisition profile

Fig. 35 summarizes the different resources needed to implement the image acquisition profile (defined here as coreIMAGE). These resources are:

**DMATtoHOSTNCh:** U16 array indicator. This indicator has the information about the number of DMA channels implemented (the array size) and channels allocated in the different DMAs. The values of the different array elements are the number of channels. A group is defined as the set of channels included in one DMA.

**DMATtoHOSTFrameType:** U8 array indicator. This array has the same dimension size that DMATtoHOSTNCh. Every element in the array contains the data format used for the DMA data. The possible values for FlexRIO are: Format A and Format B.

**Table 12: Possible values for an element in DMATtoHOSTFrameType array**

DMATtoHOSTFrameType [index]	Info
0	Format A
1	Format B

**DMATtoHOSTSampleSize:** U8 array indicator. This array has the same dimension size that DMATtoHOSTNCh. Every element in the array contains the number of bytes used per sample. In a specific design all the channels included in DMA must have the same value of this parameter for all channels. Table 8 presents the valid values.

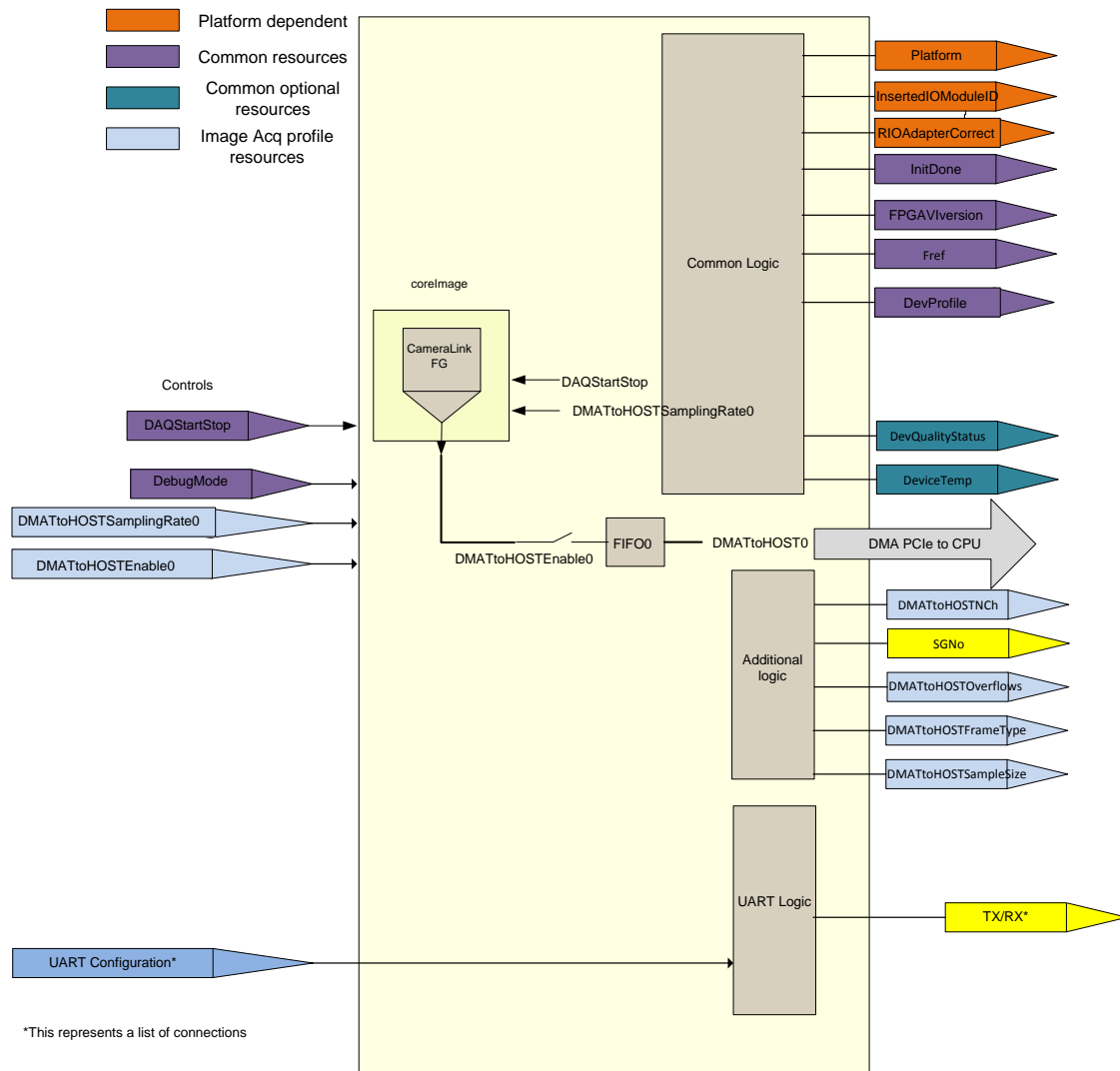
**Table 13: Valid sample size in bytes**

DMATtoHOSTSampleSize [index]	Info
0	Not valid
1	1 sample is one byte
2	1 sample is 2 bytes
4	1 sample is 4 bytes
8	1 sample is 8 bytes

**DMATtoHOST<n>:** This element is a target to host FIFO FPGA memory. This means that this memory is inside the FPGA. This memory is different of the DRAM memory externally located to the FPGA. This FIFO is always a 64-bit-wide FIFO connected to a DMA channel to send data to the HOST. The maximum number of FIFO DMAs is 16 for FlexRIO devices). Each DMA channel will send data acquired from a set of channels. We define this as DMA group.

**DMATtoHOSTEnable<n>:** Boolean register control. There are as many DMATtoHOSTEnable controls (DMATtoHOSTEnable [0.. I<sub>max</sub>-1]) as there are DMA groups. The data of the group are acquired if this control is set to true. **DMATtoHOSTEnable** is complex to design because have to be synchronized with the starting of a new frame.

**DMATtoHOSTOverflows:** U16 indicator. Each bit of this indicator will show the status of each device's possible DMAs. The status will be either Correct (0) or Overflow (1).



**Fig. 35: coreIMAGE. Minimum element for implementing data acquisition in a RIO device**

The Cameralink standard defines different communication modes. These are related with the amount of information transmitted in each clock cycle. Additionally, the standard defines a serial line that allows to send/receive command/status information to/from camera. The FPGA must contain the logic supporting this interface because the UART is implemented in the adapter module. The terminals needed to do this are:

**Configuration:** U8 Control. This terminal defines the cameralink mode. The possible values are listed in Table 14

**Table 14: Valid modes**

Configuration	Mode
0	Base

Configuration	Mode
1	Medium
2	Full/Extended/10-tap

**SignalMapping:** U8 control. This terminal allows defining the cameralink data mode. The possible values are shown in Table 15.

Table 15: Valid signal mapping.

SignalMapping	Mode
0	Standard
1	Basler 10 tap
2	Vosskühler 10-tap

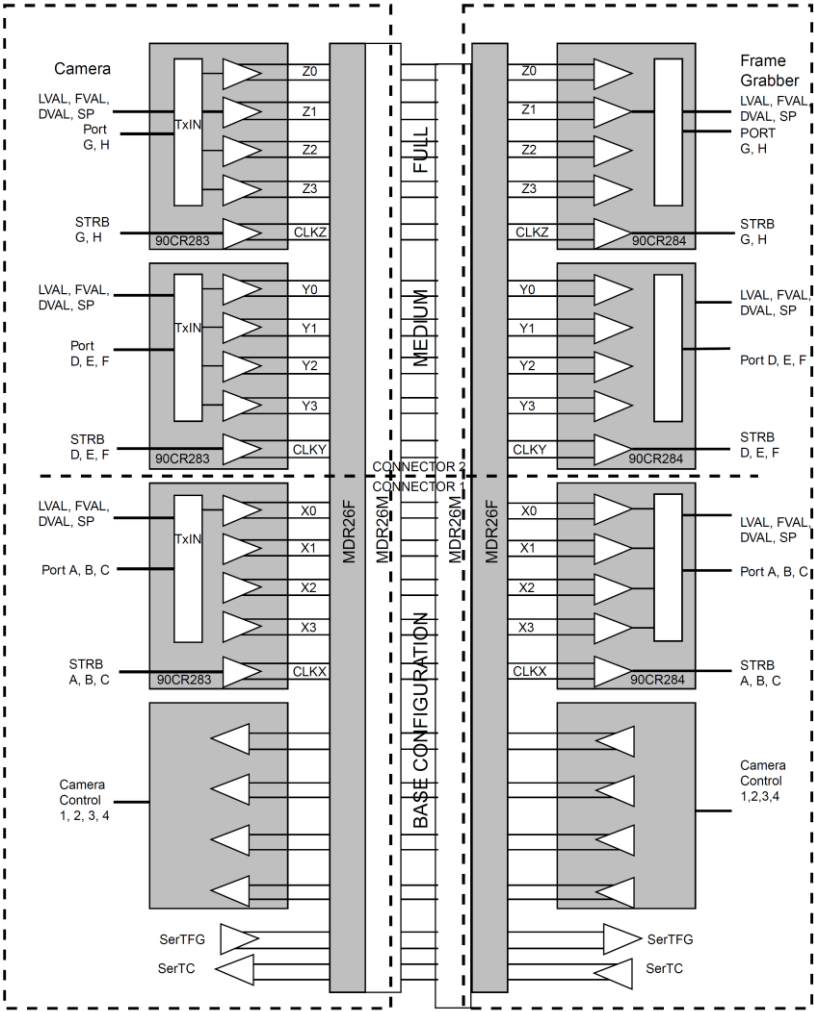


Fig. 36: Cameralink signals.



**LineScan:** Boolean. This configures if the camera is an area scan mode (FALSE) or line mode (TRUE).

**FVALHigh:** Boolean. This terminal sets the polarity of the FVAL signal. TRUE is active high, FALSE active low.

**LVALHigh:** Boolean. This terminal sets the polarity of the LVAL signal. TRUE is active high, FALSE active low.

**DVALHigh:** Boolean. This terminal sets the polarity of the DVAL signal. TRUE is active high, FALSE active low.

**SpareHigh:** Boolean. This terminal sets the polarity of the Spare signal. TRUE is active high, FALSE active low.

**ControlEnable:** Boolean. This terminal activates the signal driving in cameralink.

**uartTransmit:** Boolean control. This terminal activates the transmission of a byte using the serial line.

**uartReceive:** Boolean. This terminal activates the reception of one byte in the serial line.

**uartSetBaudRate:** Boolean. This terminal activates a new configuration of the Baudrate. The baudrate is specified in the uartBaudRate control.

**uartBaudRate:** U8. This control has an enumerated value with the Baudrate to be used in the serial line. The allowed values are: 1, 9,600 baud; 2 = 19,200 baud; 4 = 38,400 baud ; 8 = 57,600 baud ; 16 = 115,200 baud ; 32 = 230,400 baud; 64 = 460,800 baud; 128 = 921,600 baud.

**uartByteMode:** Boolean. This terminal allows configuring the mode for transmitting a collection of bytes. If true the user can send an array of bytes. Check the LabVIEW pattern provided to see the utility of this mode. The byte mode is not currently supported.

**DataBytetoTx:** U8. Byte to transmit.

**uartRxData:** U8. Data received.

**uartTxReady:** Boolean. Terminal indicating the uart is ready to transmit data.

**uartRxReady:** Boolean. Terminal indicating the uart has received a data.

**uartBreakIndicator:** Boolean. This signal indicates that the received byte being output on UART Read Data was received as part of a break condition. A break condition occurs when the RX serial input signal is held low by the camera for longer than the time required to send a full byte. In this case, the data byte received has the value 0 and the UART Break Indicator will be TRUE when the byte is read.

**uartFrammingError:** Boolean. This signal indicates that the received byte being output on UART Read Data did not have a valid stop bit. This means that a transmission error occurred and that the received data may not be reliable.

**uartOverrunError:** Boolean. When this signal rises, it means that the receive buffer has filled and one or more bytes of received data have been lost. This occurs when data is not read out from the UART quickly enough.

## 5.4.2 Data format in the DMA for Image Acquisition profile

### 5.4.2.1 PXIe 7966R / NI1483

The PXIe 7966R / 1483 bundle is oriented for the implementation of image data acquisition systems. The frame grabber is implemented using the FPGA and the NI1483 adapter module because it contains a cameralink interface.

#### 5.4.2.1.1 Format A.

The data in the DMA must be formatted according to the following rules:

- The number of channels N is always 1. The corresponding DMATtoHOSTNCh [i] element has to be 1.
- W: Bytes used per sample. W=1 for instance for NI1483 connected to a grayscale camera. All channels in the DMA use the same W. The valid values for bytes used per samples are 1, 2, 4 or 8. W is specified in the DMATtoHOSTSampleSize array.
- S is the number of samples S in a block. Every block has a length of U64 data with S samples (the number of channels included is defined with N). S must be an integer number multiple of  $N*W/4$ . This value is specified using the DMATtoHOSTBlockNWords array.
- The acquired data must be always encapsulated in 64-bit words of the DMA FIFO.

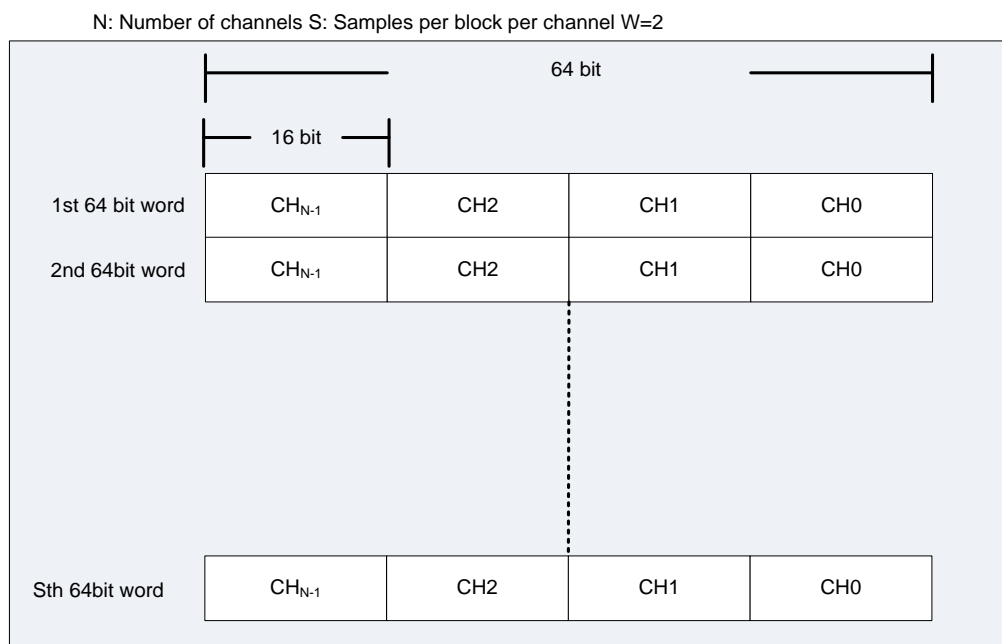


Fig. 37: Data organization in the DMA. Example for N=4



#### Examples. Using one cameralink in NI1483. N=1. W=1 S=1024.

In this example the block is S=1024 U64 words. This means that there are 1024 samples per channel in a Block.  $N*W*S/4=1024$ .



**Warning.** The correct organization of DMA data frame is responsibility of the designer. This must follow the LabVIEW for FPGA [RD5]

#### 5.4.2.1.2 Format B (TDB).

This format is TBD.

### 5.4.3 Summary of resources used for image acquisition profile

Table 16 summarizes the terminals (control and indicators) used by data acquisition profile in FlexRIO platform. The template for the bundle PXIe-7966R/NI1483 (see paragraph 7.2) has been implemented using these terminals.

**Table 16: Summary of the terminals used by the image acquisition profile**

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
Platform	U8	Indicator	This terminal defines the form factor used in the FPGA implementation	Mandatory	0- FlexRIO 1- cRIO 2- R Series	YES
Common Terminals for FlexRIO						
FPGAVIversion	Array U8	Indicator	Contains the VI version, 2 elements. One for MM major version, and the next one mm minor version. MM.mm	Mandatory	For instance 1.1 FPGAVIversion[0]=1 FPGAVIversion[1]=1	YES
InitDone	Boolean	Indicator	This terminal must be set to true when the FPGA is initialized	Mandatory	True=OK False=NOK	N/A
RIOAdapterCorrect	Boolean	Indicator	Boolean indicating if the adapter module is the correct for the application	Mandatory	by NI	NO
InsertedIOModuleID	U32	Indicator	Contains the	Mandatory	Defined by NI	NO

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			Module ID of the corresponding module			
Fref	U32	Indicator	Contains the Reference clock of the FPGA for sampling rate	Mandatory		YES
DevQualityStatus	U8	Indicator	This indicator will show the status of the acquisition	Mandatory		NO
DevTemp	I16	Indicator	This indicator will show the temperature of the FPGA	Mandatory		NO
Devprofile	U8	Indicator	This indicator defines the implementation in the FPGA (DAQ, Image, etc.)	Mandatory		YES
DebugMode	Boolean	Control	If debug is true the FPGA will simulate the acquired data. Otherwise, physical signals are acquired	Mandatory		NO
DAQStartStop	Boolean	Control	This terminal must be set to true to start data acquisition	Mandatory		
<b>Specific Terminals for image acquisition profile</b>						
DMATtoHOSTNCh	Array U16	Indicator	Describes the number of DMAs implemented in the FPGA. The array must be initializes	Mandatory		YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			with the number of channels available in each DMA.			
DMATtoHOSTFrameType	Array U8	Indicator	Describes the frame type used in the DMA frame	Mandatory		YES
DMATtoHOSTSampleSize	Array U8	Indicator	Size in bytes for the channel sample	Mandatory		YES
DMATtoHOSTBlockNWords	Array U8	Indicator	Length of the block	Mandatory		YES
DMATtoHOST<n>	FIFO	DMA to HOST	FIFO memory in the FPGA	Mandatory	n={0 .. 16}	N/A
DMATtoHOSTSamplingRate<n>	U16	Control	Integer number obtained as Sampling rate/Fref	Mandatory	n={0 .. 16}	
DMATtoHOSTEnable<n>	Boolean	Control	Enable or disable write to DMA FIFO	Mandatory	n={0 .. 16}	
DMATtoHOSTOverflows	U16	Indicator	Status of the different DMA FIFO	Mandatory		
SignalMapping	U8	Control	Select the signal mapping for the cameralink interface  0: Standard 1: Basler 10-tap 2: Voskhuler 10 tap	Mandatory	SignalMapping	YES
Configuration	U8	Control	Select cameralink interface type	Mandatory	Configuration	YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			0: Base 1: Medium 2: full			
LineScan	Boolean	Control	Set CL Line Scan	Mandatory	LineScan	YES
FVALHigh	Boolean	Control	Set CL FVAL Active High	Mandatory	FVALHigh	YES
LVALHigh	Boolean	Control	Set CL LVAL Active High	Mandatory	LVALHigh	YES
DVALHigh	Boolean	Control	Set CL DVAL Active High	Mandatory	DVALHigh	YES
SpareHigh	Boolean	Control	Set CL Spare Active High	Mandatory	SpareHigh	YES
ControlEnable	Boolean	Control	CL Control Enable	Mandatory	ControlEnable	YES
uartTransmit	Boolean	Control	Activate to transmit data	Mandatory	uartTransmit	
uartReceive	Boolean	Control	Activate to receive data	Mandatory	uartReceive	
uartSetBaudRate	Boolean	Control	Activate set Baud Rate	Mandatory	True/False	
uartBaudRate	U8	Control	Enumerated value with the baudrate  9.6, 19.2, 38.4, 57.6, 115.2, 230.4, 460.8, or 921.6 kbps	Mandatory		
uartByteMode	Boolean	Control		Mandatory	uartByteMode	
uartTxByte	U8	Control	Byte to be transmitted	Mandatory	DataBytetoTx	
uartRxBytea	U8	Indicator	Data received	Mandatory	uartRxData	
uartTxReady	Boolean	Indicator	Transmitter ready	Mandatory	uartTxReady	
uartRxReady	Boolean	Indicator	Receiver	Mandatory	uartRxReady	

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			Ready			
uartBreakIndicator	Boolean	Indicator	Uart break indicator	Mandatory	uartBreakIndicator	
uartFrammingError	Boolean	Indicator	Frame Error	Mandatory	uartFrammingError	
uartOverrunError	Boolean	Indicator	Overrun Error	Mandatory	uartOverrunError	
<b>Optional Resources for Image Profile</b>						
auxAI<n>	I32	Indicator	Auxiliary internal FPGA variables	Optional	n={0 .. 15}	
auxAO<n>	I32	Control	Auxiliary internal FPGA variables	Optional	n={0 .. 15}	
DO<n>	Boolean	Control	Digital line	Optional	n={0 .. 3}	
DI<n>	Boolean	Control	Digital line	Optional	n={0 .. 3} The sum of digital input plus outputs never be more than 4	
auxDI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 15}	
auxDO<n>	Boolean	Control	Digital line	Optional	n={0 .. 15}	

## 5.5 Analog Signal Data acquisition profile (data to GPU).

The implementation of the profile for acquiring data and moving it to GPU is exactly the same as described previously for the HOST with the difference of the DMA resources used. This section defines the terminal labels for DMA to GPU implementation.



**Warning. The implementation of Data Acquisition profile to send the data to GPU needs to use the specific hardware/software bundle NVIDIA/FlexRIO.**

### 5.5.1 Mandatory resources for data acquisition profile (data to GPU).

Fig. 38 summarizes the different resources needed to implement the data acquisition profile (defined here as coreDAQGPU). This profile sends data to GPU memory. These resources are:

**DMATtoGPUNCH:** U1

6 array indicator. This indicator has the information about the number of DMA channels implemented (the array size) and channels allocated in the different DMAs. The values of the different array elements are the number of channels. A group is defined as the set of channels included in one DMA.

**DMATtoGPUFrameType:** U8 array indicator. This array must have the same dimension and size that DMATtoGPUNCH. Every element in the array contains the data format used for the DMA data. The possible values for FlexRIO are: Format A and Format B.

Table 17: Possible values for an element in DMATtoGPUFrameType array

DMATtoHostFrameType[index]	Info
0	Format A
1	Format B

**DMATtoGPUSampleSize:** U8 array indicator. This array has the same dimension size that DMATtoHOSTNCh. Every element in the array contains the number of bytes used per sample. In a specific design all the channels included in DMA must have the same value of this parameter for all channels. Table 8 presents the valid values.

Table 18: Valid sample size in bytes

DMATtoHOSTSampleSize [index]	Info
0	Not valid
1	1 sample is one byte
2	1 sample is 2 bytes
4	1 sample is 4 bytes
8	1 sample is 8 bytes



**[Example for NVIDIA/PXIE7966R/NI6761]:** DMATtoGPUNCH[2]={1,1} This means that we have one DMA with one channel, and another DMA with another channel. One possible case is acquiring the signal with the first DMA and the FFT with the second one. DMATtoGPUFrameType[2]={0, 0}, this is the same frame type for both DMAs. DMATtoGPUSampleSize[2]={2,8}, the signal



samples have two bytes and the FFT is estimated with 32 bits for real part and 32 bits for imaginary one. Problem: If results are longer than 64 bits packaging will be required. This will complicate the design.

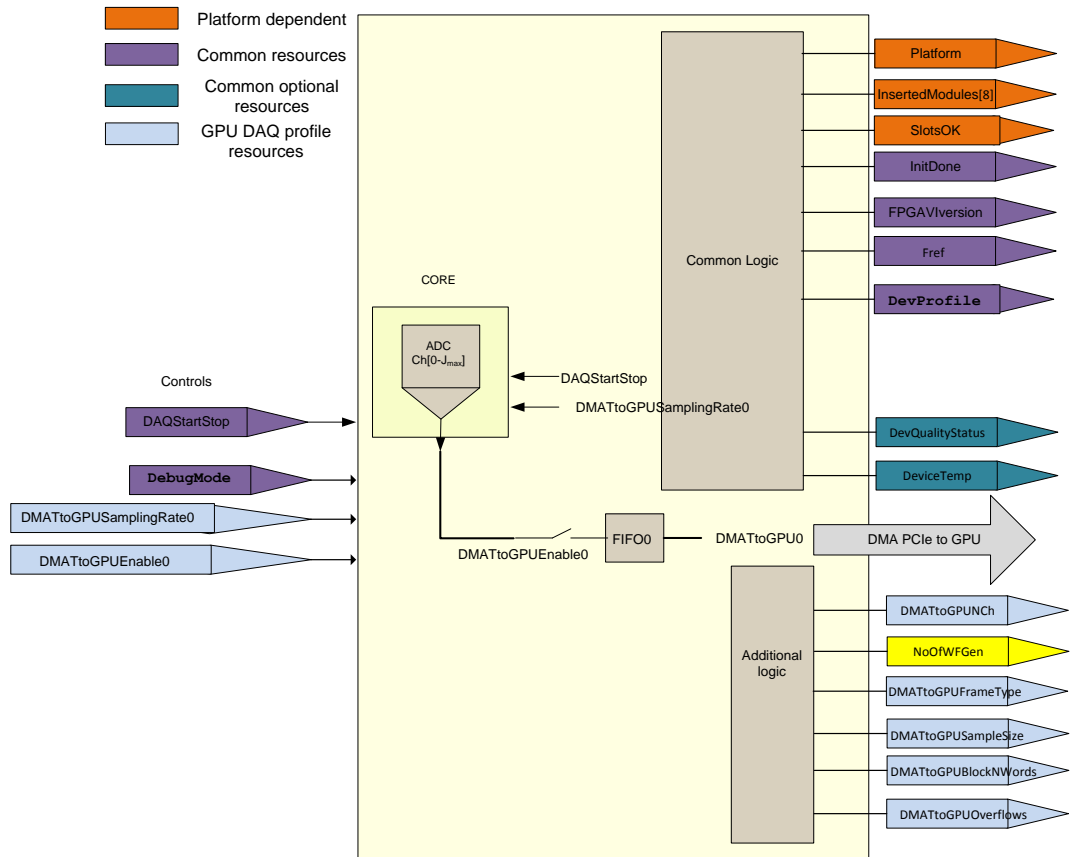
**DMATtoGPUBlockNWords<n>**: U16 array indicator. This array has the same dimension and sizes that the previous ones. Each element contains the length of the block used in the data acquisition. This terminal will inform to the software layer about the frame length. A frame is a set of samples of the different channels. The length of the block is defined as S.

**DMATtoGPU<n>**: This element is a target to host FIFO FPGA memory (for LabVIEW for FPGA there is no difference between host memory and GPU memory). This means that this memory is inside the FPGA. This memory is different of the DRAM memory externally located close to the FPGA (this depends on the FlexRIO used). This FIFO is always a 64-bit-wide FIFO connected to a DMA channel to send data to the GPU. The maximum number of FIFO DMAs is 16 for FlexRIO devices. Each DMA channel will send data acquired from a set of channels. We define and call this a DMA group.

**DMATtoGPUSamplingRate<n>**: Control register. U16. There must be as many “SamplingRate” controls as DMAs used to pass acquired data to the CPU. The data acquired will be packaged into groups of channels and then flow through each DMA. See point 5.3.2 to understand how information is formatted. The label used must be enumerated from 0 to  $I_{\max}-1$ .  $I_{\max}$  is the maximum number of DMA channels available for the FlexRIO device (16). If the design includes more than one DMA, there will be a set of controls that we can define as  $\text{SamplingRate}[0..I_{\max}-1]$ . These terminals control the sampling frequency from DMA group 0 to I-1.

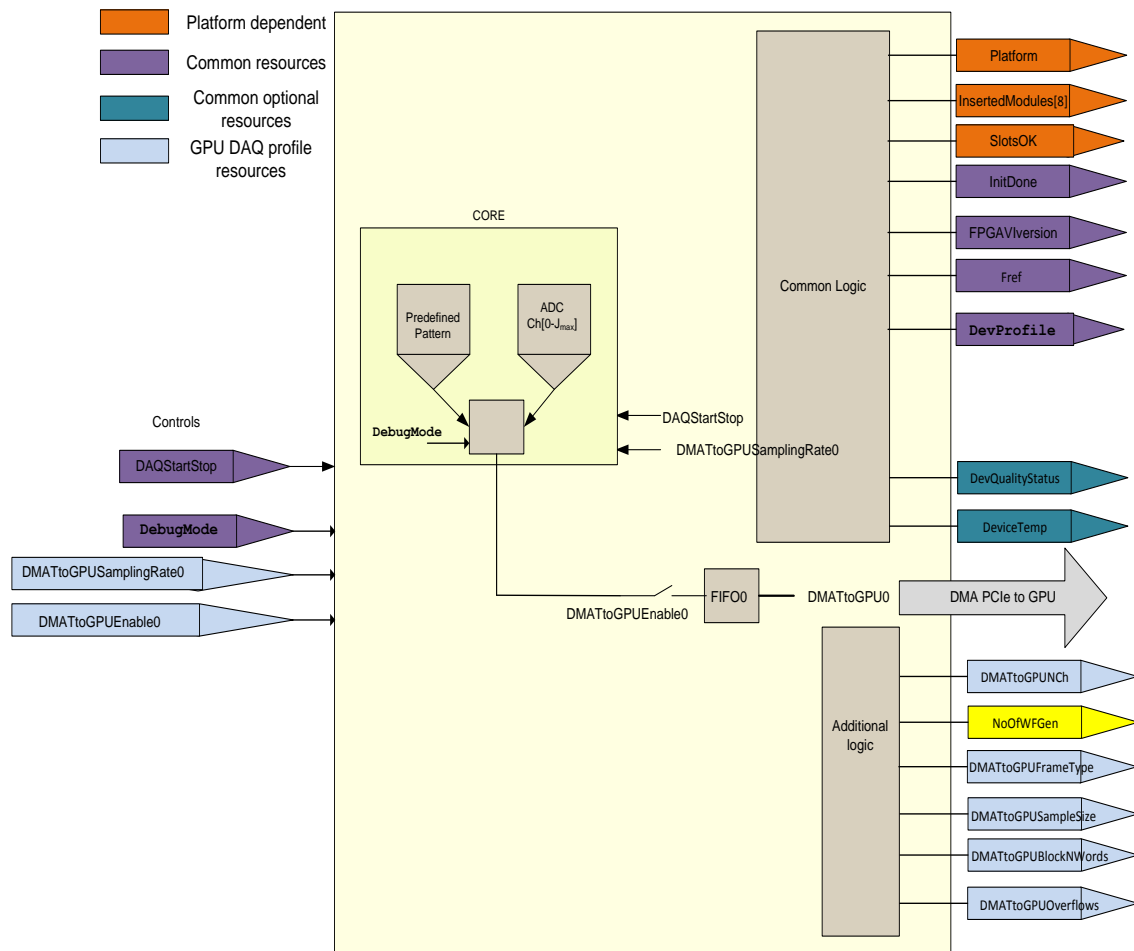
**DMATtoGPUEnable<n>**: Boolean register control. There are as many GroupEnable controls ( $\text{GroupEnable}[0..I_{\max}-1]$ ) as there are DMA groups. The data of the group are acquired if this control is set to true.

**DMATtoGPUOverflows**: U16 indicator. Each bit of this indicator will show the status of each of the device’s possible DMAs. The status will be either Correct (0) or Overflow (1).



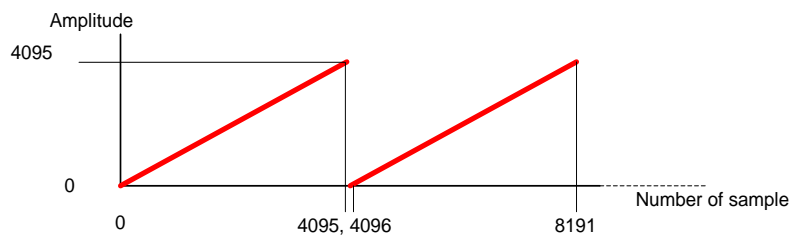
**Fig. 38: coreDAQGPU. Minimum element for implementing data acquisition in a RIO device**

The coreDAQGPU profile can be improved for supporting the debugging functionalities. Fig. 39 shows the idea of debugging the DAQ system using predefined tests patterns.



**Fig. 39: Adding Ramp Pattern Simulation to the design.**

For instance a simple hardware can be added to generate a periodic ramp signal in the FPGA to simulate the acquisition and allow the user to test the design. The pattern could follow, for instance, a ramp shape with a maximum value equals to the number of elements in a block (Fig. 40).



**Fig. 40: Ramp pattern generated by the FPGA.**

To change the operation of coreDAQGPU to simulation mode, we use a control register referred to as DebugMode.

## 5.6 Image acquisition profile (data to GPU)

The implementation of the profile for acquiring images and moving them to GPU is exactly the same as described previously for the HOST with the difference of the DMA resources used. This section defines the terminal labels for DMA to GPU implementation.



**Warning.** The implementation of Data Acquisition profile to send the data to GPU needs to use the specific hardware/software bundle NVIDIA/FlexRIO.

### 5.6.1 Mandatory resources for Image acquisition profile

Fig. 41 summarizes the different resources needed to implement the image acquisition profile (defined here as coreIMAGEGPU). These resources are:

**DMATtoGPUNCH:** U16 array indicator. This indicator has the information about the number of DMA channels implemented (the array size) and channels allocated in the different DMAs. The values of the different array elements are the number of channels. A group is defined as the set of channels included in one DMA.

**DMATtoGPUFrameType:** U8 array indicator. This array has the same dimension size that DMATtoGPUNCH. Every element in the array contains the data format used for the DMA data. The possible values for FlexRIO are: Format A and Format B.

Table 19: Possible values for an element in DMATtoHOSTFrameType array

DMATtoHOSTFrameType [index]	Info
0	Format A
1	Format B

**DMATtoGPUSampleSize:** U8 array indicator. This array has the same dimension size that DMATtoGPUNCH. Each element of the array contains the number of bytes used per sample. In a specific design all the channels included in DMA must have the same value on this parameter for all channels. Table 20 presents the valid values.

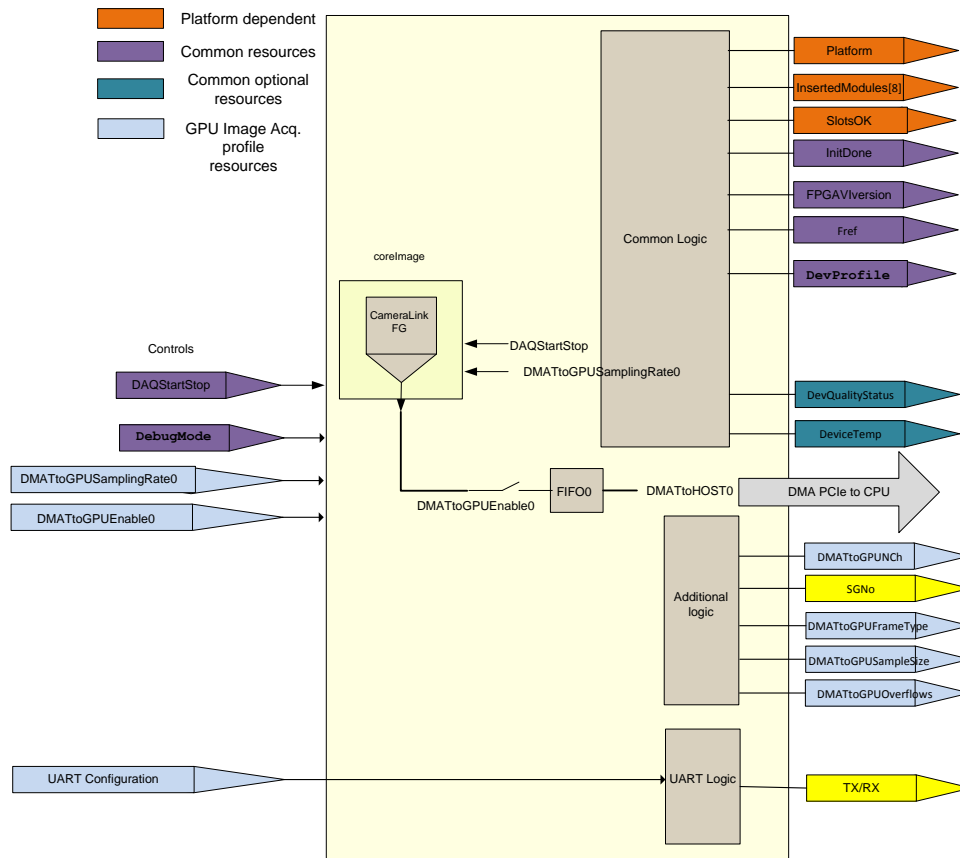
Table 20: Valid sample size in bytes

DMATtoGPUSampleSize [index]	Info
0	Not valid
1	1 sample is one byte
2	1 sample is 2 bytes
4	1 sample is 4 bytes
8	1 sample is 8 bytes

**DMATtoGPU<n>**: This element is a target to host FIFO FPGA memory. This means that this memory is inside the FPGA. This memory is different of the DRAM memory externally located to the FPGA. This FIFO is always a 64-bit-wide FIFO connected to a DMA channel to send data to the HOST. The maximum number of FIFO DMAs is 16 for FlexRIO devices). Each DMA channel will send data acquired from a set of channels. We define this as DMA group.

**DMATtoGPUEnable<n>**: Boolean register control. There are as many DMATtoGPUEnable controls (DMATtoGPUEnable [0..  $I_{max}-1$ ]) as there are DMA groups. The data of the group are acquired if this control is set to true. The enable action of the **DMATtoGPUEnable** is complex to design because have to be synchronized with the starting of a new frame.

**DMATtoGPUOverflows**: U16 indicator. Each bit of this indicator will show the status of each of the device's possible DMAs. The status will be either Correct (0) or Overflow (1).



**Fig. 41: coreIMAGEGPU. Minimum element for implementing data acquisition in a RIO device**

The remaining elements for completing this template are the same explained for coreIMAGE profile.

## 6 DESIGN RULES FOR cRIO

The FPGA VI must contain a set of terminals that are mandatory independently of the application. These terminals are presented in different colours in order to identify clearly the different functionalities. Some terminals need a default value because they will be read when the FPGA is not running yet. Additionally to this section, the FPGA code has to meet some additional rules described later in this document.



**[Important]:** The cRIO design rules have been defined considering that the main objectives are: a) the implementation of analog and digital I/O oriented applications for sampling rates below 250kS/s. b) The implementation of analog input waveform oriented applications



**Warning. LabVIEW terminal labels are case sensitive.**

### 6.1 Platform identification

**Platform:** Enum U8 Indicator register. This element is the register used to identify the hardware platform in use. The values for this terminal (see Table 21) are read by the software driver when the bitfile has been downloaded to the FPGA but is not running.

Table 21: Values for Platform indicator

Platform	Value
FlexRIO	0
cRIO	1
R-Series	2

### 6.2 Mandatory resources for a cRIO design

Fig. 42 shows the basic resources to be added in a LabVIEW for FPGA design for cRIO. The meaning and functionality of the different terminals are explained below.

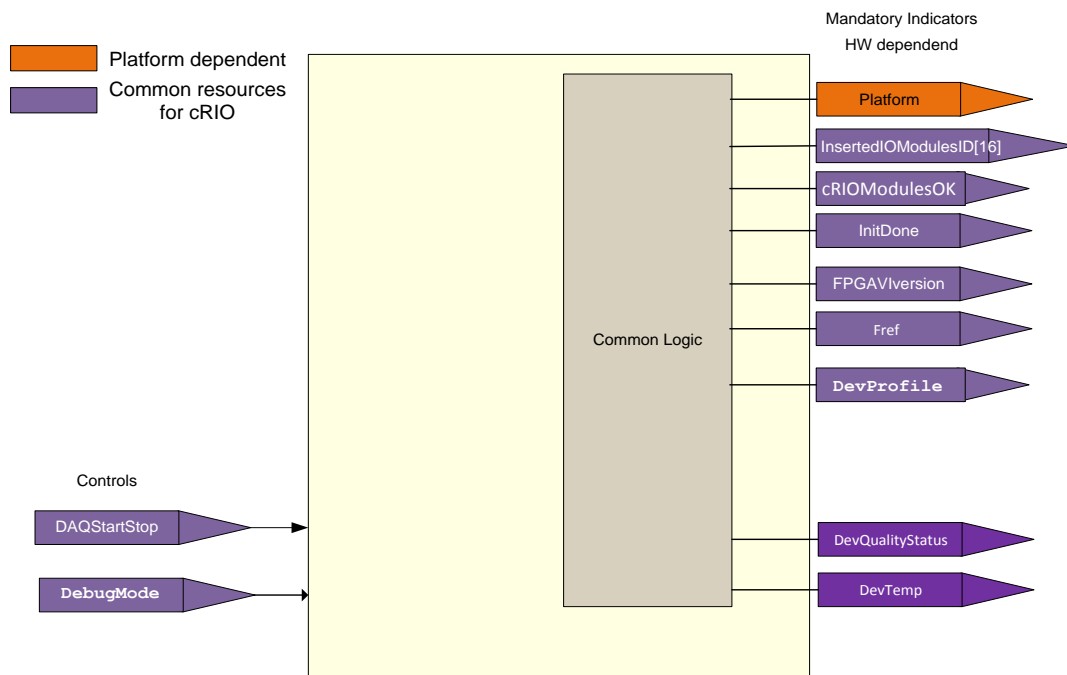


Fig. 42: Common terminals in the VI for cRIO

**FPGAVersion:** U8 array indicator. This indicator contains the version of the VI and is checked by the software driver. The array only uses two elements. The first one includes the major version “MM”, and the second one the minor version, “mm”.

**InitDone:** Boolean register indicator. This indicator signals that the FPGA and modules are correctly initialised. “Zero” means that the FGPA is not ready, and “One” means that the FPGA is ready. The designer should define when the FPGA and the I/O elements are ready to work checking the information provided by the manufacturer. The FPGA designer has to follow the specific steps defined for the I/O modules to execute the initialization.

Table 22: Values for Boolean Initdone indicator

Initdone	Value
Correct	True
Incorrect	False

**InsertedIOModulesID[16]:** Indicator Array of U16. Each position contains the Module ID as defined by National Instruments [RD6] and shown in Table 23. The size of this array should match the size of the chassis used.

Table 23: Module ID for cRIO

Module	ExpectedIOModuleID	I/O resources
--------	--------------------	---------------

Module	ExpectedIOModuleID	I/O resources
NO Module	0x0000	No module installed
NI9205	0x712a	16/32 analog inputs
NI9264	0x 745C	16 analog outputs
NI9401	0x7130	TTL 8 I/O
NI9425	0x712F	24 V, Sinking Digital Input, 32 Ch. Module
NI9426	0x736A	32-Channel, 24 V, 7 $\mu$ s Sourcing Digital Input Module
NI9476	0x7133	24 V, Sourcing Digital Output, 32 Ch. Module
NI9477	0x71CB	60 V, Sinking Digital Output, 32 Ch. Module

**cRIOModulesOK:** Boolean. The expected modules and the modules installed match.

**Fref:** U32 indicator. The indicator will contain the reference clock used for the sampling rate acquisition.

**DevQualityStatus:** U8 indicator. This indicator informs the software driver about the possible errors in the signal conditioning or other possible situations.

**DevTemp:** I16 indicator. This indicator contains the temperature value of the cRIO FPGA.

**DevProfile:** U8 indicator. This indicator is used to determine the kind of application implemented in the FPGA. The value specified here is very important because it defines the resources that mandatory will be searched and the optional resources used. The meaning of DevProfile is different in the different platforms, if DevProfile=0 the implementation contains a design for analog waveform oriented data acquisition, then the resources defined for that profile are mandatory. For this profile waveform output generation, digital and analog point by point I/O are optional. If DevProfile=1 the profile Point By Point data acquisition is implemented. Table 24 and Table 25 summarize the mandatory and optional resources for the profiles. In the case of Point by Point acquisition at least one of the optional elements must be implemented.

**Table 24: Values for DevProfile indicator**

DevProfile	Info
0	Data acquisition
1	Point by Point acquisition (PBP)

**Table 25: Resources for data acquisition profile (cRIO)**

Resources	Info
-----------	------



Resources	Info
Common	Mandatory
Data acquisition	Mandatory
Analog Input	Optional
Analog Output	Optional
Aux Analog Input	Optional
Aux Analog Output	Optional
Digital Output	Optional
Aux Digital Output	Optional
Digital Input	Optional
Aux Digital Input	Optional
DDS Waveform Generation	Optional

Table 26: Resources for point by point (PBP) acquisition profile (cRIO)

Resources	Info
Common	Mandatory
Data acquisition	forbidden
Analog Input	Optional
Analog Output	Optional
Aux Analog Input	Optional
Aux Analog Output	Optional
Digital Output	Optional
Aux Digital Output	Optional
Digital Input	Optional
Aux Digital Input	Optional
DDS Waveform Generation	Optional

**DAQStartStop:** Boolean Control register. This element is the register used to start and stop the data acquisition/generation in the RIO device. This terminal will start data acquisition/generation process in all the FPGA resources.

**DebugMode:** Boolean Control register. This element is the register used to simulate the data acquired by the device. The behaviour of the simulation mode is defined by the developer.

## 6.3 Analog Signal Data acquisition profile (DMA-based)

### 6.3.1 Mandatory resources for data acquisition profile

**DMATtoHOSTNCh:** U16 array indicator. This indicator has the information about the number of DMA channels implemented and channels allocated inside the different DMAs. The values of the different array elements are the number of channels. A group is defined as the set of channels included in one DMA (see Fig. 43).

**DMATtoHOSTFrameType:** U8 array indicator. This array has the same dimension size that DMATtoHOSTNCh. Every element in the array contains the data format used for the DMA data. The possible values for cRIO are described in Table 27.

Table 27: Possible values for an element in DMATtoHOSTFrameType array

DMATtoHOSTFrameType [index]	Info
0	Format A
1	Format B

**DMATtoHOSTSampleSize:** U8 array indicator. This array has the same dimension size that DMATtoHOSTNCh. Every element in the array contains the number of bytes used per sample. In a specific design all the channels included in DMA (DMA group) must have the same value of this parameter for all channels. The valid values are these:

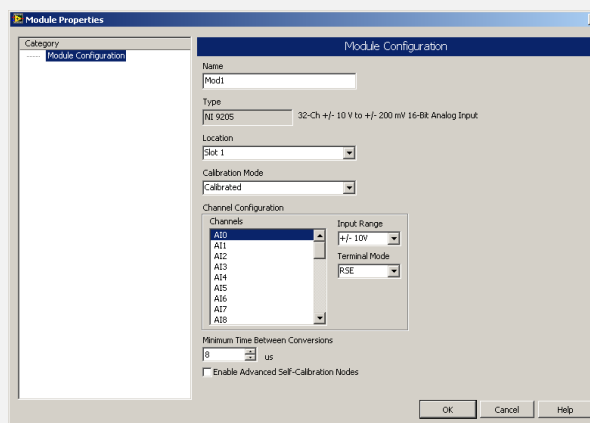
Table 28: Valid sample size in bytes

DMATtoHOSTSampleSize[n]	Info
0	Not valid
1	1 sample is one byte
2	1 sample is 2 bytes
4	1 sample is 4 bytes
8	1 sample is 8 bytes



**[Example for NI9159/NI9205]:**  $\text{DMATtoHOSTNCh}[1] = \{8\}$  Eight channels for data acquisition.  $\text{DMATtoHOSTFrameType}[1]=\{0\}$ ;

$\text{DMATtoHOSTSampleSize}[1] = \{4\}$ . The number of samples per channels depends on the configuration of the cRIO module in the LabVIEW Projects. Raw configuration provides 2 bytes (I16) and calibrated provides 4 bytes because NI9205 represent the data in fixed point format with 21 bits, 5 for the integer part and 16 for the decimal one. These 21 bits are allocated in 4 bytes in the DMA as an I32 data.



**DMATtoHOSTBlockNWords<n>**: U16 array indicator. This array has the same dimension and sizes that the previous ones. Each element contains the length of the block used in the data acquisition. This terminal will inform to the software layer about the frame length. A frame is a set of samples of the different channels. The length of the block is defined as S.

**DMATtoHOST<n>**: This element is a target to host FIFO FPGA memory. This means that this memory is inside the FPGA implemented with the embedded RAM in the Virtex-5 LX110. This memory is used as a FIFO and it is always a 64-bit-wide FIFO connected to a DMA channel to send data to the HOST. The maximum number of FIFO DMAs is 3 for cRIO devices. If you have more than one DMA channel there will be as many DMATtoHOST elements as DMAs up to 3. The identification has to be correlative. Each DMA channel will send data acquired from a group of channels. Every DMA is a DMA group.

**DMATtoHOSTSamplingRate<n>**: Control register. U16. There must be as many “DMATtoHOSTSamplingRate” controls as DMAs used to pass acquired data to the CPU. The data acquired will be packaged into groups of channels and then sent through each DMA. The label used must be enumerated from 0 to  $I_{\max}-1$ .  $I_{\max}$  is the maximum number of DMA channels available for the cRIO device (3). If the design includes more than one DMA, there will be a set of controls that we can define as  $\text{DMATtoHOSTSamplingRate0}[0..I_{\max}-1]$ . These terminals control the sampling frequency from DMA group 0 to I-1.

**DMATtoHOSTEnable<n>**: Boolean register control. There are as many DMATtoHOSTEnable controls ( $\text{DMATtoHOSTEnable}[0..I_{\max}-1]$ ) as DMA groups. The data of the group are acquired if this control is set to true.

**DMATtoHOSTOverflows:** U16 indicator. Each bit of this indicator will show the status of each of the device's possible DMAs. The status will be either Correct (0) or Overflow (1).

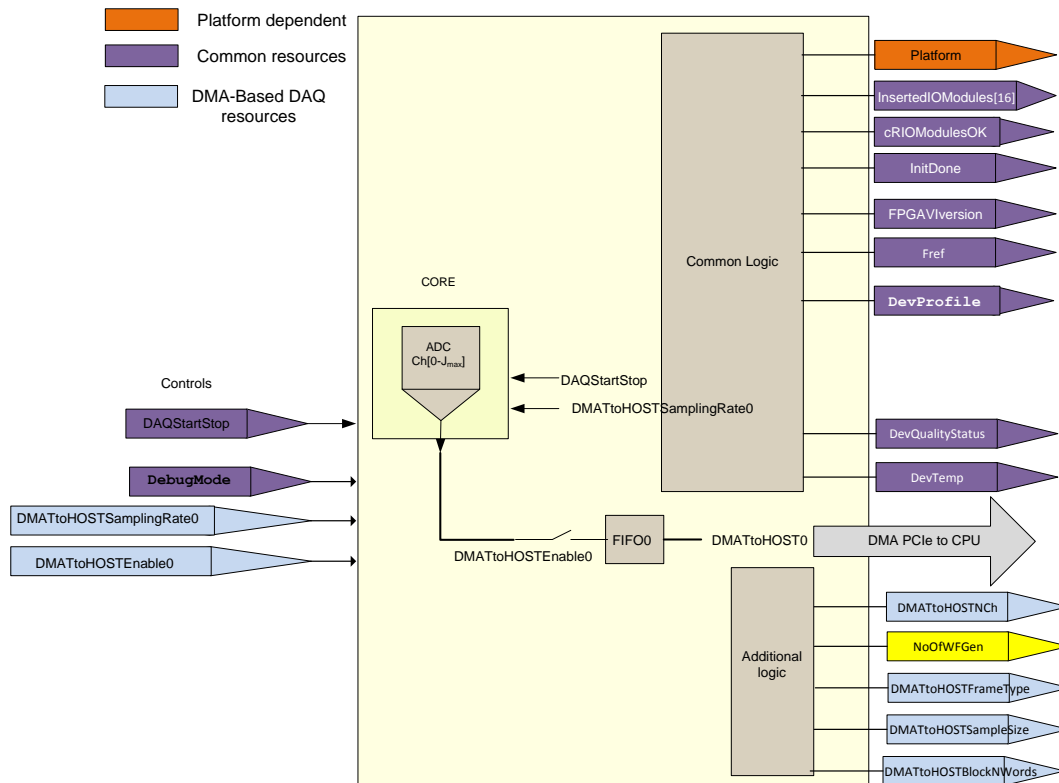


Fig. 43: coreDAQ. Minimum element for implementing data acquisition in a cRIO device

### 6.3.2 Data format in the DMA for Data acquisition profile.

The data acquisition profile is oriented for the acquisition of analog input channels and supports different formats in the data stream sent to the HOST using the DMA.

#### 6.3.2.1 NI9159/NI9205

Every NI9205 provides 16/32 analog inputs. The NI9205 can be configured in raw format or in the calibrated format. One channel sample uses an I16 in raw mode and 21 bits in calibrated one. This point should be considered when defining the bytes used for sample.

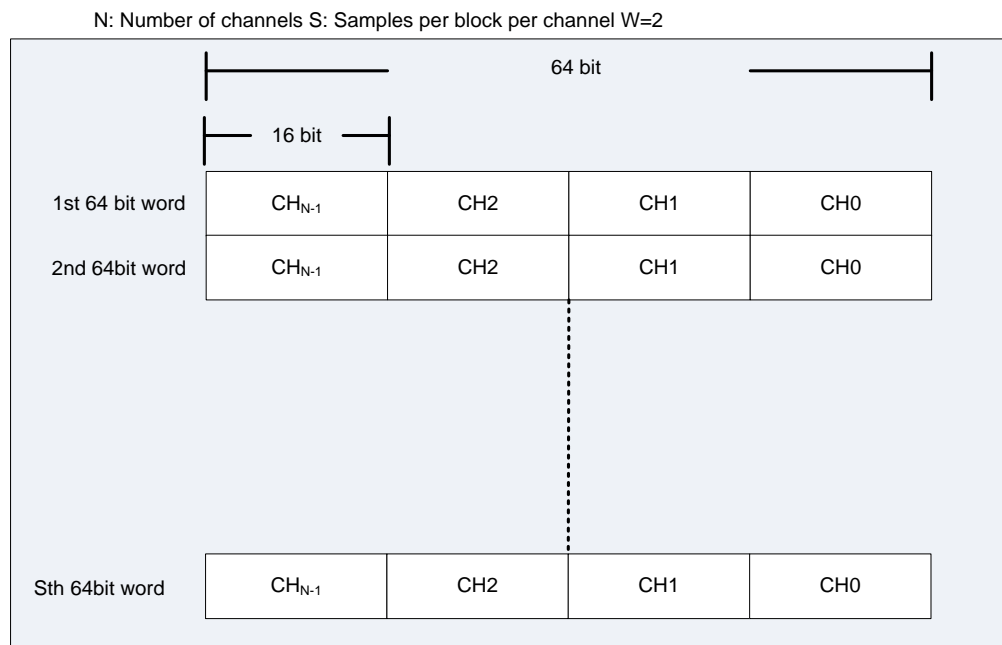


**Warning. NI9205 is the only module with ADC functionality.**

#### 6.3.2.2 Format A: DAQ samples

The data in the DMA must be formatted according to the following rules:

- The number of channels  $N$  is variable between 1 and 256.  $N$  is configured in the FPGA for every DMA using the corresponding  $\text{DMATtoHOSTNCh}[i]$  element.
- $W$ : Bytes used per sample.  $W=2$  for instance for NI9205 in raw format or  $W=4$  in calibrated one. All channels in the DMA use the same  $W$ . The valid values for bytes used per samples are 1, 2, 4 or 8.  $W$  is specified in the  $\text{DMATtoHOSTSampleSize}$  array.
- $S$  is the number of samples  $S$  in a block. Every block has a length of  $U64$  data with  $S$  samples (the number of channels included is defined with  $N$ ).  $S$  must be an integer number multiple of  $N \cdot W / 4$ . This value is specified using the  $\text{DMATtoHOSTBlockNWords}$  array.
- The acquired data must be always encapsulated in 64-bit words of the DMA FIFO (see an example in Fig. 44 and Fig. 45).



**Fig. 44: Data organization in the DMA. Example for  $N=4$**

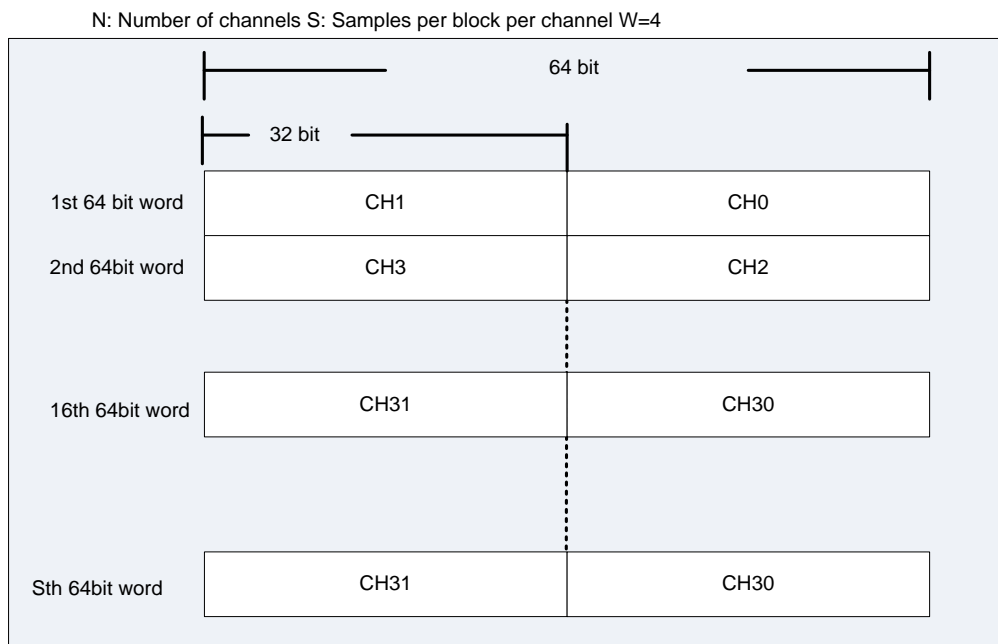


Fig. 45: Data organization in the DMA. Example for N=32

### 6.3.2.3 *Format B: (TBD)*

## 6.3.3 Optional resources

The optional resources checked in this profile are listed below (see Fig. 46).

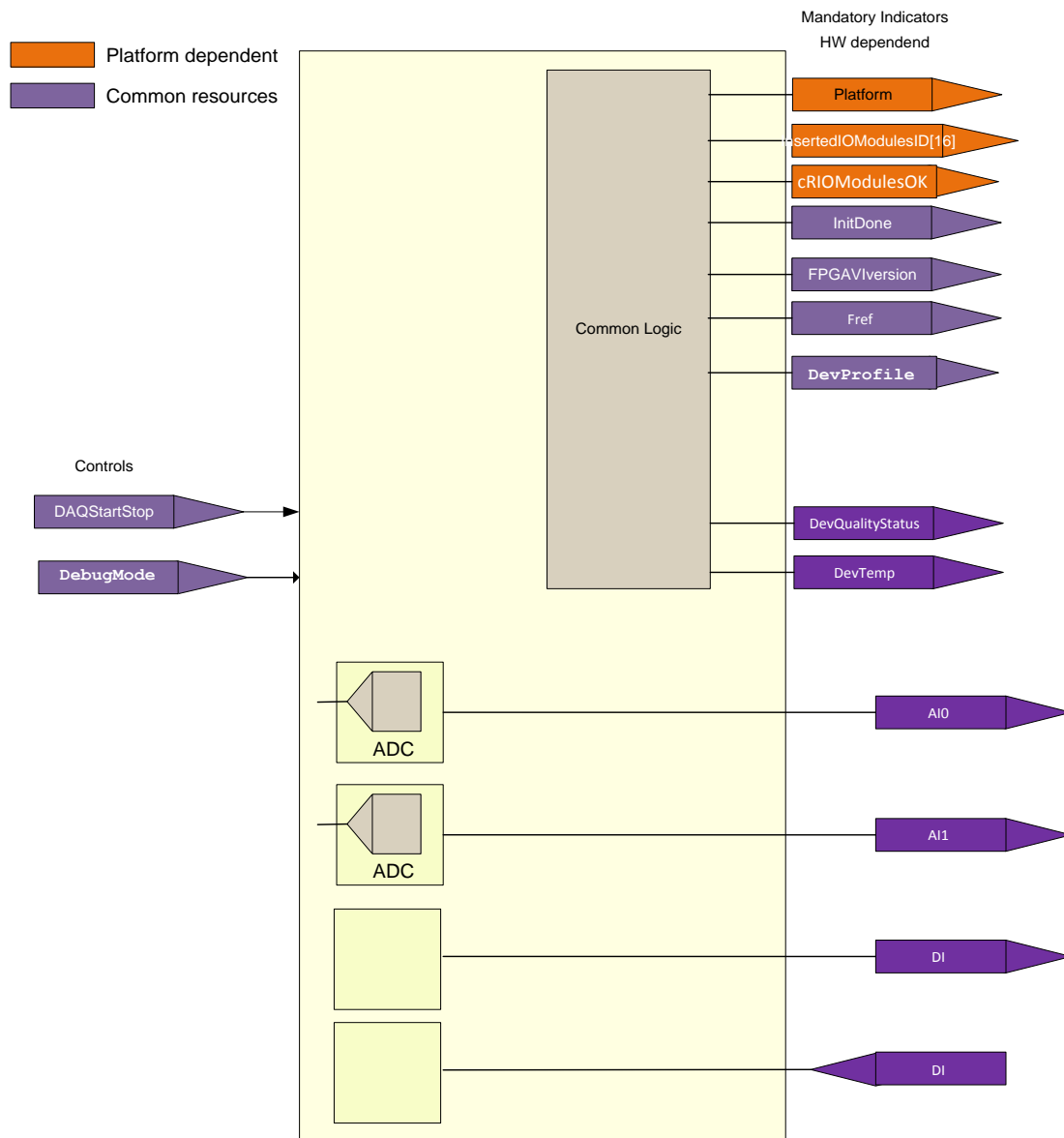


Fig. 46 coreDAQ. DAQ resources

### 6.3.3.1 Analog inputs

**AI<x>:** I32 indicator. The FPGA LabVIEW programmer can add read-only registers (indicators) with the last sample acquired from an analog input. This indicator will be updated at the sampling rate programmed for the channel. The nomenclature for naming the indicator will be "AI" followed by the number of the channel. The maximum number of AI terminals is  $32 \times 14 = 448$  (14 slots with NI9205) The AI<x> are only used if there are NI9205 modules installed.

### 6.3.3.2 Auxiliary analog inputs

**auxAI<x>:** I32 indicator. The FPGA designer can include indicators identified as auxAI<N> with LabVIEW I32 data type representing any internal variable in the FPGA. For instance, you can

acquire one sample from adapter module analog input channels (I16), operate the data and connect the result to an I32 terminal labelled as auxAI0. The maximum number is 16.

### 6.3.3.3 Analog Output

**AO<x>**: I32 indicator. These terminals will be connected to the physical I/O nodes available in I/O module. The maximum number of analog outputs is 32.

### 6.3.3.4 Auxiliary analog output

**auxAO<x>**: I32 control. The FPGA designer can include controls identified as auxAO<x> with LabVIEW I32 data type representing any internal variable in the FPGA. The maximum number of auxAO is 16.

### 6.3.3.5 Digital input/output

**DO<n>**: Boolean control. The FPGA designer can include controls identified as DO<n>. These controls will be connected to physical digital outputs in an I/O module. The maximum number of DO is 96.

**DI<n>**: Boolean indicator. The FPGA designer can include indicators identified as DI<n>. The maximum number is 96.

**auxDO<n>**: Boolean control. The FPGA designer can include controls identified as DO<n>. These controls will be connected to internal FPGA signals. The maximum number is 16.

**auxDI<n>**: Boolean indicator. The FPGA designer can include indicators identified as auxDI<n>. These indicators will be connected to internal FPGA signals in a FlexRIO adapter module. The maximum number is 16.

### 6.3.3.6 Signal generator

**SGNo**: U8 indicator. This indicator is initialised with the number of waveform generators included in the design. A null value means no signal generator implemented. The values allowed are from 0 to 16.

In the cRIO device, the user can add an element to implement signal generation using the analog outputs. The templates provide a signal generator implemented with direct digital synthesis (DDS) technique. In this method, the FPGA contains a memory with a predefined pattern. The details of the implementation are explained in the document [RD7]. The terminals available to use this block are described in Table 29.

Table 29: Signal generator terminals

LabVIEW Terminal Name	Type	Functionality	Notes
<b>SGFreq&lt;n&gt;</b>	U32, Control	Frequency of the signal to be generated	<p>The desired frequency (freq) in Hertz and the terminal follow this equation</p> $SGFreq = freq \times \frac{2^{32}}{Freqo} \times SGUpdateRate$ <p>SGFreq is the frequency used</p>



LabVIEW Terminal Name	Type	Functionality	Notes
			in the output generator
<b>SGAmp&lt;n&gt;</b>	U16, Control	Amplitude of the signal to be generated	The value to be written in the terminal must be a value from 0 to 32767.
<b>SGPhase&lt;n&gt;</b>	U32, Control	Phase control for the signal	The terminal contains the phase shift
<b>SGSignalType&lt;n&gt;</b>	U8, Control	Signal type among DC, Sine, Triangular and Square	Enumerated value to select the signal needed
<b>SGUpdateRate&lt;n&gt;</b>	U32, control	Update rate frequency used for signal generation	The analog output is updated using a frequency equals to the SGUpdateRate
<b>SGFref&lt;n&gt;</b>	U32, Indicator	Defines the reference frequency used by the signal generator	Defines the reference frequency used by the signal generator

### 6.3.4 Summary of resources for cRIO DAQ profile

Table 30 summarizes the terminals (control and indicators) used by data acquisition profile in cRIO platform. The template dataacquisitionDMA (see paragraph 7.3) has been implemented using these terminals.

**Table 30: Summary of resources for cRIO DAQ profile**

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
<b>Platform</b>	<b>U8</b>	Indicator	This terminal defines the form factor used in the FPGA implementation	Mandatory	0- FlexRIO 1- cRIO 2- R Series	YES
<b>Mandatory resources for cRIO</b>						
<b>FPGAVIversion</b>	<b>Array U8</b>	Indicator	Contains the VI version, 2 elements. One for MM major	Mandatory	For instance 1.1 FPGAVIversion[0]=1 FPGAVIversion[1]	YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			version, and the next one mm minor version. MM.mm		] = 1	
InitDone	Boolean	Indicator	This terminal must be set to true when the FPGA is initialized	Mandatory	True=OK False=NOK	N/A
InsertedOModulesID	Array U16	Indicator	Numeric array of values indicating the c-Modules IDs detected	Mandatory	Defined by NI	NO
cRIOModulesOK	Boolean	Indicator	I/O Modules correctly detected	Mandatory		NO
Fref	U32	Indicator	Contains the Reference clock of the FPGA for sampling rate	Mandatory		YES
DevQualityStatus	U8	Indicator	This indicator will show the status of the acquisition	Mandatory		NO
DevTemp	I16	Indicator	This indicator will show the temperature of the FPGA	Mandatory		NO
DevProfile	U8	Indicator	This indicator defines the	Mandatory		YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			implementation in the FPGA (DAQ, Image, etc.)			
DebugMode	Boolean	Control	If debug is true the FPGA will simulate the acquired data. Otherwise, physical signals are acquired	Mandatory		NO
DAQStartStop	Boolean	Control	This terminal must be set to true to start data acquisition	Mandatory		
<b>Specific Terminals for cRIO DAQ profile</b>						
DMATtoHOSTNCh	Array U16	Indicator	Describes the number of DMAs implemented in the FPGA. The array must be initialized with the number of channels available in each DMA.	Mandatory	$n = \{0 \dots 2\}$	YES
DMATtoHOSTFrameType	Array U8	Indicator	Describes the frame type used in the DMA frame	Mandatory	$n = \{0 \dots 2\}$	YES
DMATtoHOSTSampleSize	Array U8	Indicator	Size in bytes for the channel sample	Mandatory	$n = \{0 \dots 2\}$	YES
DMATtoHOSTBlockNWords	Array	Indicator	Length of	Mandatory	$n = \{0 \dots 2\}$	YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
	U16		the block			
DMATtoHOST<n>	FIFO	DMA target to HOST	FIFO memory in the FPGA	Mandatory	n={0 .. 2}	N/A
DMATtoHOSTSamplingRate<n>	U16	Control	Integer number obtained as Sampling rate/Fref	Mandatory	n={0 .. 2}	
DMATtoHOSTEnable<n>	Boolean	Control	Enable or disable write to DMA FIFO	Mandatory	n={0 .. 2}	
DMATtoHOSTOverflows	U16	Indicator	Status of the different DMA FIFO	Mandatory		
<b>Optional Resources for this profile</b>						
AI<n>	I32	Indicator	Digital sample	Optional	n={0 .. 255}	
auxAI<n>	I32	Indicator	Auxiliary internal FPGA variables	Optional	n={0 .. 255}	
AO<n>	I32	Indicator	Digital Sample	Optional	n={0 .. 255}	
auxAO<n>	I32	Control	Auxiliary internal FPGA variables	Optional	n={0 .. 255}	
AOEnable<n>	Boolean	Control	Enable or Disable analog output		n={0 .. 255}	
DO<n>	Boolean	Control	Digital line	Optional	n={0 .. 255}	
auxDO<n>	Boolean	Control	Digital line	Optional	n={0 .. 255}	
DI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 255}	
auxDI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 255}	

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
SGNo	U8	Control	Number of waveform generators	Optional	n={0 .. 255}	YES
SGSignalType<n>	U8	Control	Signal shape to be generated	Optional	User selectable	YES
SGFreq<n>	U32	Control	DSS accumulator increment	Optional	n={0 .. 255}	YES
SGAmp<n>	U16	Control		Optional	n={0 .. 255}	YES
SGPhas<n>	U32	Control	Phase control	Optional	n={0 .. 255}	YES
SGUpdateRate<n>	U32	Control	Update rate	Optional	n={0 .. 255}	
SGFref<n>	U32	Indicator	Reference frequency	Optional	n={1..255}	

## 6.4 Point by Point acquisition profile

This profile is oriented to the implementation of analog and digital I/O operations.

### 6.4.1 Mandatory resources for point by point I/O profile

**SamplingRate<n>**: U16 control. This terminal allows to program the sampling rate. The sampling rate value in S/s is the Fref value divided by SamplingRate<n>. This sampling rate is used for controlling the data acquisition/generation of the different I/O elements in the cRIO chassis.

### 6.4.2 Optional resources

The following resources are optional and can appear or not in the design.

#### 6.4.2.1 Analog inputs

**AI<x>**: I32 indicator. The FPGA LabVIEW programmer can add read-only registers (indicators) with the last sample acquired from an analog input. This indicator will be updated at the sampling rate programmed for the channel. The nomenclature for naming the indicator will be "AI" followed by the number of the channel. The maximum number of AI terminals is 32\*14=384 (14 slots with NI9205). The AI<x> are only used if there are NI9205 modules installed.

#### 6.4.2.2 Auxiliary analog inputs

**auxAI<x>**: I32 indicator. The FPGA designer can include indicators identified as auxAI<N> with LabVIEW I32 data type representing any internal variable in the FPGA. For instance, you can

acquire one sample from adapter module analog input channels (I16), operate the data and connect the result to an I32 terminal labelled as auxAI0. Maximum number of auxAI is 16.

#### 6.4.2.3 Analog Output

**AO<x>**: I32 control. These terminals will be connected to the physical I/O nodes available in I/O module. The maximum number of AO terminals is  $16 \times 14 = 224$  (14 slots with NI9264).

#### 6.4.2.4 Auxiliary analog output

**auxAO<x>**: I32 control. The FPGA designer can include controls identified as auxAO<x> with LabVIEW I32 data type representing any internal variable in the FPGA. The maximum number of auxAO is 256.

#### 6.4.2.5 Digital input/output

**DO<n>**: Boolean control. The FPGA designer can include controls identified as DO<n>. These controls will be connected to physical digital outputs in an I/O module. The maximum number of DO is 256.

**DI<n>**: Boolean indicator. The FPGA designer can include indicators identified as DI<n>. The maximum number is 256.

**auxDO<n>**: Boolean control. The FPGA designer can include controls identified as DO<n>. These controls will be connected to internal FPGA signals. The maximum number is 256.

**auxDI<n>**: Boolean indicator. The FPGA designer can include indicators identified as auxDI<n>. These indicators will be connected to internal FPGA signals in a cRIO adapter module. The maximum number is 256.

#### 6.4.2.6 Signal Generator

See signal generator description in data acquisition profile.

### 6.4.3 Summary of resources for cRIO Point by Point profile

Table 31 summarizes the terminals (control and indicators) used by data acquisition profile in cRIO platform. The template pointbypoint (see paragraph 7.3) has been implemented using these terminals.

Table 31: Summary of resources for PBP Profile

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
Platform	U8	Indicator	This terminal defines the form factor used in the FPGA implementation	Mandatory	0- FlexRIO 1- cRIO 2- R Series	YES
Common Terminals for cRIO						
FPGAVIversion	Array U8	Indicator	Contains the VI version, 2 elements. One for MM major version, and the next	Mandatory	For instance 1.1 FPGAVIversion[0]=1	YES

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
			one mm minor version. MM.mm		FPGAVersion[1]=1	
InitDone	Boolean	Indicator	This terminal must be set to true when the FPGA is initialized	Mandatory	True=OK False=NOK	N/A
InsertedOModulesID	Array U16	Indicator	Numeric array of values indicating the c-Modules IDs detected	Mandatory	Defined by NI	NO
cRIOModulesOK	Boolean	Indicator	I/O Modules correctly detected	Mandatory		NO
Fref	U32	Indicator	Contains the Reference clock of the FPGA for sampling rate	Mandatory		YES
DevQualityStatus	U8	Indicator	This indicator will show the status of the acquisition	Mandatory		NO
DevTemp	I16	Indicator	This indicator will show the temperature of the FPGA	Mandatory		NO
Devprofile	U8	Indicator	This indicator defines the implementation in the FPGA (DAQ, Image, etc.)	Mandatory		YES
DebugMode	Boolean	Control	If debug is true the FPGA will simulate the acquired data. Otherwise, physical signals are acquired	Mandatory		NO
DAQStartStop	Boolean	Control	This terminal must be set to true to start data acquisition	Mandatory		
<b>Specific Terminals for Point by Point acquisition profile</b>						
SamplingRate<n>	U16	Control	Integer number obtained as Sampling rate/Fref	Mandatory	n = {0 .. 2}	
Optional Resources for Point by Point profile						

Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
AI<n>	I32	Indicator	Digital sample	Optional	n={0 .. 255}	
auxAI<n>	I32	Indicator	Auxiliary internal FPGA variables	Optional	n={0 .. 255}	
AO<n>	I32	Indicator	Digital Sample	Optional	n={0 .. 255}	
auxAO<n>	I32	Control	Auxiliary internal FPGA variables	Optional	n={0 .. 255}	
AOEnable<n>	Boolean	Control	Enable or Disable analog output		n={0 .. 255}	
DO<n>	Boolean	Control	Digital line	Optional	n={0 .. 255}	
auxDO<n>	Boolean	Control	Digital line	Optional	n={0 .. 255}	
DI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 255}	
auxDI<n>	Boolean	Indicator	Digital line	Optional	n={0 .. 255}	
SGNo	U8	Control	Number of waveform generators	Optional	n={0 .. 255}	YES
SGSignalType<n>	U8	Control	Signal shape to be generated	Optional	User selectable	YES
SGUpdateRate<n>	U32	Update rate frequency used for signal generation	The analog output is updated using a frequency equals to the SGUpdateRatexf refo	SGUpdateRate<n>	U32, control	Update rate frequency used for signal generation
SGFreq<n>	U32	Control	DSS accumulator increment	Optional	n={0 .. 255}	YES
SGAmp<n>	U16	Control		Optional	n={0 .. 255}	YES
SGPhase<n>	U32	Control	Phase control	Optional	n={0 .. 255}	YES



Terminal Name	Data type	Type	Detail	Information	Values	Initialized before run?
SGFref<n>	U32	Indicator	Reference frequency	Optional	n={1..255}	

## 6.5 cRIO Examples

### 6.5.1 cRIO Basic requirements for the examples provided

This section describes the system requirements for running the examples using a NI CompactRIO chassis and seven NI CompactRIO I/O Modules hosted in the chassis. NI 9159, 14-slot CompactRIO Chassis, LX 110 FPGA, MXIe

- NI 9205 32-Ch  $\pm 200$  mV to  $\pm 10$  V, 16-Bit, 250 kS/s AI Module
- NI 9264 16-Ch  $\pm 10$  V, 16-Bit, 25 kS/s Analog Output Module
- NI 9477 32-Ch 24 V, 8  $\mu$ s, Sinking DO Module
- NI 9425 32-Ch 24 V, 7  $\mu$ s, Sinking DI Module
- NI 9476 32-Ch 24 V, 500  $\mu$ s, Sourcing DO Module
- NI 9426 32-Ch 24 V, 7  $\mu$ s, Sourcing DI Module
- NI 9401 8-Ch, 5 V/TTL High-Speed Bidirectional Digital I/O Module

The module placement for running the examples is depicted in Fig. 47.

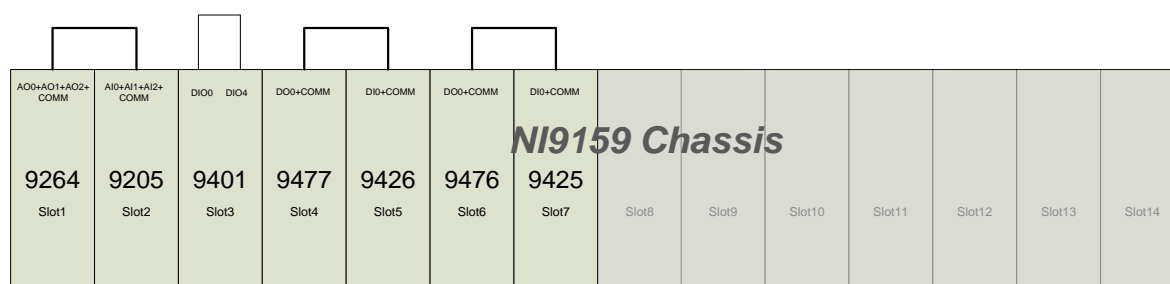
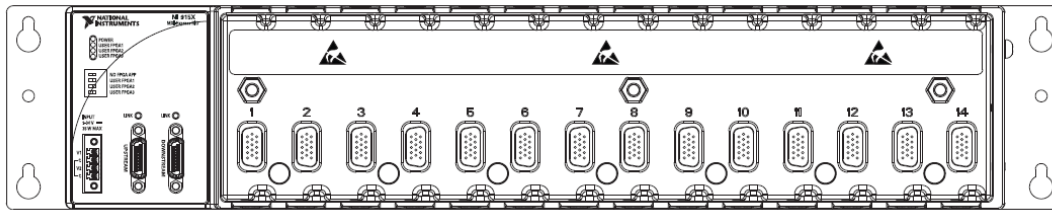


Fig. 47 NI9159 Chassis Generic Architecture

### 6.5.2 Module Identification in the Chassis

A NI9159 chassis has 14 slots for connecting compact RIO modules (see Fig. 48). The number scheme of the modules inserted in each slot are identified from Module 1 to Module 14 numbered from left to right. The examples described in these sections are implemented with 7 modules.



**Fig. 48 Chassis NI9159 and Slot Numbering Schema**

### 6.5.3 Module Description and Signal Interconnections

The subsequent section describes the modules used in the system and the signal interconnection among them supporting the connections schema depicted in Fig. 47.

### 6.5.3.1 NI9205 Analog Input Module

This module is used to measure analog input (AI) signals (see Table 32). The first three AI ports of NI9205 module is connected to the first three ports of the NI9264 module (Module 1) (see Fig. 49 and Fig. 50)

- AO<sub>0</sub> NI9264 Module<sub>1</sub> to AI<sub>0</sub> Module<sub>2</sub>
- AO<sub>1</sub> NI9264 Module<sub>1</sub> to AI<sub>1</sub> Module<sub>2</sub>
- AO<sub>2</sub> NI9264 Module<sub>1</sub> to AI<sub>2</sub> Module<sub>2</sub>

The COMM port of Module<sub>1</sub> and Module<sub>2</sub> is interconnected. Aforementioned connections are Referenced Single-Ended (RSE mode).

### Table 32 NI9205 Module Relevant Characteristics

NI9205 Module relevant characteristics (extracted from datasheet)	
Conversion Time	R Series Expansion chassis: 4.50μs
	All other chassis: 4.00μs

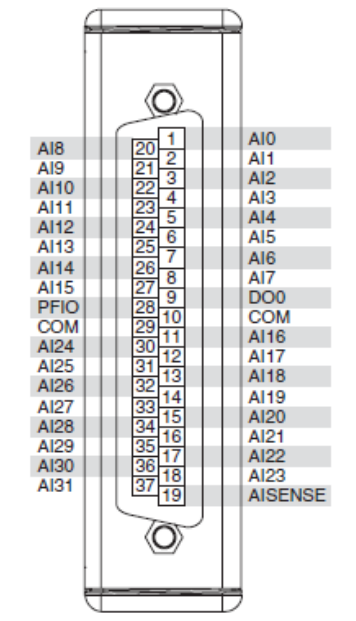


Fig. 49 NI9205 Signal Connector

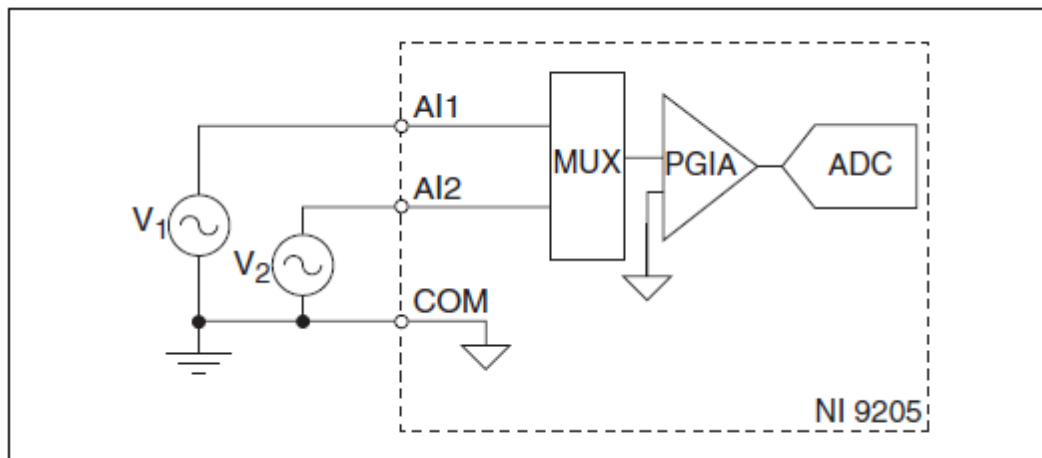
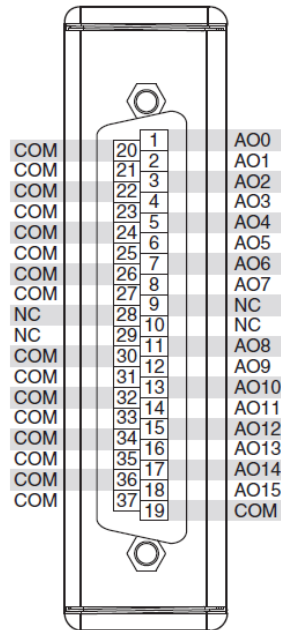


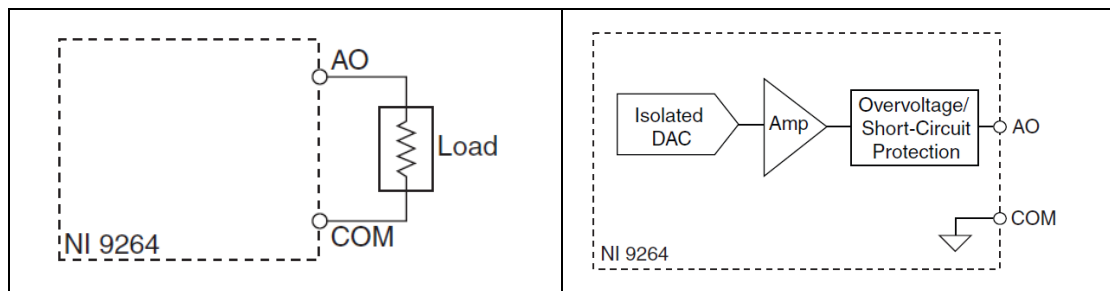
Fig. 50 Signal connection in RSE mode

### 6.5.3.2 NI9264 Analog Output Connection

The NI9264 (see Fig. 51) is used to generate analog output signals. The connection to Module2 is described in Table 34.



**Fig. 51 : Signals in the NI9264 analog output module**



**Fig. 52 Connection for analog output channels**

**Table 33 NI9264 relevant characteristics**

NI9264 Module relevant characteristics (extracted from datasheet)		
Settling time (100pF load, to 1 LSB)	20V Step	20 $\mu$ s
	1V Step	15 $\mu$ s
	0.1V Step	13 $\mu$ s
Update Time	1 Channel	3.1 $\mu$ s min.
	2 Channels	5.3 $\mu$ s min.
	3 Channels	7.5 $\mu$ s min.

**NI9264 Module relevant characteristics (extracted from datasheet)**

	16 Channels	37 $\mu$ s min.
--	-------------	-----------------

**Table 34 Interconnection between NI9264 and NI9205**

NI9264 Analog output channel	NI9205 Module/channel
Module <sub>1</sub> /Channel <sub>0</sub>	Module <sub>2</sub> /Channel <sub>0</sub>
Module <sub>1</sub> /Channel <sub>1</sub>	Module <sub>2</sub> /Channel <sub>1</sub>
Module <sub>1</sub> /Channel <sub>2</sub>	Module <sub>2</sub> /Channel <sub>2</sub>

**6.5.3.3 NI9401 Digital Input/Output**

The NI9401 is an 8-Channel Digital Input/Output TTL Module. The Port<sub>0</sub> and the Port<sub>4</sub> is externally interconnected.



**Warning.** This Module has to be configured to set the direction of the I/O ports

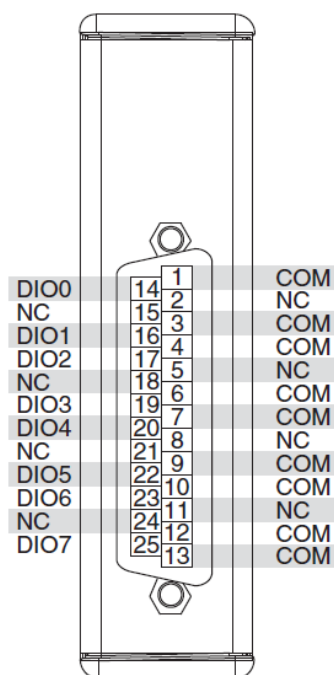


Fig. 53 Signal connector for NI9401

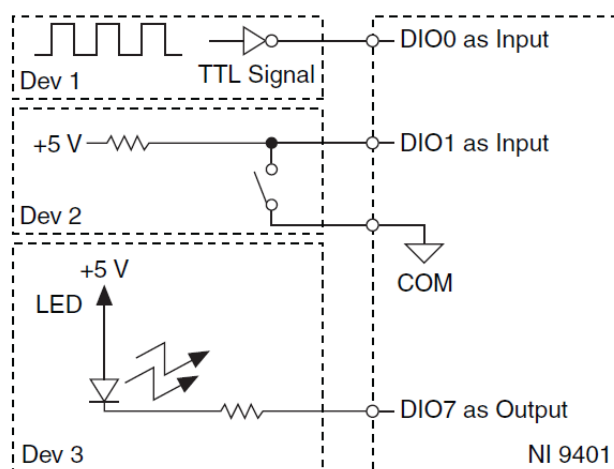


Fig. 54 Signal connection for Digital Input and Output in NI9401.

Table 35 NI9401 relevant characteristics

NI9401 Digital Input/Output Module relevant characteristics (extracted from datasheet)		
Maximum input signal switching frequency by number of input channels, per channel	8 input channels	9 MHz
	4 input channels	16 MHz
	2 input channels	30 MHz

### NI9401 Digital Input/Output Module relevant characteristics (extracted from datasheet)

Maximum output signal switching frequency by number of output channels with an output load of 1mA, 50pF, per channel	8 output channels	5 MHz
	4 output channels	10 MHz
	2 output channels	20 MHz

#### 6.5.3.4 NI9477 Digital sinking output module

The NI9477Sinking Digital Output module is connected to the Module<sub>5</sub> NI9426.

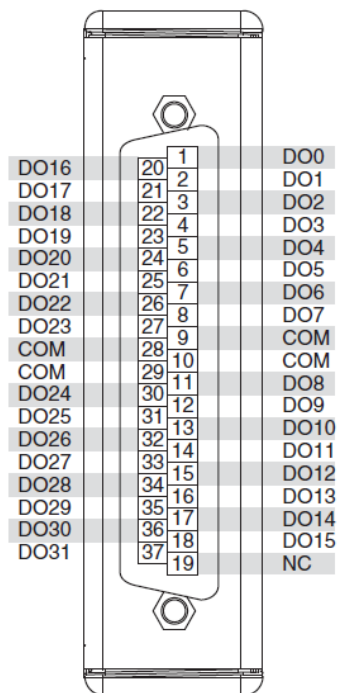


Fig. 55 NI9477 32 channel digital output Signal Connector

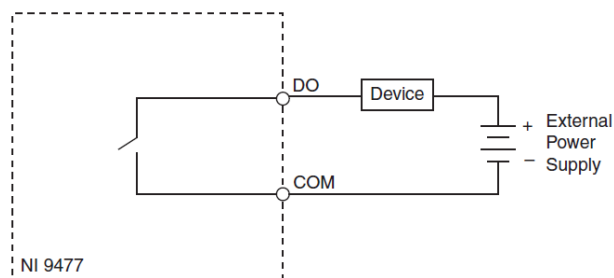


Fig. 56 Connection of an external device to NI9477

Table 36 NI9477 Relevant Characteristics

NI9477 0-60V Sinking Digital Output Module relevant characteristics (extracted from datasheet)	
Maximum Update Rate	8 $\mu$ s max.
Propagation Delay	1 $\mu$ s max.



**Warning.** Check the proper connection to avoid damages in the module.

### 6.5.3.5 NI9426 Sourcing Digital Input Module

The NI9426 is a sourcing digital input module. One digital output (DO) channel of the NI9477 Module<sub>4</sub> is connected to a digital input (DI) of the NI9426 Module<sub>5</sub>.

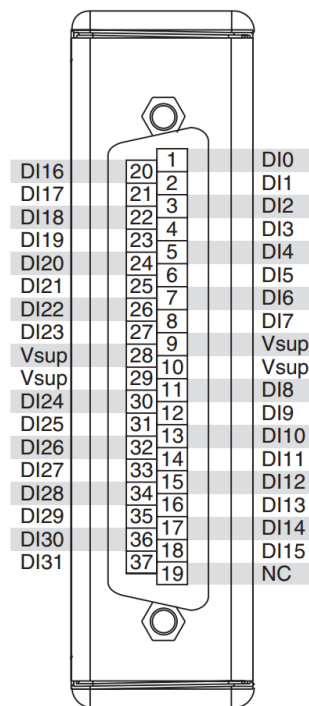
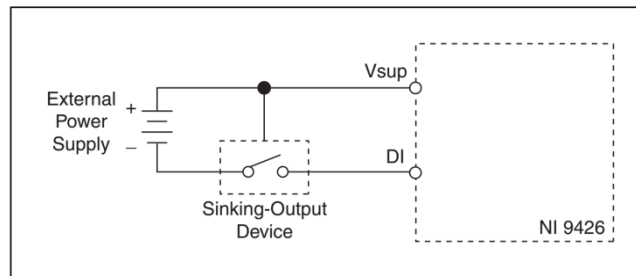


Fig. 57 Signals in the NI9426 digital input module





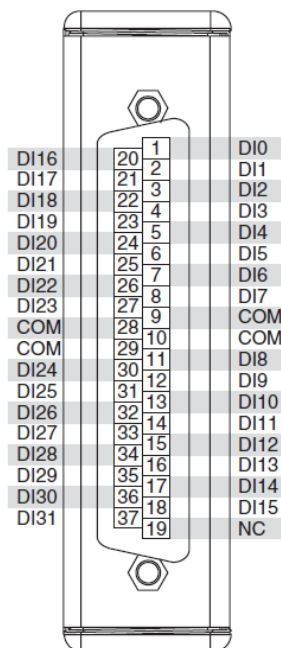
**Fig. 58 Connecting a device to the NI9426**

**Table 37 NI9426 Relevant Characteristics**

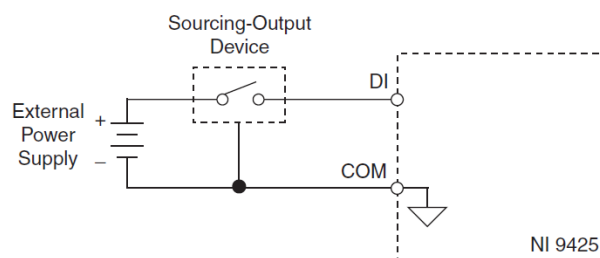
<b>NI9426 32 channel 24V Sourcing Digital Input Module relevant characteristics (extracted from datasheet)</b>	
Digital Logic Levels Input Voltages	OFF State $\geq (V_{sup} - 5 \text{ V})$
	ON State $\leq (V_{sup} - 10 \text{ V})$
Update/Transfer time	7 $\mu\text{s}$ max.
Setup time	1 $\mu\text{s}$ min.

### 6.5.3.6 NI9425 Sinking Digital Input Module

NI9425 is used as Digital Input (DI) module and it is connected to the NI9476 Module<sub>6</sub>.



**Fig. 59 Connection in NI9425 module**



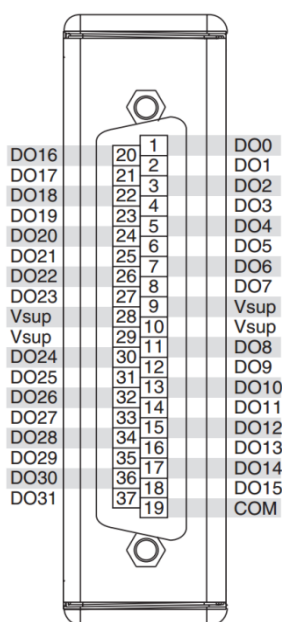
**Fig. 60 Connecting a device to the NI9425**

**Table 38 NI9425 Relevant Characteristics**

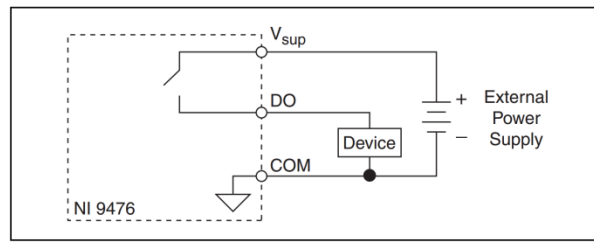
<b>NI9425 32 channel 24V Sinking Digital Input Module relevant characteristics (extracted from datasheet)</b>	
Digital Logic Levels Input Voltages	OFF State $\leq 5$ V
	ON State $\geq 10$ V
Update/Transfer time	7 $\mu$ s max.
Setup time	1 $\mu$ s min.

### 6.5.3.7 NI9476 Sourcing Digital Output Module

The NI9476 32-Channel 24V sourcing digital output module



**Fig. 61 NI9476 Signal Connector**



**Fig. 62 Connection of a device to the NI9476**

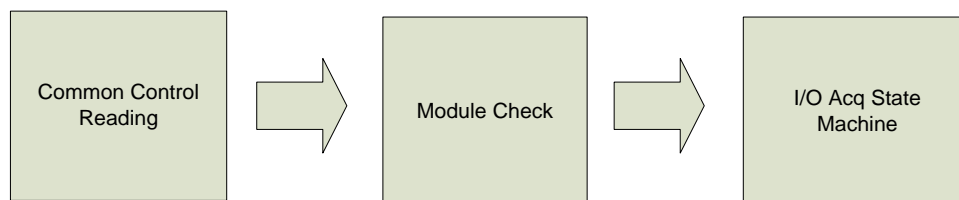
**Table 39 NI9476 Relevant Characteristics**

<b>NI9476 32 channel 24V Sourcing Digital Output Module relevant characteristics (extracted from datasheet)</b>	
Maximum Update Rate	40 $\mu$ s max.
Propagation Delay	500 $\mu$ s max.

## 6.5.4 System General Description

### 6.5.4.1 General Block Diagram

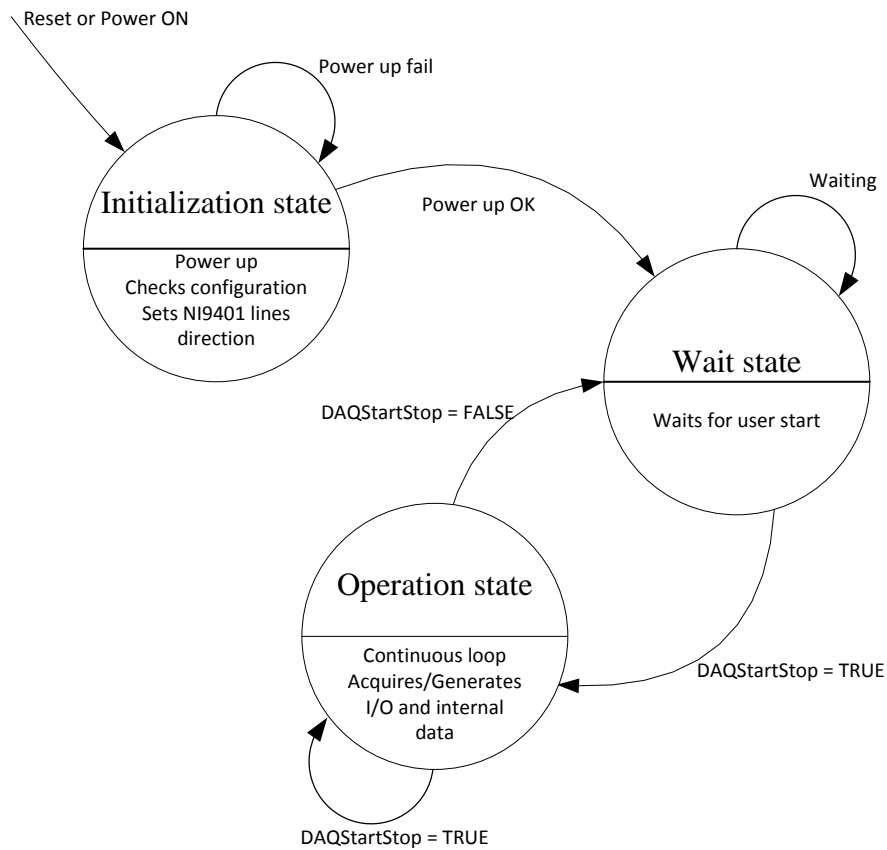
The two examples for cRIO architecture presented bellow have the same high level architecture depicted in



**Fig. 63 Functional Architecture**

### 6.5.4.2 State Machine

The system behaviour as a state machine composed by three main states, as represented Fig. 64. *Initialization state* performs the system power up, checks the system configuration (all modules are present in their corresponding chassis slot and working properly). *Wait state* waits for operator click start button in the HMI. The *Operation State Machine* (a.k.a. *I/O Acquisition Loop*) performs the acquisition and generation of signals in a continuous loop. The system continues its operation till the *DAQStartStop* remains true, if *DAQStartStop* changes to false the state machine returns to the *Initialization state*.



**Fig. 64 cRIO I/O Acquisition Loop State Machine**

#### 6.5.4.3 *Operation State: I/O Acquisition Loop*

This piece of logic implemented in the FPGA performs the main functionality of the modelled system. The concrete behaviour of each system is described in the following sections.

#### 6.5.4.4 *System Management: Host HMI*

The CompactRIO systems are based on reconfigurable FPGA chassis and I/O modules. The LabVIEW FPGA Module enables to translate this code directly to hardware. This process requires the compilation of the code to be synthesized to a bitfile (see Fig. 65).



**Fig. 65 LabVIEW FPGA Compilation Process**

To ease the process of downloading the bitfile to the FPGA, run and monitor the application, a host HMI application has been developed using LabVIEW on Windows OS. Running this application, the bitfile is downloaded to the FPGA target of the chassis and the state machine (described in 6.5.4.2) is started. From this point, the application remains communicating with the FPGA code to monitor the status of the system. Fig. 66 depicts the front panel of a generic host application.

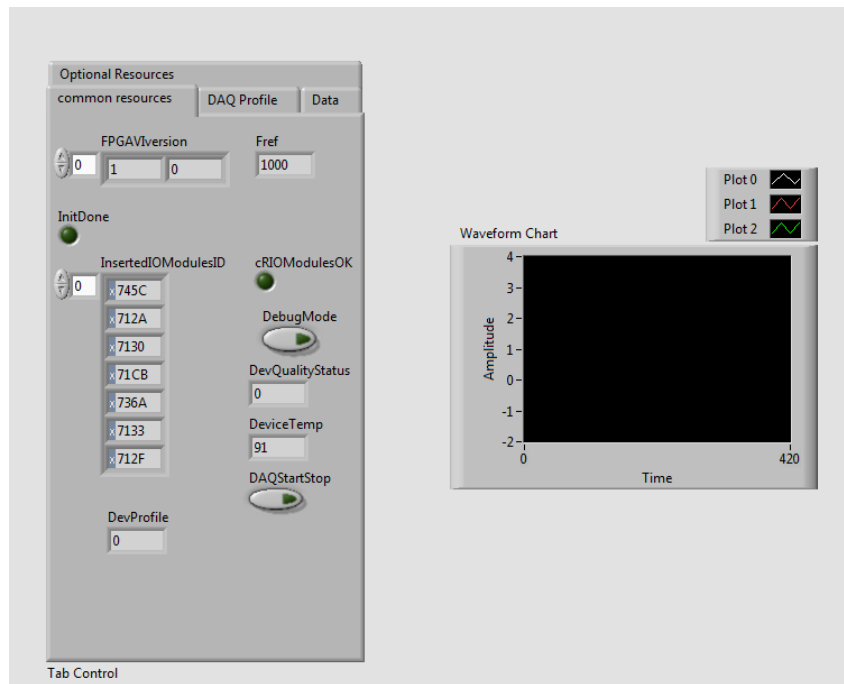


Fig. 66 HMI Host Application Front Panel

## 6.5.5 Point by Point DAQ Profile Example



Template Name: cRIOIO.lvproj

### 6.5.5.1 Objective

The intention of the subsequent template is to implement the acquisition and generation of analog and digital signals using the cRIO platform. This template provides/generates one sample in every period of time configured in the cRIO (sampling rate).

### 6.5.5.2 cRIO Hardware Elements Used

Table 40 summarizes the cRIO modules used in the chassis and its allocation.

**Table 40 Allocation of cRIO Modules in one NI9159 Chassis**

NI9159 Slot							
1	2	3	4	5	6	7	8 - 14
NI9264	NI9205	NI9401	NI9477	NI9426	NI9476	NI9425	Free



**Warning. The position of the cRIO modules in the chassis is mandatory.**

### 6.5.5.3 *Signal Connection*

The signal connections for properly run this example are described in section 6.5.3.

### 6.5.5.4 *Mandatory Resources for Point by Point I/O Profile*

The mandatory resources for these types of systems are described in section 6.4.1.



**Fig. 67 Mandatory Resources for point by point I/O profile**

### 6.5.5.5 *Optional Resources*

The optional resources of this type of profile are described in section 6.4.2. The optional resources implemented in this example are used for

- Analog Output signal generation

- Analog Input signal acquisition
- Digital Output signal generation
- Digital Input signal acquisition

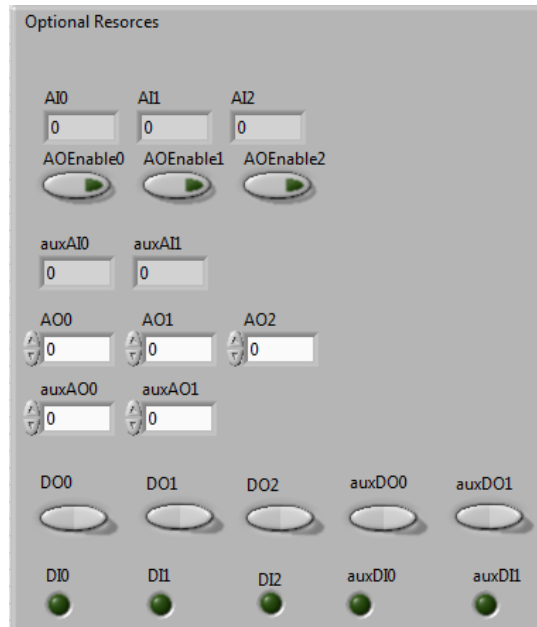


Fig. 68 Optional Resources Implemented in the Point by Point Example

#### 6.5.5.6 *LabVIEW Implementation for a cRIO Point by Point DAQ*

Fig. 69 shows the LabVIEW project for this template.

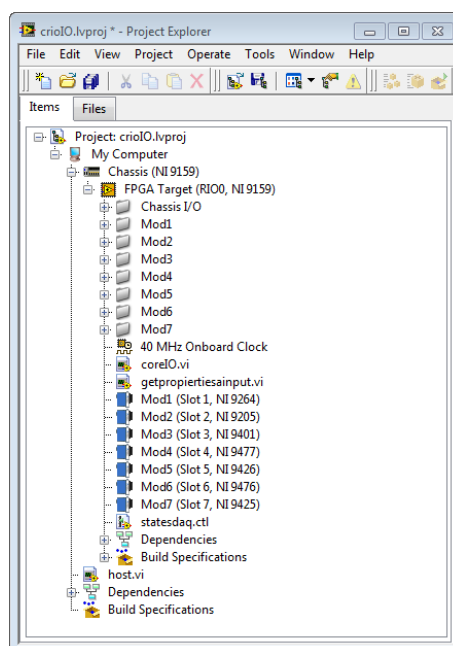


Fig. 69 LabVIEW Project for Point by Point Example

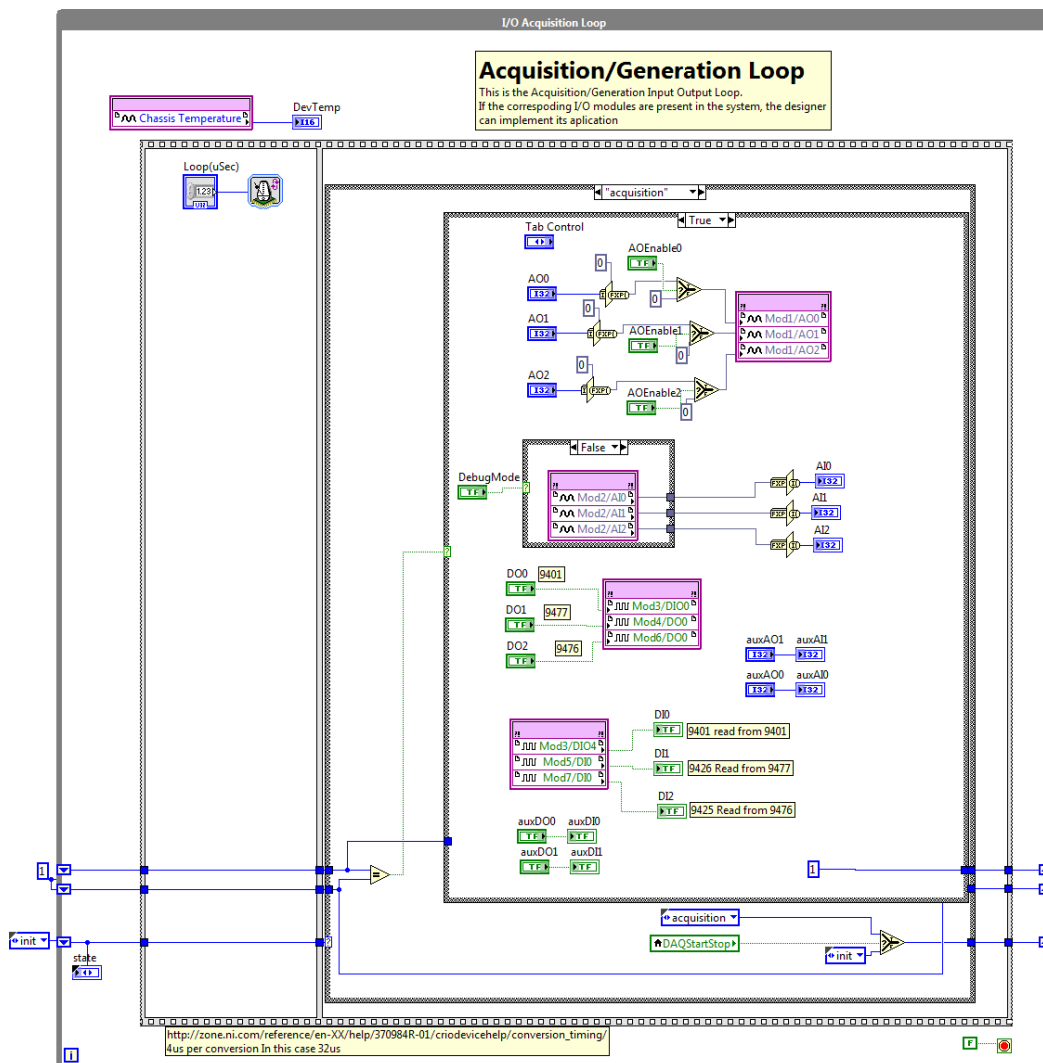


Fig. 70 I/O Acquisition Loop State Machine

Table 41: Resources identification

Terminals	I/O	Datatype	Info
AO<0-2>	Control	I32	If AOEnable<0-2> is true the I32 data converted to FXP is generated in its analog output pin of the NI9205 Module.
AOEnable<0-2>	Control	Boolean	Enables the Analog Output signal generation for channel 0 to 2 respectively.
Debug Mode	Control	Boolean	If False acquisition from NI9205 is performed
AI<0-2>	Indicator	I32	NI9205 Acquisition and converted from FXP read to I32



Terminals	I/O	Datatype	Info
			data types
DO0	Control	Boolean	NI9401 Channel 0 generation
DO1	Control	Boolean	NI9477 Channel 0 generation
DO2	Control	Boolean	NI9476 Channel 0 generation
DI0	Indicator	Boolean	NI9401 Channel 4 read from NI9401 Channel 0
DI1	Indicator	Boolean	NI9426 Channel 0 read from NI9477 Channel 0
DI2	Indicator	Boolean	NI9425 Channel 0 read from NI9476 Channel 0
auxDO<0-1>	Control	Boolean	Terminals not connected to any output hardware, only connected internally to auxDI<0-1> respectively
auxDI<0-1>	Indicator	Boolean	Indicators wired to aforementioned auxDO<0-1>
auxAO<0-1>	Control	I32	Terminals not connected to any output hardware, only connected internally to auxAI<0-1> respectively
auxAI<0-1>	Indicator	I32	Indicators wired to aforementioned auxAO<0-1>
State	Indicator	Enum	Indicates the state of the I/O Acquisition Loop State Machine

#### 6.5.5.7 Host HMI Program

The host program links to the FPGA controls and indicators in order to download the Bitfile to the target, configure its parameters and control and monitor the implemented instrument. The input signals read are plotted in a waveform graph.

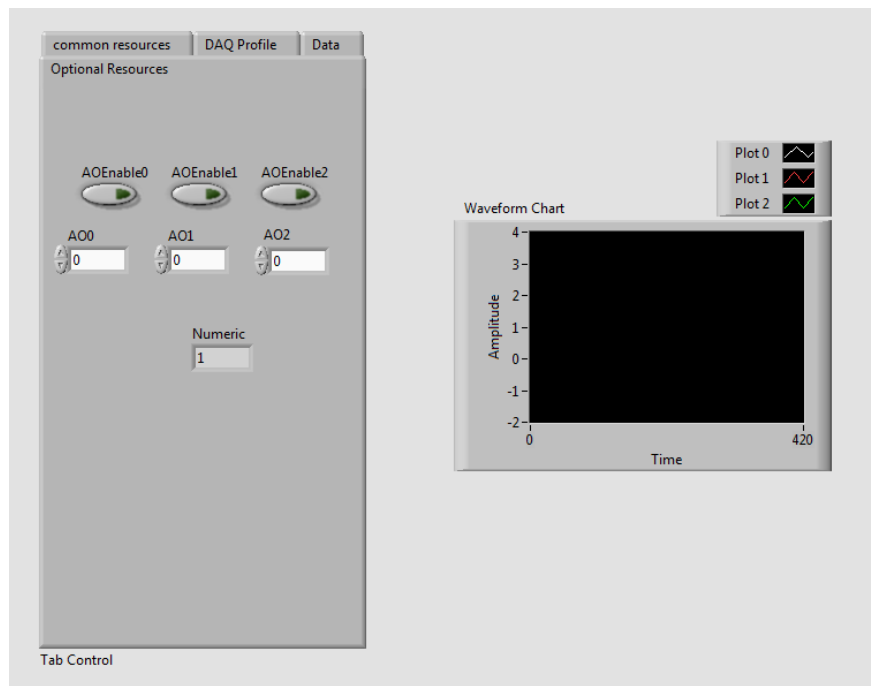


Fig. 71 HMI Front Panel of Point by Point Example

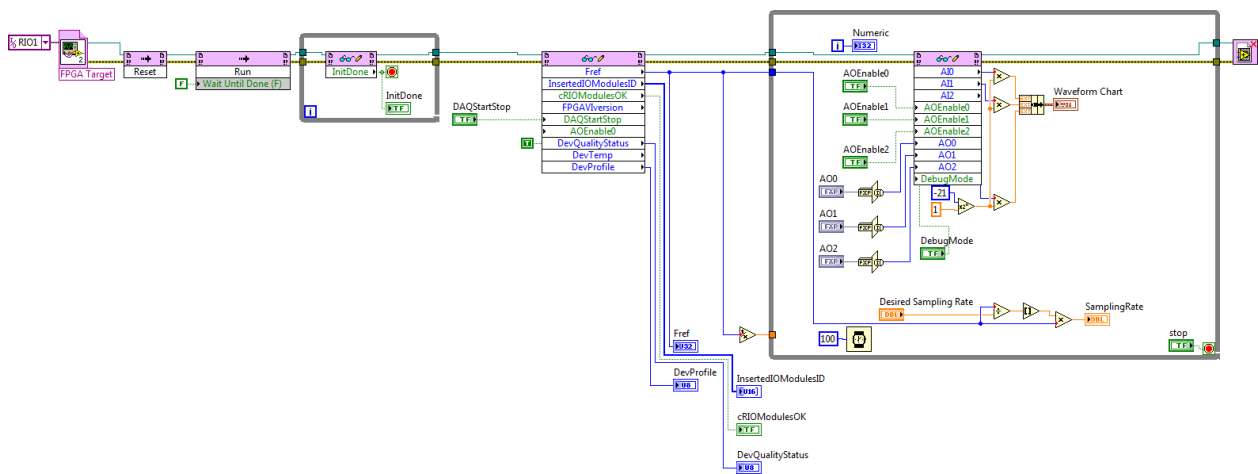


Fig. 72 HMI Block Diagram of Point by Point Example

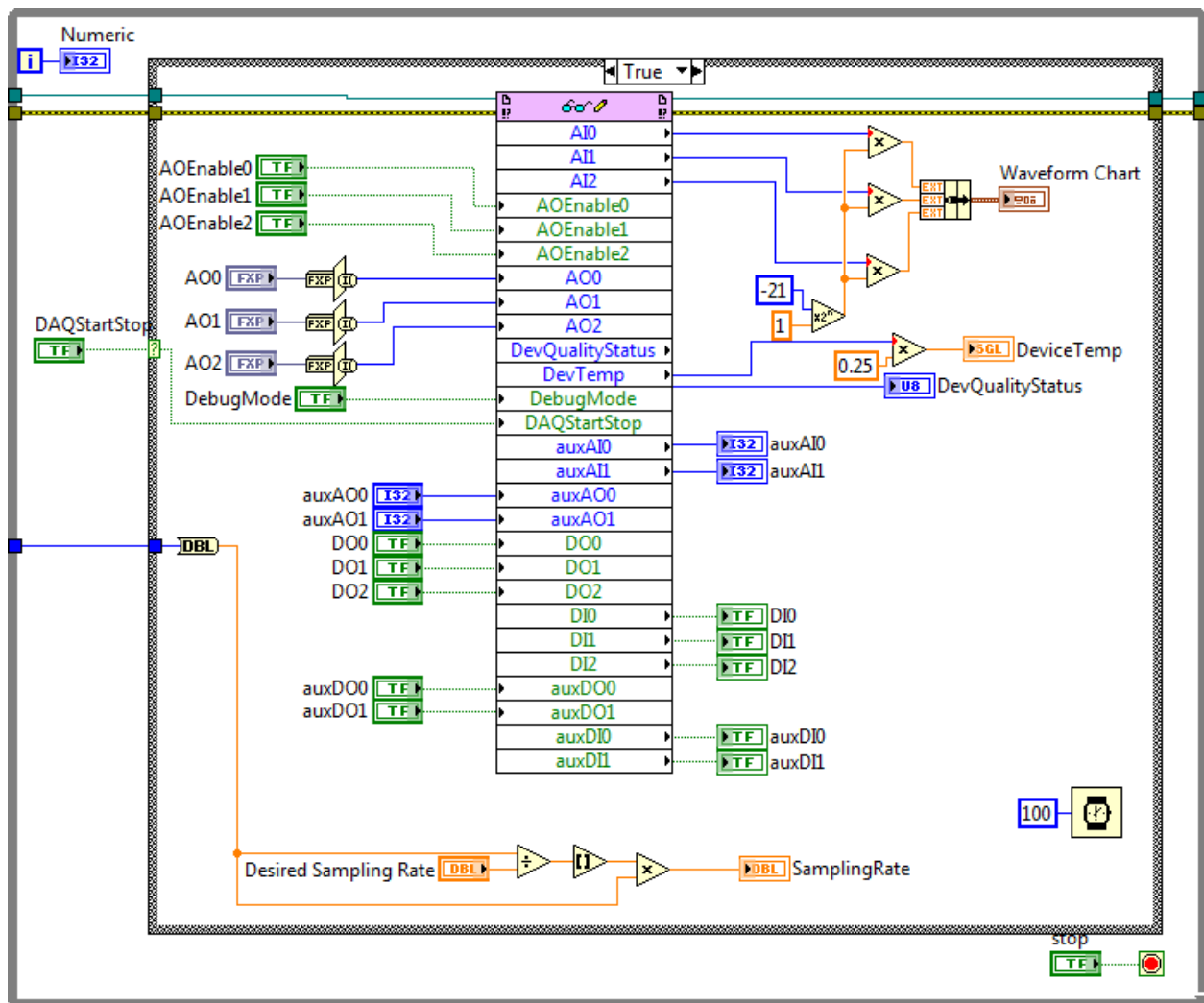


Fig. 73 HMI Main Acquisition Loop of Point by Point Example

## 6.5.6 Analog Signal DAQ Profile (DMA based) Example

### 6.5.6.1 Objective

The intention of the subsequent template is to implement data acquisition applications acquiring continuously block of samples using DMA.

### 6.5.6.2 cRIO Hardware Elements Used

Table 40 summarizes the cRIO modules used in the chassis and its allocation.

Table 42 Allocation of cRIO Modules in the NI9159 Chassis

NI9159 Slot							
1	2	3	4	5	6	7	8 - 14

NI9159 Slot							
NI9264	NI9205	NI9401	NI9477	NI9426	NI9476	NI9425	Free



**Warning.** The position of the cRIO modules in the chassis is mandatory.

### 6.5.6.3 Signal Connection

The signal connections for properly run this example are described in section 6.5.3.

### 6.5.6.4 Mandatory Resources for Analog Signal DAQ Profile

The mandatory resources for these type of systems is described in section 6.3.1

**Fig. 74 Mandatory Resources for Analog Signal Example**

### 6.5.6.5 Optional Resources

Optional resources for this type of profile are described in section 6.3.3

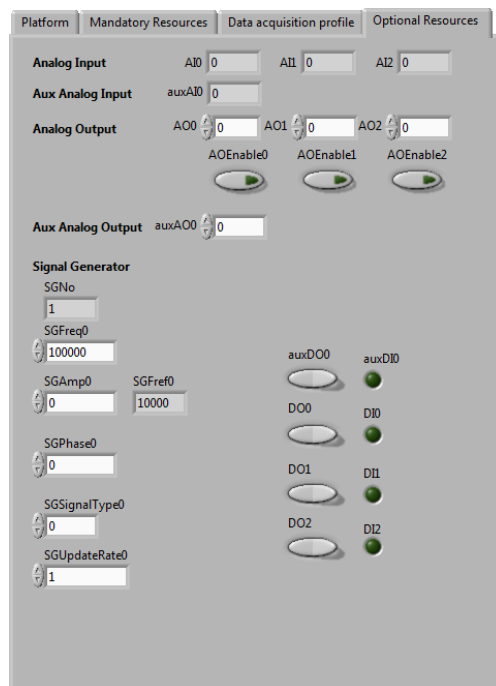


Fig. 75 Optional Resources for Analog Signal Example

#### 6.5.6.6 *LabVIEW Implementation for a cRIO Analog Signal DAQ Profile*

Fig. 76 shows the LabVIEW project for this template. Fig. 77 and Fig. 78 display the LabVIEW code implementing the functionality. Table 43 summarizes the terminals used.

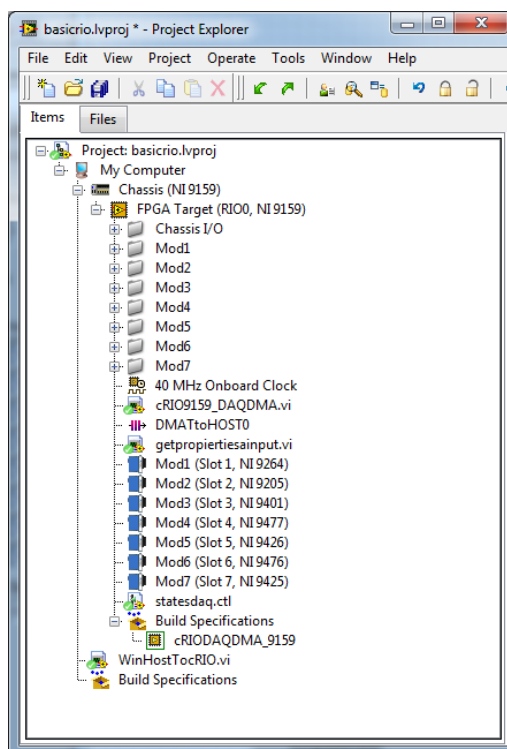


Fig. 76 LabVIEW Project for Analog Signal Example

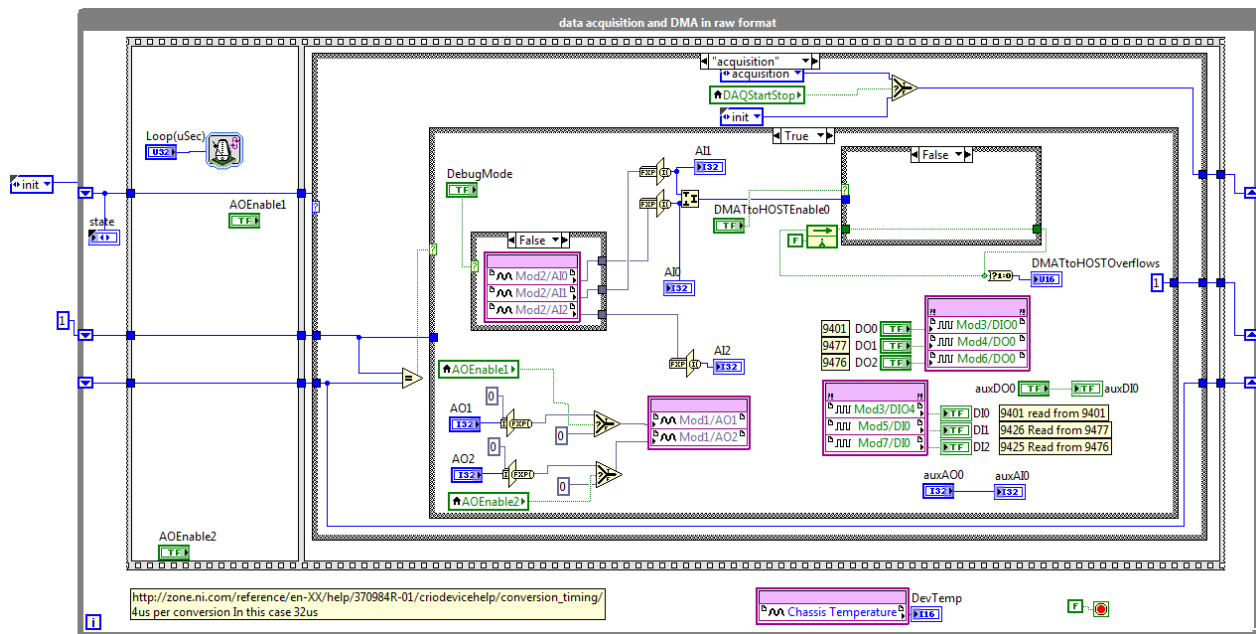


Fig. 77 Acquisition Loop State Machine

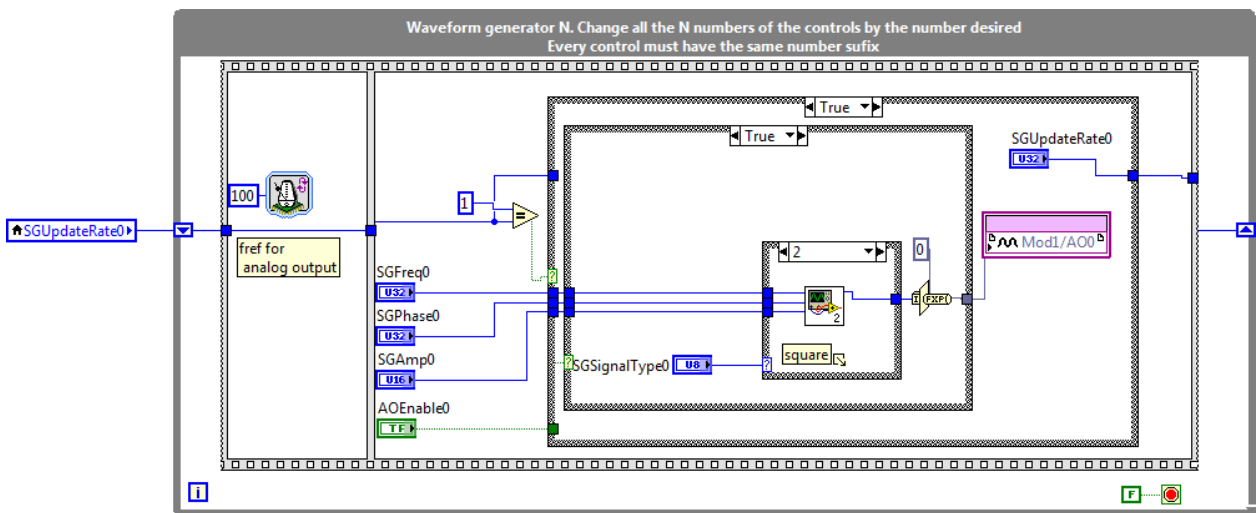


Fig. 78 DDS Signal Generation

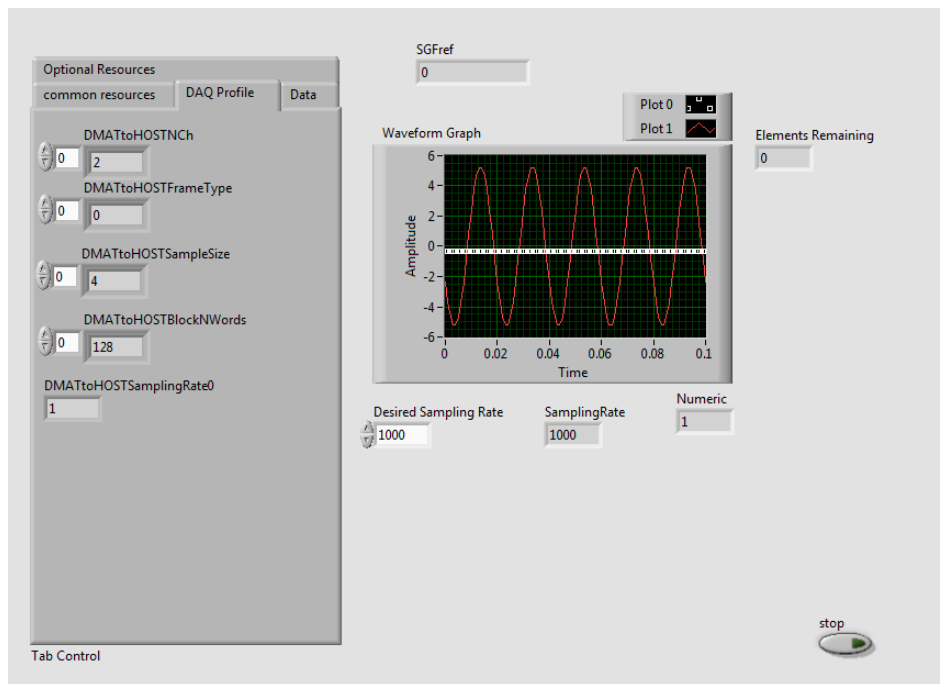
Table 43: Summary of the terminals

Terminals	I/O	Datatype	Info
AO0	Control	I32	<p>If AOEnable0 is TRUE the NI9264 output</p> <ul style="list-style-type: none"> <li>• AO0 (SGSignalType0 = 0)</li> <li>• DDS Sinusoidal Signal (SGSignalType0 = 1)</li> <li>• DDS Square Signal (SGSignalType0 = 2)</li> <li>• DDS Triangle Signal (SGSignalType0 = 3)</li> </ul>

Terminals	I/O	Datatype	Info
AO<1-2>	Control	I32	If AOEnable<1-2> is true the I32 data converted to FXP is generated in its analog output pin of the NI9205 Module.
AOEnable<0-2>	Control	Boolean	Enables the Analog Output signal generation for channel 0 to 2 respectively.
Debug Mode	Control	Boolean	If False acquisition from NI9205 is performed
AI<0-2>	Indicator	I32	NI9205 Acquisition and converted from FXP read to I32 data types
auxAO0	Control	I32	Terminal not connected to any output hardware, only connected internally to auxAI0 respectively
auxAI0	Indicator	I32	Indicator wired to aforementioned auxAO0
State	Indicator	Enum	Indicates the state of the I/O Acquisition Loop State Machine
SGNo	Indicator	U8	See section 6.3.3
SGFreq0	Control	U32	See section 6.3.3
SGAmp0	Control	U16	See section 6.3.3
SGPhase0	Control	U32	See section 6.3.3
SGSignalType0	Control	U8	See section 6.3.3
SGUpdateRate0	Control	U32	See section 6.3.3

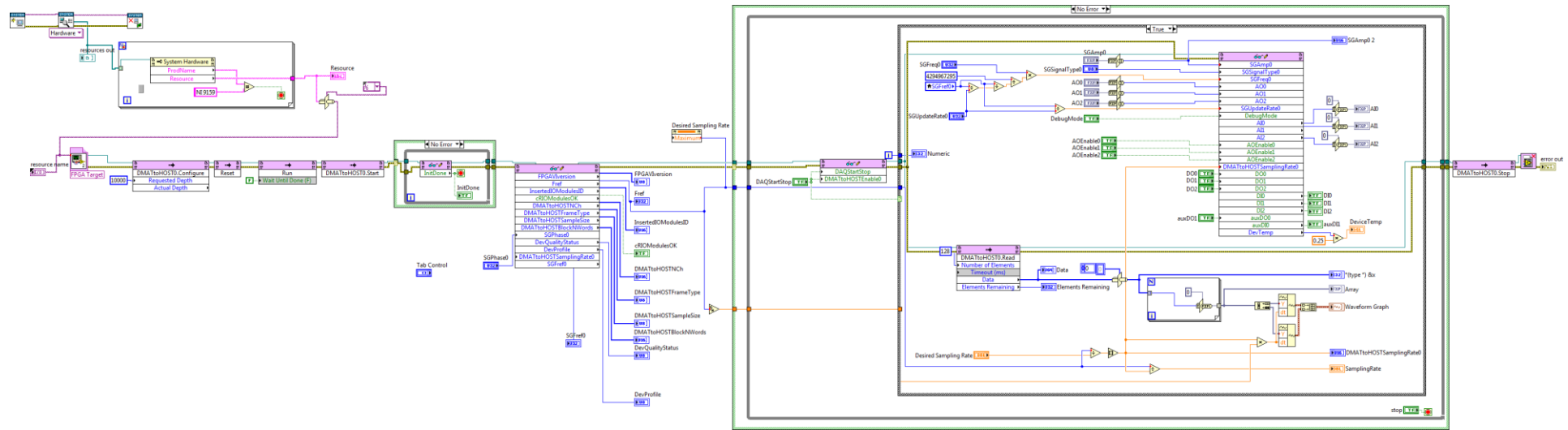
#### 6.5.6.7 Host HMI Program

The host program links to the FPGA controls, indicators and DMAs in order to download the bitfile to the target, configure its parameters and control and monitor the implemented application. The DMA-to-Host is read in the host main acquisition loop (see Fig. 80 and Fig. 81) and the input signals are plotted in a waveform graph.

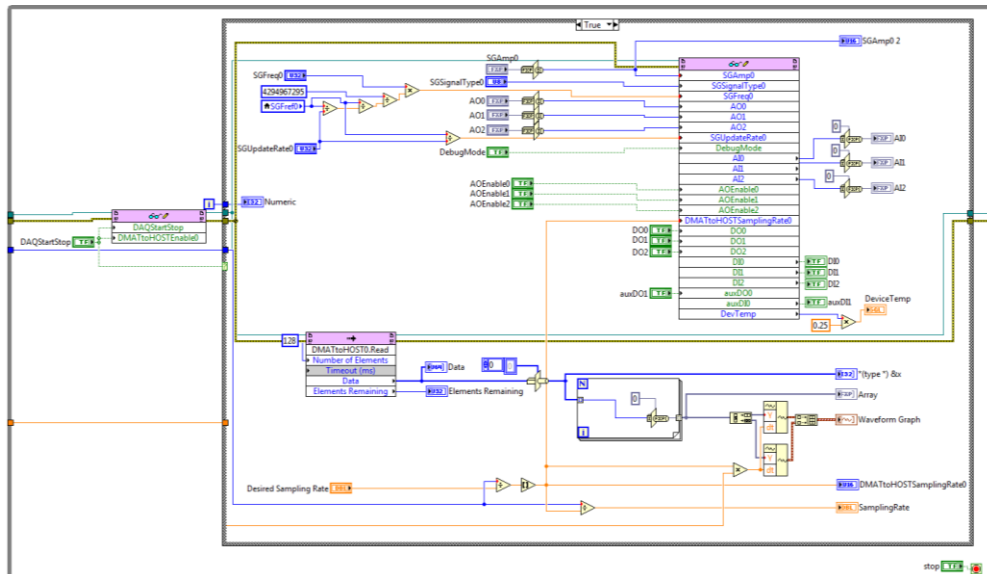


**Fig. 79 HMI Front Panel of Analog Signal DAQ Example**





**Fig. 80 Block Diagram of Analog Signal DAQ Example**



**Fig. 81 Host Main Acquisition Loop of Analog Signal DAQ Example**

## 7 LABVIEW FOR FPGA TEMPLATES

### 7.1.1 Location of the templates in GitHub repository

The templates are available in [https://github.com/irio-i2a2/IRIO\\_LabVIEW\\_Test\\_Templates/releases](https://github.com/irio-i2a2/IRIO_LabVIEW_Test_Templates/releases)

### 7.1.2 LabVIEW template directory structure

The LabVIEW templates are organized in two separate folders: cRIO and FlexRIO. Subfolder organization is summarized in Table 44.

Table 44: Folder organization for the templates

Templates	Subfolder name	Target	Template organization	folder
cRIO	dataacquisitionDMA pointbypoint	compactRIO (NI9159)	<tn>.lvproj <fpga>.vi <host>.vi FPGA Bitfiles datatypes	
FlexRIO	FlexRIOOnlyResources NI1483 NI5761 NI6581 nomodule	PXIe796x PXIe7966R/NI1483 PXIe7966R/NI5761 PXIe7961R/NI6581 PXIe796x	<tn>.lvproj <fpga>.vi <host>.vi FPGA Bitfiles datatypes	

## 7.2 FlexRIO templates

Detailed in Table 45, the definition of the main elements to take into account for managing and developing LabVIEW templates for FlexRIO bundles are:

- **Bundle:** It is the set of FlexRIO devices plus a NI adapter module. Below in the table are listed the bundles supported. Note that there are two bundles that do not use any adapter module. These ones are templates examples that are used for testing purposes and are templates for starting new developments.
- **LabVIEW Project:** It is the file path that contains the information of the LabVIEW project for the corresponding template.
- **VI Template:** It contains all the LabVIEW code for defining the FlexRIO (FPGA) hardware configuration. This file contains LabVIEW code commented with instructions, to help FPGA project developer to start new designs.

- **Windows Bitfile:** This is the bitfile already compiled for the template defined in the *VI Template*. This file is used to configure the FlexRIO device from a Windows Host.
- **Linux bitfile & header file:** For using the bitfiles with IRIO library, it is necessary to generate a new bitfile and a header file from the *Windows Bitfile* using the NI FPGA C API generator tool. Among the files generated only bitfile and header file (with the same name) are required.

**Table 45 FlexRIO Templates Information**

Bundle	LabVIEW Project	VI Template	Windows Bitfile	Linux Bitfile & header file
<b>PXIe-796XR*</b>	FlexRIO/nomodule/ FlexRIOOnoModule.lvproj	FlexRIOOnoModule.vi	PXIe-796XR_FlexRIOOnoModule_v1_1.lvbitx	NiFpga_FlexRIOOnoModule_796X.lvbitx NiFpga_FlexRIOOnoModule_796X.h
<b>PXIe-796XR*</b>	FlexRIO/nomodule/ FlexRIOOnlyResources.lvproj	FlexRIOOnlyResources.vi	PXIe-796XR_FlexRIOOnlyResources_v1_1.lvbitx	NiFpga_FlexRIOOnlyResources_796X.lvbitx NiFpga_FlexRIOOnlyResources_796X.h
<b>PXIe-7966R + NI-5761**</b>	FlexRIO/NI5761/FlexRIO5761.lvproj	FlexRIO5761.vi	PXIe-796XR_FlexRIOMod5761_v1_1.lvbitx	NiFpga_FlexRIOMod5761_796X.lvbitx NiFpga_FlexRIOMod5761_796X.h
<b>PXIe-7966R + NI-6581***</b>	FlexRIO/NI6581/FlexRIO6581.lvproj	FlexRIO6581.vi	PXIe-796XR_FlexRIOMod6581_v1_1.lvbitx	NiFpga_FlexRIOMod6581_796X.lvbitx NiFpga_FlexRIOMod6581_796X.h
<b>PXIe-7966R + NI-1483***</b>	FlexRIO/NI1483/FPGA148/ FlexRIO1483_8Tap8/FPGA1483	FlexRIO1483_8T8.vi	PXIe-796XR_FlexRIOMod1483_8T8_v1_1.lvbitx	NiFpga_FlexRIOMod1483_8T8_796X.lvbitx NiFpga_FlexRIOMod1483_8T8_796X.h

\*The FlexRIO targets for which the templates has been generated are: **PXIe-7961R, PXIe-7965R, PXIe-7966R**

\*\*The FlexRIO targets for which the templates has been generated are: **PXIe-7961R, PXIe-7965R**

\*\*\*The FlexRIO targets for which the templates has been generated are: **PXIe-7965R, PXIe-7966R**

## 7.3 cRIO templates

The main elements to take into account for managing and developing LabVIEW templates for cRIO are:

<b>cRIO platform</b>	<b>LabVIEW Project</b>	<b>VI Template</b>	<b>Windows Bitfile</b>	<b>Linux Bitfile &amp; header file</b>
<b>NI9159</b>	cRIO/pointbypoint/cRIOIO.lvproj	cRIO9159_IO.vi	cRIO9159_cRIOIO_v1_1.lvbitx	NiFpga_cRIOIO_9159.lvbitx NiFpga_cRIOIO_9159.h
<b>NI9159</b>	cRIO/dataacquisitionDMA/basicrio.lvproj	cRIO9159_DAQDMA.vi	cRIO9159_cRIODAQDMA_v1_1.lvbitx	NiFpga_cRIODAQDMA_9159.lvbitx NiFpga_cRIODAQDMA_9159.h

## 8 USING THE LABVIEW TEMPLATES

### 8.1 Overview

Templates are complete LabVIEW for FPGA projects implemented using the design rules for FlexRIO and cRIO devices. This means that are fully compliant with the IRIO design rules methodology. The bitfiles obtained from these templates can be used to program FlexRIO/cRIO devices using IRIO library.

The FPGA project developer is in charge of the definition of the FlexRIO/cRIO hardware functionalities, helped by the predefined LabVIEW templates provided and taking into account the maximum I/O resources that each device contains and.

For instance, the template FlexRIO6581, has defined the use of 8 digital lines of the adapter module as digital outputs, and 8 digital lines as digital inputs. One functionality implemented can be to read digital lines from one 8 bit digital port from the NI6581 adapter module, and the second functionality can be to write to one of the 8 bit digital port. Although the NI6581 has 54 digital I/O, only some of them have been used in the template.

Other interesting example to demonstrate the purpose of the templates is the case of FlexRIOonModule template. This template is implemented for a FlexRIO device without using any adapter module (this means without using physical I/O). The template presents one waveform generator (signal generator) connected to one DMA channel of a DMA implemented for acquiring four analog input channels internally simulated. This demonstrates that it is not mandatory to connect the device to any I/O module to develop applications.

The templates provide to the FPGA project developer:

- An overview of the possibilities that can be implemented using the IRIO design rules.
- A development starting point for reusing the template to achieve the final design required for the application.

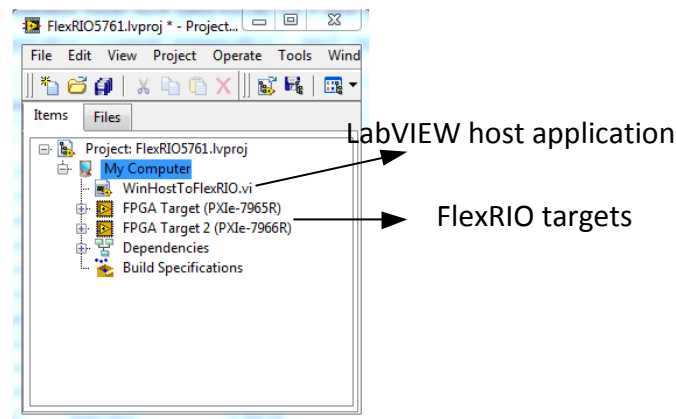
The following sections describe the basic concepts that the FPGA developer should apply to new developments based in aforementioned templates.

### 8.2 Templates

In this section describes the basic steps to start working with the LabVIEW templates.

#### 8.2.1 LabVIEW template browser description

Fig. 82 depicts the project browser corresponding to a template (in this case FlexRIO+5761). The LabVIEW host application for all templates is called *WinHostToFlexRIO.vi* and it is used for testing the FPGA code in a Windows platform. The FlexRIO targets are the devices for which this template has been compiled. More targets can be added easily.



**Fig. 82: LabVIEW Project browser**

The revision history of the LabVIEW FPGA implementation for a determined device can be checked by clicking with right mouse button on it, selecting properties, and pressing revision history button.

## 8.2.2 Folder Libs

Libs folder, found inside some FPGA targets of the templates, provide the developer VIs for implementing some predefined functionalities to the FPGA design. These VIs can be duplicated and modified as the developer considers.

The **SGTemplate.vi** includes a signal generator to the FPGA which can be copied and pasted to a determined implementation.

The **diagnostic.vi** is an empty VI template that has one input and three outputs. These input and outputs terminals can be changed (type of data, and more I/O terminals can be added) according to the requirements of the FPGA developer. Inside this VI, the developer can implement its diagnostics (e.g. for an input).

The **GlobalReg.vi** contains all the global registers (terminals) that can be accessed from any part of code implemented for the FPGA. By default the VI contains the mandatory terminals like DAQStartStop, DMAOverflow, etc., and other ones used for all templates. If more global registers are required the developer can add them in this VI.

## 8.2.3 Target Clocks

As every design for FPGA time domain clocks are required, according to the hardware requirements and the adapter module constrains. For this template (FlexRIO6761), apart from the 40MHz On-board Clock, there are several clocks that can be useful for new designs. Nevertheless, more clocks derived from the ones that are present in the project browser can be added.

## 8.2.4 DMAs to Host

64bit DMA-to-Host resources can be found depending on the template, named as the IRIO design rules stipulate. Additional DMAs or internal FIFOs can be added up to the FPGA hardware resources limit.

## 8.2.5 NI Adapter module

Extending the IO Module item, all the terminals provided for the IO module CLIP, are listed. To use them, drag and drop them into the VI FPGA design. If another Adapter module is required, then click right mouse button on the IO module icon, and then select properties, the menu will appear, and on the General category, select the IO module required, and the desired component level IP for that module.

## 8.2.6 Build specifications

For building the bitfile for the corresponding main VI (in this case FlexRIO5761), it is required to define a *build specification*.

## 8.2.7 LabVIEW template VI

### 8.2.7.1 Control panel

After clicking on the LabVIEW template VI, the control panel will appear. Here, depending on the target (FlexRIO or cRIO) some differences can be found. The developer will have to configure some mandatory controls in order to fit the developer requirements. After applying new values on the controls, it is required to click on it with the right mouse button and select: **Data operations-> Make Current Value Default**.

**Platform Tab:** Select the type of target device for which the bitfile is going to be built (Fig. 83).

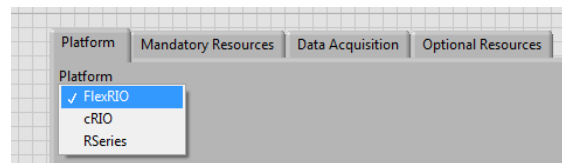


Fig. 83: Selection of the hardware platform

**Mandatory Resources Tab:** Here there are mandatory controls and indicators that the VI must have according to this document. The FPGA developer has to insert the Fref value, the DevProfile and if required, change the FPGAVersion. DAQStartStop and DebugMode have to be initialized to false.

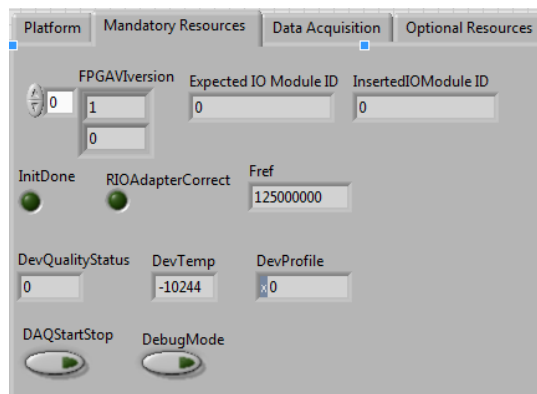


Fig. 84: Mandatory resources

**Data Acquisition tab:** The developer has to fill array controls for DMA-to-Host configuration. This means that not only the content of the arrays will be configured, but also the size of the arrays. There are four arrays: DMATtoHOSTNCh, DMATtoHOSTFrameType, DMATtoHOSTSampleSize,



and DMATtoHOSTBlockNWords. These arrays must be configured with the same size, according to the number of DMAs used. The value configured at every array address position of every array is applied to the corresponding DMA. In the FlexRIO5761 template, the values by default are:

- Size of 1 for every array. This means that there is only one DMA in the FPGA design.
- 4 channels are packaged in a 64 bit word DMA.
- FrameType, to configure the packing of the data, in this case type 0.
- Size in bytes used per channel is: 2 bytes.
- Number of words of 64bit, to tell the IRIO driver the amount of data to form an entire data block, in this case 4096 64 bit words.

The developer should change these values according to its requirements. The default values for DMATtoHOSTEnable0 and DMATtoHostOverflows are false and 0.

The screenshot shows the 'Data Acquisition' tab of a configuration interface. It contains the following settings:

- DMATtoHOSTNCh:** A numeric input field with the value 4.
- DMATtoHOSTFrameType:** A numeric input field with the value 0.
- DMATtoHOSTSampleSize:** A numeric input field with the value 2.
- DMATtoHOSTBlockNWords:** A numeric input field with the value 4096.
- DMATtoHOSTSamplingRate0:** A numeric input field with the value 10000.
- DMATtoHOSTEnable0:** A checkbox that is currently checked.
- DMATtoHOSTOverflows:** A numeric input field with the value 0.

**Fig. 85: Data acquisition tab.**

### **Optional Resources tab:**

Present here there additional controls and indicators to provide extra functionalities to the FPGA design. They have to be compliant with the IRIO design rules (name nomenclature, and data type). The developer should insert here all the additional controls and indicators that are going to be added to the template.

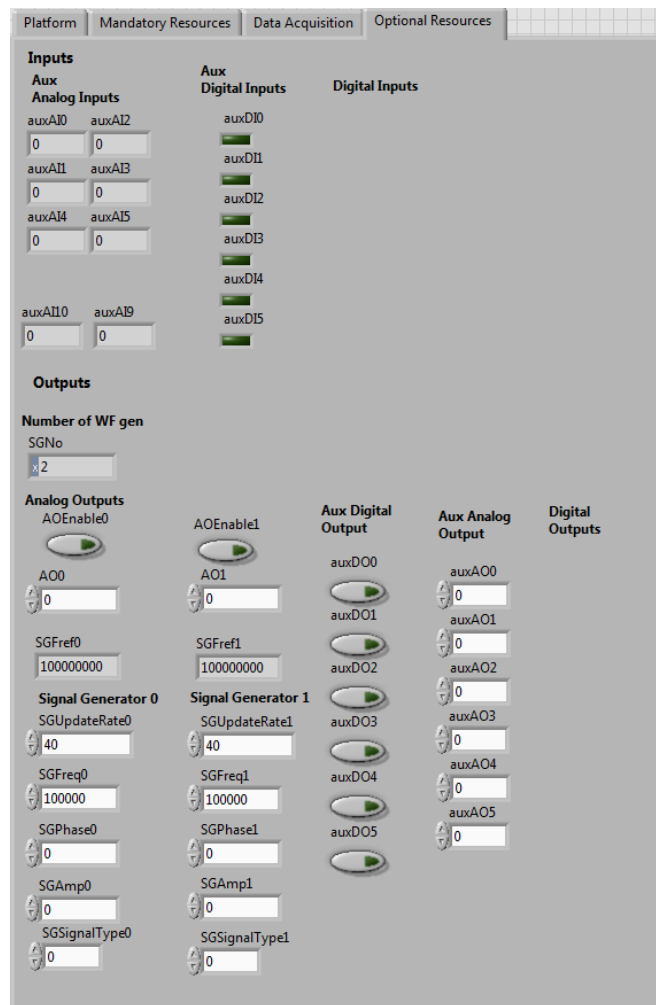


Fig. 86: Optional Resources

### 8.2.7.2 Block diagram

On the left side of the block diagram, there is the main part of LabVIEW code that will be “executed” in the FPGA (see Fig. 87). Here it is the initialization of some indicators that will provide information to the IRIO library. The second step of the sequence structure, checks if the adapter module connected to the FlexRIO had been recognized and correctly initialized.

On the left side of the block diagram the code to perform the initialization is present, the initialization of some indicators and the identification and initialization of the adapter module/s connected (see Fig. 87).

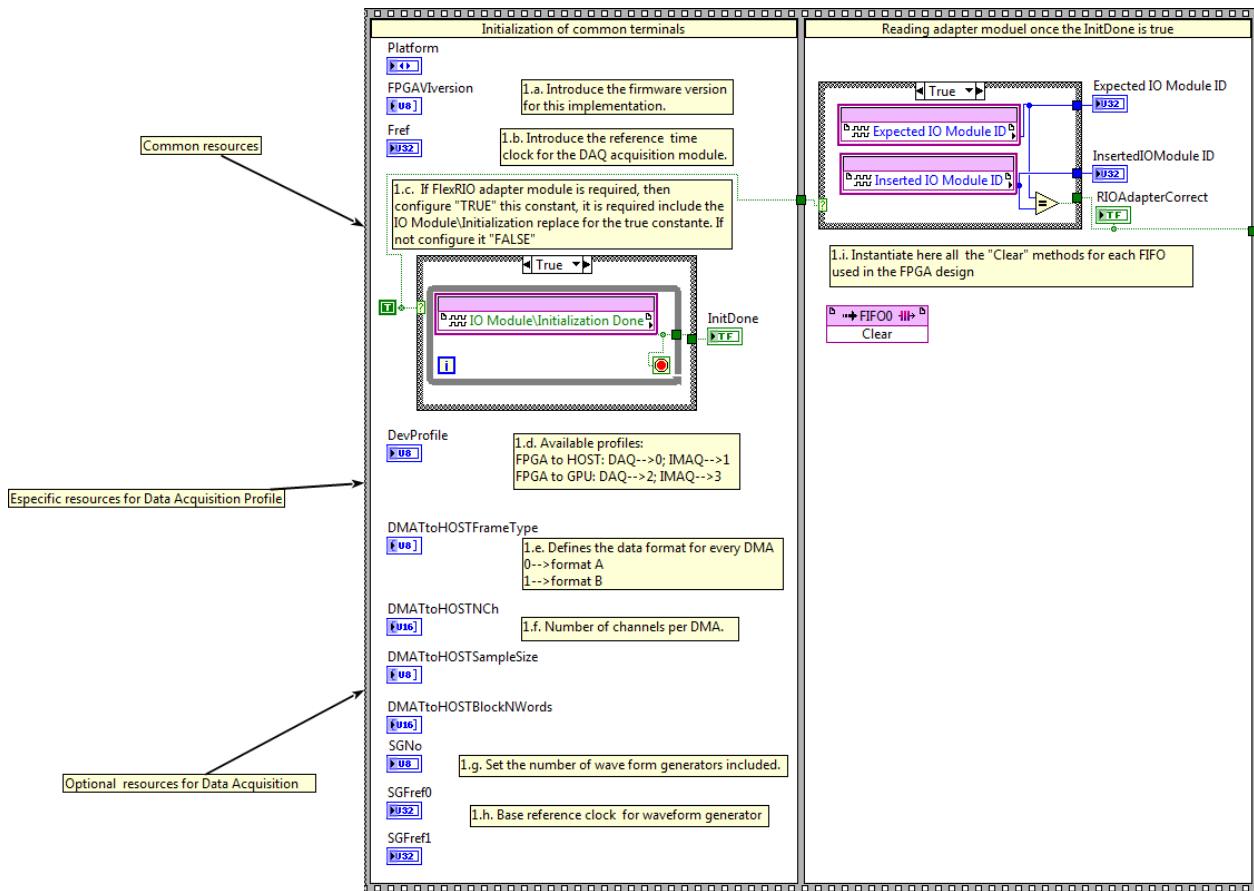


Fig. 87: Initialization.



**Warning.** The values read by software layer in the initialization phase are read from terminals in the control panel.

Once the adapter module is initialized, the user can add the code supporting the DMA acquisition. This is represented in Fig. 88 and can be replicated as many times as DMAs are in use.

Fig. 89 displays the details of what have to be modified when adding a new DMA.

- Constant value to set the number of the DMA that the DAQ DMA module corresponds to.
- Select the number of channels required.
- Marked in blue, you have to select the corresponding intermediate internal FIFO. The developer should add as many intermediate FIFOs as DMAs have been included in the design. The name chosen of the intermediate FIFOs is a developer decision, but FIFO<N> is recommended.
- Under the Green Square, you have to choose the DMA. This one has to be compliant with the IRIO design rules.

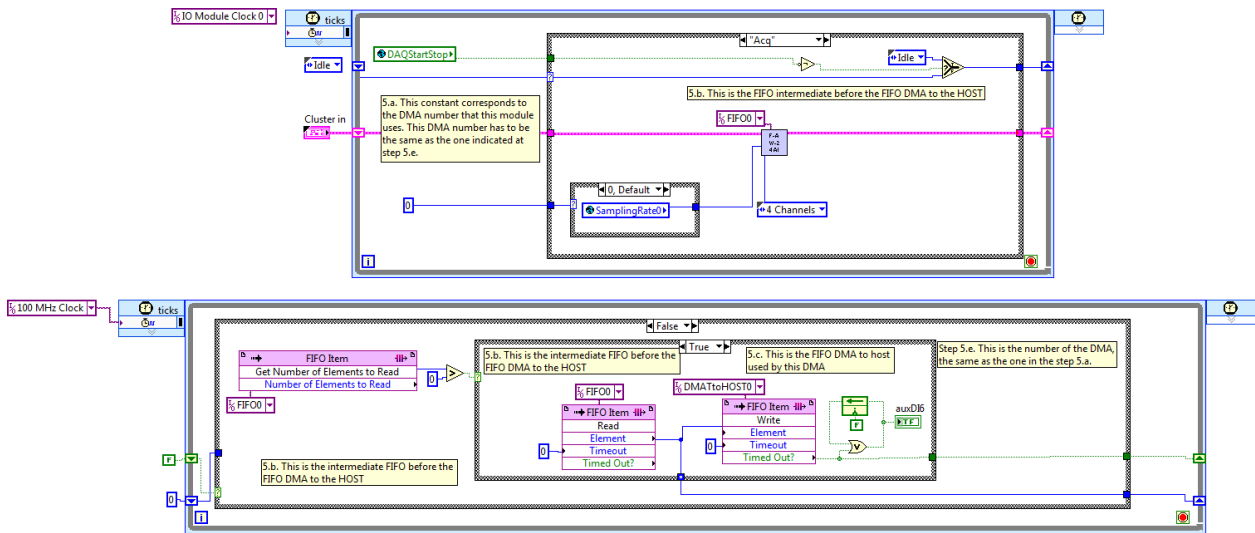


Fig. 88: Implementation of the DMA for FlexRIO platform.

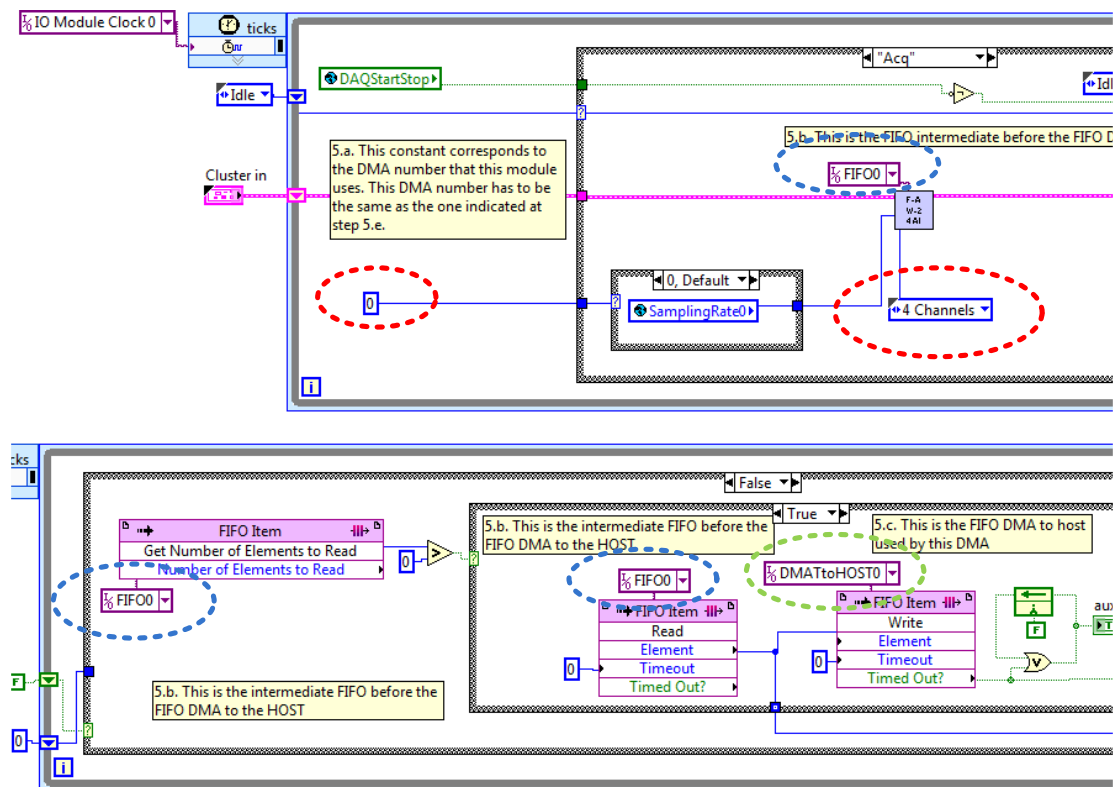


Fig. 89: Detail of DMA implementation in the FPGA.

If more than one DMA is required, then “DMAtoHOSTSamplingRate<N>” controls have to be added into the timed loop that contains global variables, with the corresponding SamplingRate<N> global variable connected to (see Fig. 90).

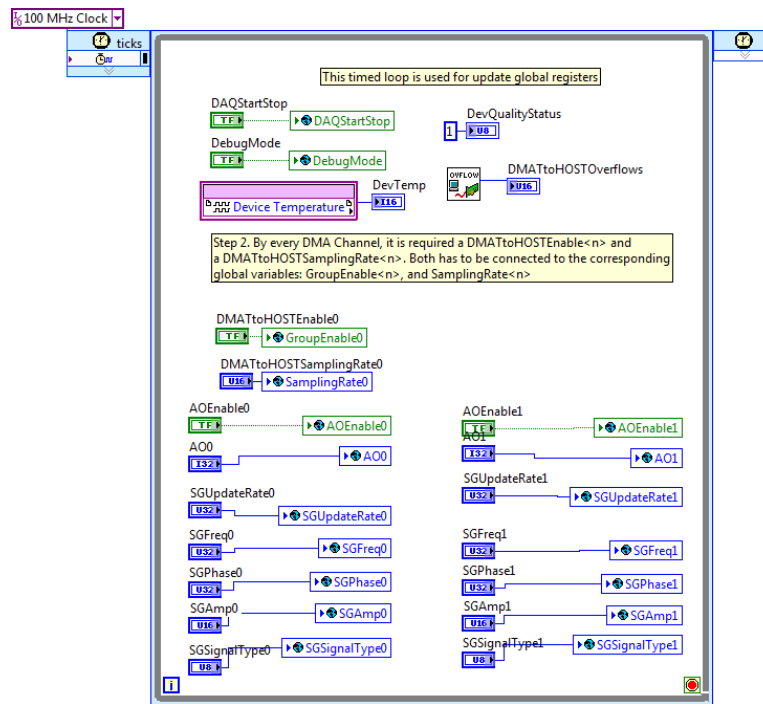


Fig. 90: SamplingRate terminals

In the global variables timed loop there are the global references to the controls of the signal generators. If these ones are not going to be used, they can be removed. You can add as many signal generators as the design requires and the FPGA resources permit. The signal generator template can be taken from its VI file, executing a “copy and paste” into the main VI. After that, the controls, and indicators, should be changed by the corresponding global variables (Fig. 91).

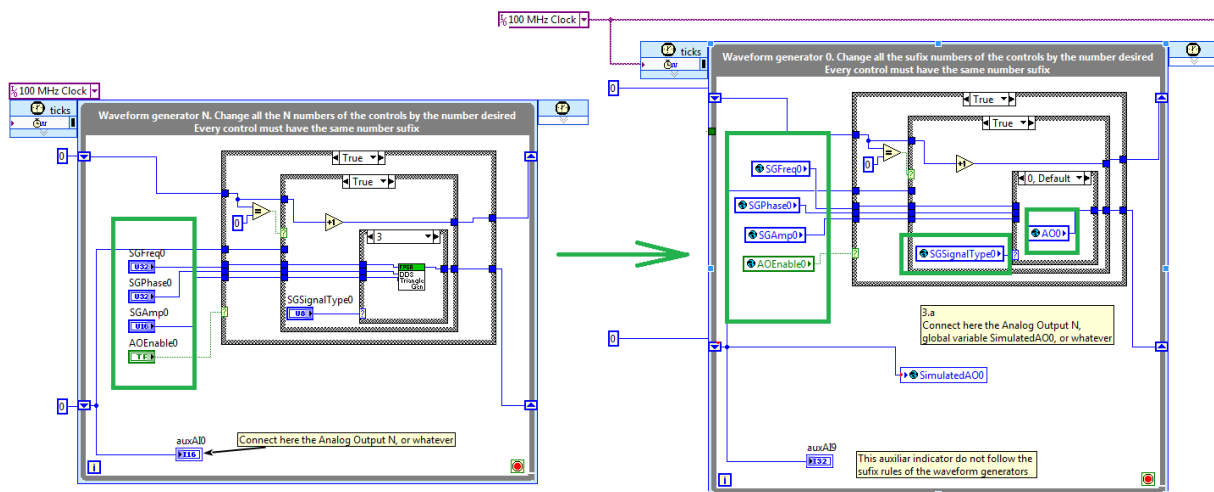


Fig. 91: Inserting the signal generator code in the VI.

Finally, the template has two loops, with a basic example using auxiliary analog and digital inputs and outputs. The developer can add or remove under the requirements of the design, and implement any logic or algorithm between the input/output terminals, as the example below shows (see Fig. 92).

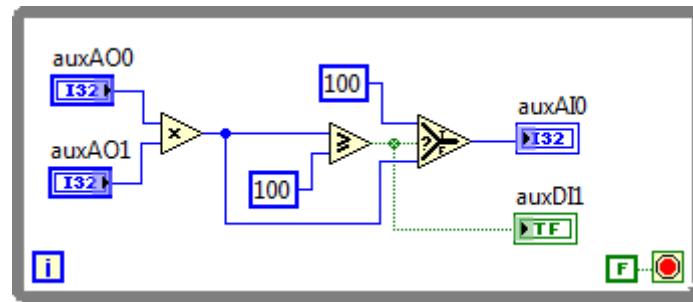


Fig. 92: Example of code for using auxiliary terminals

## 9 NI FPGA INTERFACE C API GENERATOR

### 9.1 Executing the application

In the LabVIEW project window, the user must select the VI with the FPGA application and right click on it, in the pop-up menu, the user selects “Launch C API Generator”, a window immediately appears, the user must select the name of the bitfile with the design of the FPGA and an output directory for the headers file that will be generated with this application. Finally, the user must click on the Generate button.

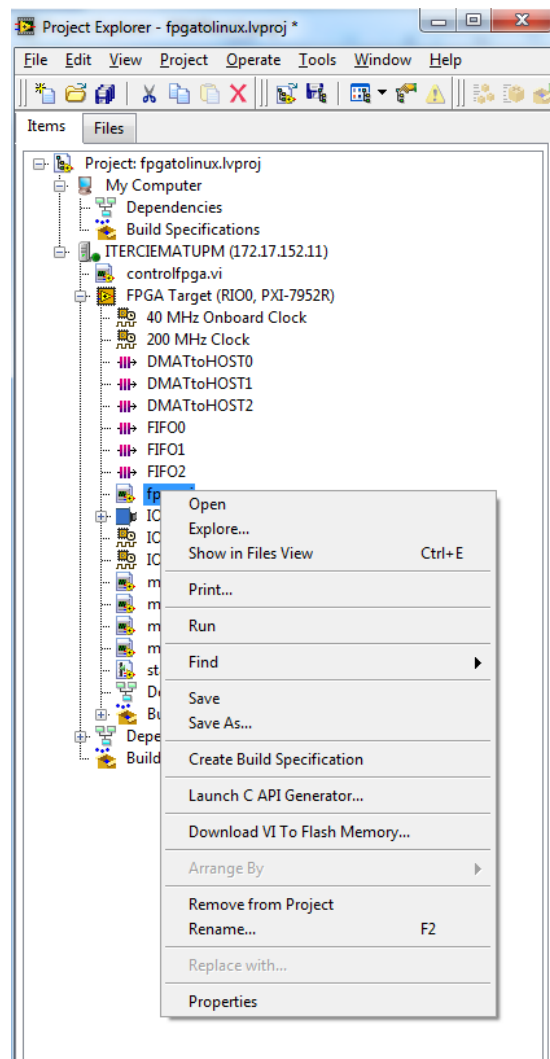
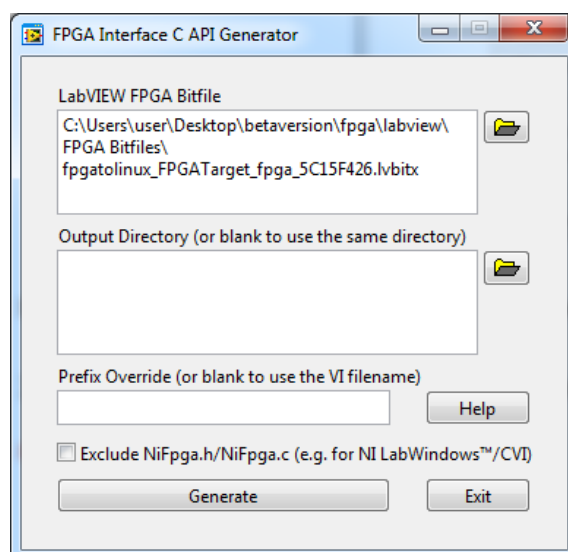


Fig. 93: Launching the C API Generator application.



**Fig. 94: Application C API generator in LabVIEW FPGA.**

This program outputs four files, but only 2 are necessary. The first file is a renamed version of the bitfile with the name `NiFpga_”projectName”.lvbitx`, the second one is a header file with the name `NiFpga_”projectName”.h`, which includes the identification of the different control and indicators and the resources used in the FPGA design.



**[NOTE]:** Every VI design must be compiled using the LabVIEW/FPGA environment. The lvbitx file obtained must be used with the C API generator that outputs the specific header file.

The header file contains many definitions with the different enumerated data types and their assignments, which will be used by the NI-RIO EPICS support driver to obtain the information for the FPGA.



**[NOTE]:** The enumerated data types are identified using the following criteria: `NiFpga_”vi name”_TerminalType_& DataType_Terminal_Name`. The terminal data type will be Control or Indicator followed by the terminal data type: Boolean, integer, etc.

## 9.2 Header file generated.

The NI FPGA INTERFACE C API Generator outputs a header file that will be used by the EPICS applications. This is an example of this header file:



```

/*
 * Generated with the FPGA Interface C API Generator 13.0.0
 * for NI-RIO 13.0.0 or later.
 */

#ifndef __NiFpga_cRIODAQ_h__
#define __NiFpga_cRIODAQ_h__

#ifndef NiFpga_Version
#define NiFpga_Version 1300
#endif

#include "NiFpga.h"

/**
 * The filename of the FPGA bitfile.
 *
 * This is a #define to allow for string literal concatenation. For example:
 *
 * static const char* const Bitfile = "C:\\\\" NiFpga_cRIODAQ_Bitfile;
 */
#define NiFpga_cRIODAQ_Bitfile "NiFpga_cRIODAQ.lvbitx"

/**
 * The signature of the FPGA bitfile.
 */
static const char* const NiFpga_cRIODAQ_Signature =
"8205846622339C4D9F1D2927E16981C5";

typedef enum
{
    NiFpga_cRIODAQ_IndicatorBool_InitDone = 0x8152,
    NiFpga_cRIODAQ_IndicatorBool_cRIOModulesOK = 0x8162,
} NiFpga_cRIODAQ_IndicatorBool;

typedef enum
{
    NiFpga_cRIODAQ_IndicatorU8_DevProfile = 0x815A,
    NiFpga_cRIODAQ_IndicatorU8_DevQualityStatus = 0x816E,
    NiFpga_cRIODAQ_IndicatorU8_Platform = 0x817A,
    NiFpga_cRIODAQ_IndicatorU8_SGNo = 0x810E,
} NiFpga_cRIODAQ_IndicatorU8;

typedef enum
{
    NiFpga_cRIODAQ_IndicatorI16_DevTemp = 0x815E,
} NiFpga_cRIODAQ_IndicatorI16;

typedef enum
{
    NiFpga_cRIODAQ_IndicatorU16_state = 0x8192,
} NiFpga_cRIODAQ_IndicatorU16;

typedef enum
{
    NiFpga_cRIODAQ_IndicatorI32_AI0 = 0x8184,
    NiFpga_cRIODAQ_IndicatorI32_AI1 = 0x8188,
    NiFpga_cRIODAQ_IndicatorI32_auxAI0 = 0x8180,

```

```
} NiFpga_cRIODAQ_IndicatorI32;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_IndicatorU32_Fref = 0x8154,
} NiFpga_cRIODAQ_IndicatorU32;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_ControlBool_AOEnable0 = 0x8116,
    NiFpga_cRIODAQ_ControlBool_AOEnable1 = 0x8132,
    NiFpga_cRIODAQ_ControlBool_DAQStartStop = 0x8176,
    NiFpga_cRIODAQ_ControlBool_DMATtoHOSTEnable0 = 0x8146,
    NiFpga_cRIODAQ_ControlBool_DebugMode = 0x8172,
} NiFpga_cRIODAQ_ControlBool;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_ControlU8_SGSignalType0 = 0x811A,
} NiFpga_cRIODAQ_ControlU8;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_ControlU16_DMATtoHOSTOverflows = 0x814A,
    NiFpga_cRIODAQ_ControlU16_DMATtoHOSTSamplingRate0 = 0x8142,
    NiFpga_cRIODAQ_ControlU16_SGAmp0 = 0x812A,
} NiFpga_cRIODAQ_ControlU16;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_ControlI32_A00 = 0x811C,
    NiFpga_cRIODAQ_ControlI32_A01 = 0x8110,
    NiFpga_cRIODAQ_ControlI32_auxA00 = 0x817C,
} NiFpga_cRIODAQ_ControlI32;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_ControlU32_LoopuSec = 0x8194,
    NiFpga_cRIODAQ_ControlU32_SGFreq0 = 0x8124,
    NiFpga_cRIODAQ_ControlU32_SGPhase0 = 0x8120,
    NiFpga_cRIODAQ_ControlU32_SGUpdateRate0 = 0x812C,
    NiFpga_cRIODAQ_ControlU32_TabControl = 0x818C,
} NiFpga_cRIODAQ_ControlU32;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_IndicatorArrayU8_DMATtoHOSTFrameType = 0x813A,
    NiFpga_cRIODAQ_IndicatorArrayU8_DMATtoHOSTSampleSize = 0x813E,
    NiFpga_cRIODAQ_IndicatorArrayU8_FPGAVIversion = 0x816A,
} NiFpga_cRIODAQ_IndicatorArrayU8;
```

```
typedef enum
```

```
{
    NiFpga_cRIODAQ_IndicatorArrayU8Size_DMATtoHOSTFrameType = 1,
    NiFpga_cRIODAQ_IndicatorArrayU8Size_DMATtoHOSTSampleSize = 1,
    NiFpga_cRIODAQ_IndicatorArrayU8Size_FPGAVIversion = 2,
} NiFpga_cRIODAQ_IndicatorArrayU8Size;
```

```

typedef enum
{
    NiFpga_cRIODAQ_IndicatorArrayU16_DMATtoHOSTBlockNWords = 0x814E,

    NiFpga_cRIODAQ_IndicatorArrayU16_DMATtoHOSTNCh = 0x8136,
    NiFpga_cRIODAQ_IndicatorArrayU16_InsertedIOModulesID = 0x8164,
} NiFpga_cRIODAQ_IndicatorArrayU16;

typedef enum
{
    NiFpga_cRIODAQ_IndicatorArrayU16Size_DMATtoHOSTBlockNWords = 1,
    NiFpga_cRIODAQ_IndicatorArrayU16Size_DMATtoHOSTNCh = 1,
    NiFpga_cRIODAQ_IndicatorArrayU16Size_InsertedIOModulesID = 16,
} NiFpga_cRIODAQ_IndicatorArrayU16Size;

typedef enum
{
    NiFpga_cRIODAQ_TargetToHostFifoU64_DMATtoHOST0 = 0,
} NiFpga_cRIODAQ_TargetToHostFifoU64;

#endif

```