ELVIS III Top Board Developer’s Manual

# Overview

This manual helps users of the National Instruments ELVIS III develop a custom application top board. The manual includes information on licensing, mechanical design, electrical design, basic software interfacing, and courseware support. For information on the use of ELVIS III or administering and provisioning ELVIS III in a laboratory setting refer to the product documentation.

Contents

[Overview 1](#_Toc502915906)

[Licensing 5](#_Toc502915907)

[Mechanical Features 5](#_Toc502915908)

[Edge Connector 6](#_Toc502915909)

[Mounting Holes 6](#_Toc502915910)

[Engagement Holes 6](#_Toc502915911)

[Removal Hole 6](#_Toc502915912)

[Rubber Feet 6](#_Toc502915913)

[Detailed Dimensions 6](#_Toc502915914)

[Top Board Appearance and Branding 9](#_Toc502915915)

[Thermal Considerations 10](#_Toc502915916)

[Edge Connector Signals 10](#_Toc502915917)

[Pinout 10](#_Toc502915918)

[Signal Descriptions 12](#_Toc502915919)

[Multisim and Ultiboard Design Templates 12](#_Toc502915920)

[Hardware Capabilities 13](#_Toc502915921)

[Analog Input 13](#_Toc502915922)

[Analog Output 14](#_Toc502915923)

[Digital Input/Output 14](#_Toc502915924)

[USB Host Port 15](#_Toc502915925)

[Board Identification 15](#_Toc502915926)

[Timing Diagram 15](#_Toc502915927)

[Required Read Support 16](#_Toc502915928)

[Optional Write Support 17](#_Toc502915929)

[Example Implementation 18](#_Toc502915930)

[Board ID Contents 19](#_Toc502915931)

[Board ID CRC Format 20](#_Toc502915932)

[Top Board Software Access 20](#_Toc502915933)

[LabVIEW FPGA API Support 20](#_Toc502915934)

[Analog Input 20](#_Toc502915935)

[IO Node 20](#_Toc502915936)

[Method Node to Change the AI Range 20](#_Toc502915937)

[Analog Output 20](#_Toc502915938)

[Digital Input and Output 20](#_Toc502915939)

[LabVIEW API Support 21](#_Toc502915940)

[LabVIEW APIs for Top Board Access 21](#_Toc502915941)

[Use LabVIEW APIs with Custom FPGA Personality 21](#_Toc502915942)

[Situation 1: The peripherals supported by the custom FPGA personality is the subset of the default shipping FPGA personality 21](#_Toc502915943)

[Situation 2: The peripheral supported by the custom FPGA personality is the subset of default shipping FPGA personality, but with channel(s) added or removed. 21](#_Toc502915944)

[Situation 3: The custom FPGA personality supports a new peripheral which is not supported by the default shipping FPGA personality 22](#_Toc502915945)

[Migrating NI ELVIS II/II+ Code 22](#_Toc502915946)

[Migrating myRIO Code 23](#_Toc502915947)

[Instrument Launcher 26](#_Toc502915948)

[Device Tab 26](#_Toc502915949)

[Instruments Tab 26](#_Toc502915950)

[Resources Tab 26](#_Toc502915951)

[Teaching Resources Development 27](#_Toc502915952)

[Teaching Resources Quality Guidelines 27](#_Toc502915953)

[Templates for Creating Offline Versions of Curriculum 32](#_Toc502915954)

[Reference 32](#_Toc502915955)

[Default Shipping FPGA Personality 32](#_Toc502915956)

[System Control / Function Select 32](#_Toc502915957)

[Function Select Registers (SYS.SELECTx) 32](#_Toc502915958)

[Onboard Device Registers 34](#_Toc502915959)

[LED 34](#_Toc502915960)

[Button 34](#_Toc502915961)

[AI/AO 35](#_Toc502915962)

[Analog Counter Register (AI.x.CNT) 35](#_Toc502915963)

[Analog Configuration Registers (AI.X.CNFG) 35](#_Toc502915964)

[Analog Divisor Registers (AI.X.CNTR, AO.X.DMA\_CNTR) 35](#_Toc502915965)

[Analog Input DMA Enable Registers (AI.X.DMA\_ENA) 36](#_Toc502915966)

[Analog Ouptput DMA Enable Registers (AO.X.DMA\_ENA) 36](#_Toc502915967)

[Analog DMA IDLE Registers (AI.X.DMA\_IDL, AO.X.DMA\_IDL) 36](#_Toc502915968)

[Analog Value Registers (AI.X.VAL, AO.X.VAL) 36](#_Toc502915969)

[DIO 37](#_Toc502915970)

[Data Direction Registers (DIO.xx.DIR) 37](#_Toc502915971)

[Pin Input Registers (DIO.xx.IN) 37](#_Toc502915972)

[Pin Output Registers (DIO.xx.OUT) 37](#_Toc502915973)

[Digital Divisor Registers (DI.X.DMA\_CNTR, DO.X.DMA\_CNTR) 37](#_Toc502915974)

[Digital Input DMA Enable Registers (DI.X.DMA\_ENA) 38](#_Toc502915975)

[Digital Output DMA Enable Registers (DO.X.DMA\_ENA) 38](#_Toc502915976)

[Digital DMA Idle Registers (DI.X.DMA\_IDL, DO.X.DMA\_IDL) 38](#_Toc502915977)

[PWM 38](#_Toc502915978)

[PWM Configuration Registers (PWM.x.CNFG) 38](#_Toc502915979)

[PWM Clock Select Registers (PWM.x.CS) 39](#_Toc502915980)

[PWM Maximum Count Registers (PWM.x.MAX) 39](#_Toc502915981)

[PWM Compare Registers (PWM.x.CMP) 40](#_Toc502915982)

[PWM Counter Registers (PWM.x.CNTR) 40](#_Toc502915983)

[PWM Frequency Generation 40](#_Toc502915984)

[SPI Master 41](#_Toc502915985)

[SPI Configuration Registers (SPI.x.CNFG) 41](#_Toc502915986)

[SPI Counter Registers (SPI.x.CNT) 44](#_Toc502915987)

[SPI Execute Registers (SPI.x.GO) 44](#_Toc502915988)

[SPI Status Registers (SPI.x.STAT) 44](#_Toc502915989)

[SPI Data Out Registers (SPI.x.DATO) 45](#_Toc502915990)

[SPI Data In Registers (SPI.x.DATI) 45](#_Toc502915991)

[SPI Frequency Generation 45](#_Toc502915992)

[Encoder 46](#_Toc502915993)

[Encoder Configuration Registers (ENC.x.CNFG) 46](#_Toc502915994)

[Encoder Status Registers (ENC.x.STAT) 47](#_Toc502915995)

[Encoder Counter Value Registers (ENC.x.CNTR) 48](#_Toc502915996)

[I2C 48](#_Toc502915997)

[I2C Configuration Registers (I2C.x.CNFG) 48](#_Toc502915998)

[I2C Slave Address Registers (I2C.x.ADDR) 49](#_Toc502915999)

[I2C Counter Registers (I2C.x.CNTR) 49](#_Toc502916000)

[I2C Data Out Registers (I2C.x.DATO) 50](#_Toc502916001)

[I2C Data In Registers (I2C.x.DATI) 50](#_Toc502916002)

[I2C Status Registers (I2C.x.STAT) 50](#_Toc502916003)

[I2C Control Registers (I2C.x.CNTL) 51](#_Toc502916004)

[I2C Execute Registers (I2C.x.GO) 55](#_Toc502916005)

[Timer Interrupt 55](#_Toc502916006)

[Timer Read Register (IRQ.TIMER.READ) 55](#_Toc502916007)

[Timer Write Register (IRQ.TIMER.WRITE) 56](#_Toc502916008)

[Timer Set Time Register (IRQ.TIMER.SETTIME) 56](#_Toc502916009)

[Analog Input Interrupt 56](#_Toc502916010)

[Analog IRQ Threshold Register (IRQ.AI\_xx.THRESHOLD) 56](#_Toc502916011)

[Analog IRQ Hysteresis Register (IRQ.AI\_xx.HYSTERESIS) 56](#_Toc502916012)

[Analog IRQ Configuration Register (IRQ.AI\_ XX.CNFG) 56](#_Toc502916013)

[Analog IRQ Number Register (IRQ.AI\_xx.NO) 57](#_Toc502916014)

[Digital Input Interrupt 57](#_Toc502916015)

[Digital Enabling Register (IRQ.DIO\_xx.ENA) 57](#_Toc502916016)

[Digital Rising Register (IRQ.DIO\_xx.RISE) 58](#_Toc502916017)

[Digital Falling Register (IRQ.DIO\_xx.FALL) 58](#_Toc502916018)

[Digital IRQ Number Register (IRQ.DIO\_xx.NO) 59](#_Toc502916019)

[Digital Count Register (IRQ.DIO\_A\_0.CNT) 59](#_Toc502916020)

[Button Interrupt 59](#_Toc502916021)

[Button Enabling Register (IRQ.DI\_BTN.ENA) 59](#_Toc502916022)

[Button Rising Register (IRQ.DI\_BTN.RISE) 59](#_Toc502916023)

[Button Falling Register (IRQ.DI\_BTN.FALL) 59](#_Toc502916024)

[Button IRQ Number Register (IRQ.DI\_BTN.NO) 60](#_Toc502916025)

[Button Count Register (IRQ.DI\_BTN.CNT) 60](#_Toc502916026)

# Licensing

Top boards that are developed for resale, or to be distributed to third parties, or that are not otherwise intended for internal use or evaluation purposes are expected to apply for licensing from National Instruments. To be eligible for licensing, top boards should meet all guidelines and requirements outlined in this document. Benefits of licensing include the right to include certain logos and markings on the product, as well as an improved software experience with enhanced top board identification. For more information on the licensing program, please contact the NI Academic Product Marketing team.

There are no licensing requirements for top boards developed for internal use or evaluation purposes only. An example of internal use would be the development of a custom top board by a university professor for use in classes taught at that university. An example of evaluation purposes would be the rapid prototyping of a custom top board by a National Instruments Alliance Partner that is then provided to a potential client on a temporary basis as a pre-sales tool.

# Mechanical Features

A basic top board consists of a single PCB with integrated mounting features, an edge connector for interfacing with the ELVIS III signals, and a set of rubber feet.

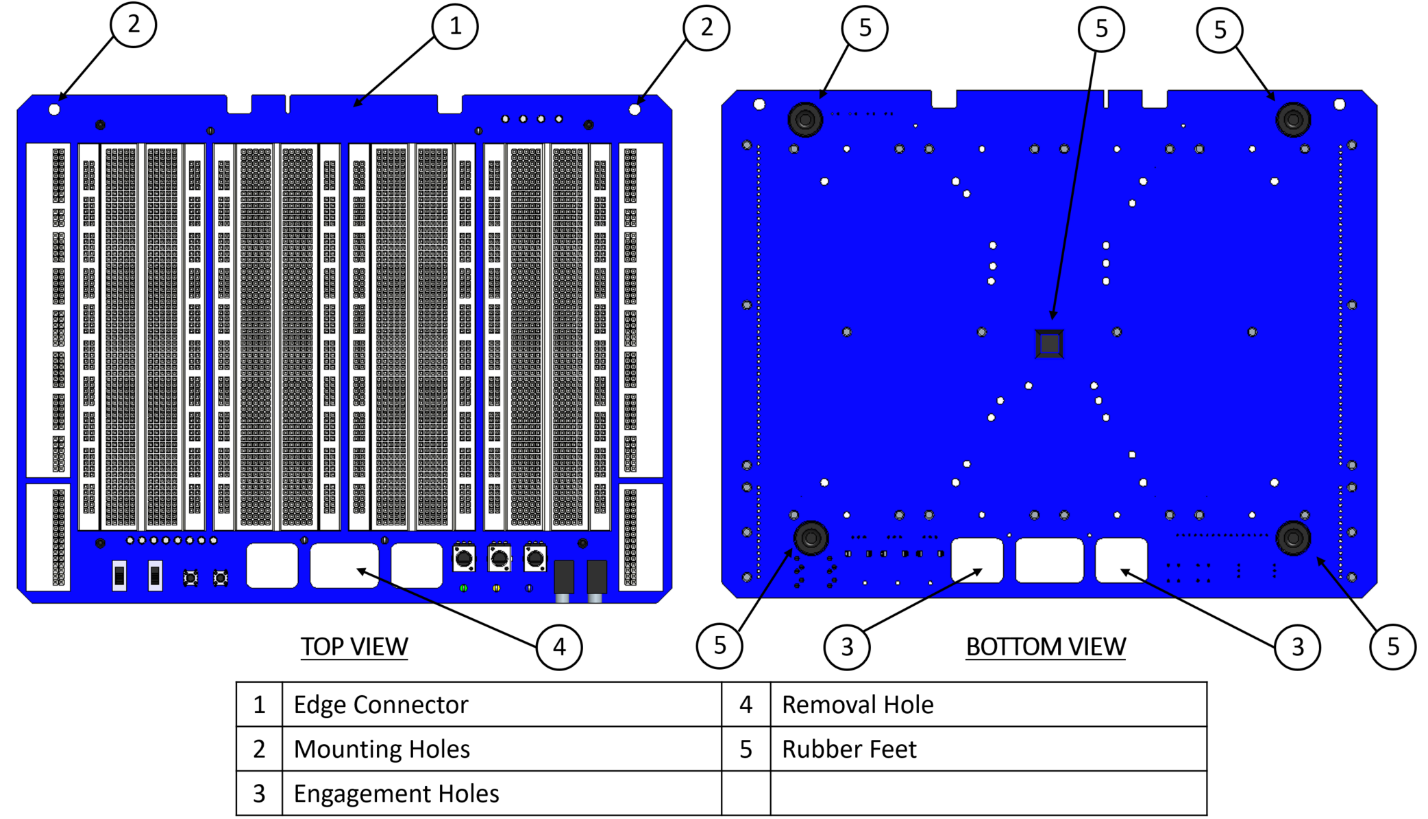


Figure : Basic Top Board Features

## Edge Connector

The edge connector provides access to the RIO based IO on ELVIS III. This includes the analog input, analog output, digital I/O, fixed power supplies, board identification features, and a USB 2.0 host port. The top board connector on ELVIS III is a standard 32-bit PCI connector, although it does not use PCI signaling nor pinout. Benefits of using a PCI connector include ease of board insertion, mechanical robustness, electrical performance, low cost, and simple edge connector design on the top board. The top board edge connector uses industry standard hard gold fingers with a beveled leading edge.

## Mounting Holes

The two mounting holes are located towards the rear of the top board and work in conjunction with the front engagement holes to firmly attach the top board to the ELVIS III. This provides a chassis ground connection through the enclosure metal, as well as ensuring a robust electrical connection between the edge connector on the top board and the PCI connector on ELVIS III. It also provides a more mechanically stable solution for top boards that include moving parts with notable mass. ELVIS III incorporates threaded holes at these locations, and an M4x6 screw should be used to attach the top board.

## Engagement Holes

The two engagement holes towards the front of the top board are used to help align the top board during insertion and help retain the top board during use. When installing the top board, it should be placed on the top surface such that the engagement hooks on ELVIS III are protruding through the engagement holes on the top board. Then the top board should be pushed straight back into the top board connector. The engagement hooks will then serve to hold down the front of the top board and prevent it from being lifted vertically, which could damage the top board connector.

## Removal Hole

The removal hole is used to aid in the safe removal of the top board from the ELVIS III. Users typically place their index finger in the hole, their thumb on the front of ELVIS III, and pull the top board forward. Having a centrally located hole minimizes any side to side rocking during removal, which could damage the top board connector.

## Rubber Feet

For maximum stability and ease of use five rubber feet should be used, one for each corner and one in the center. The feet in the corners keep the top board level during installation, which helps align and protect the top board connector from damage. The front two feet further stabilize the top board after installation, as the front would otherwise only be supported near the center by the engagement hooks. The foot in the center helps prevent the top board from bowing during use. It is important to note that the drawings indicate the recommended locations for the rubber feet. It is recognized that some top boards may have other features that conflict with the installation of the rubber feet in these exact locations, in which case they should be located as close as practically possible. Deviating notably from the recommended locations may cause compatibility issues with future ELVIS product offerings.

## Detailed Dimensions

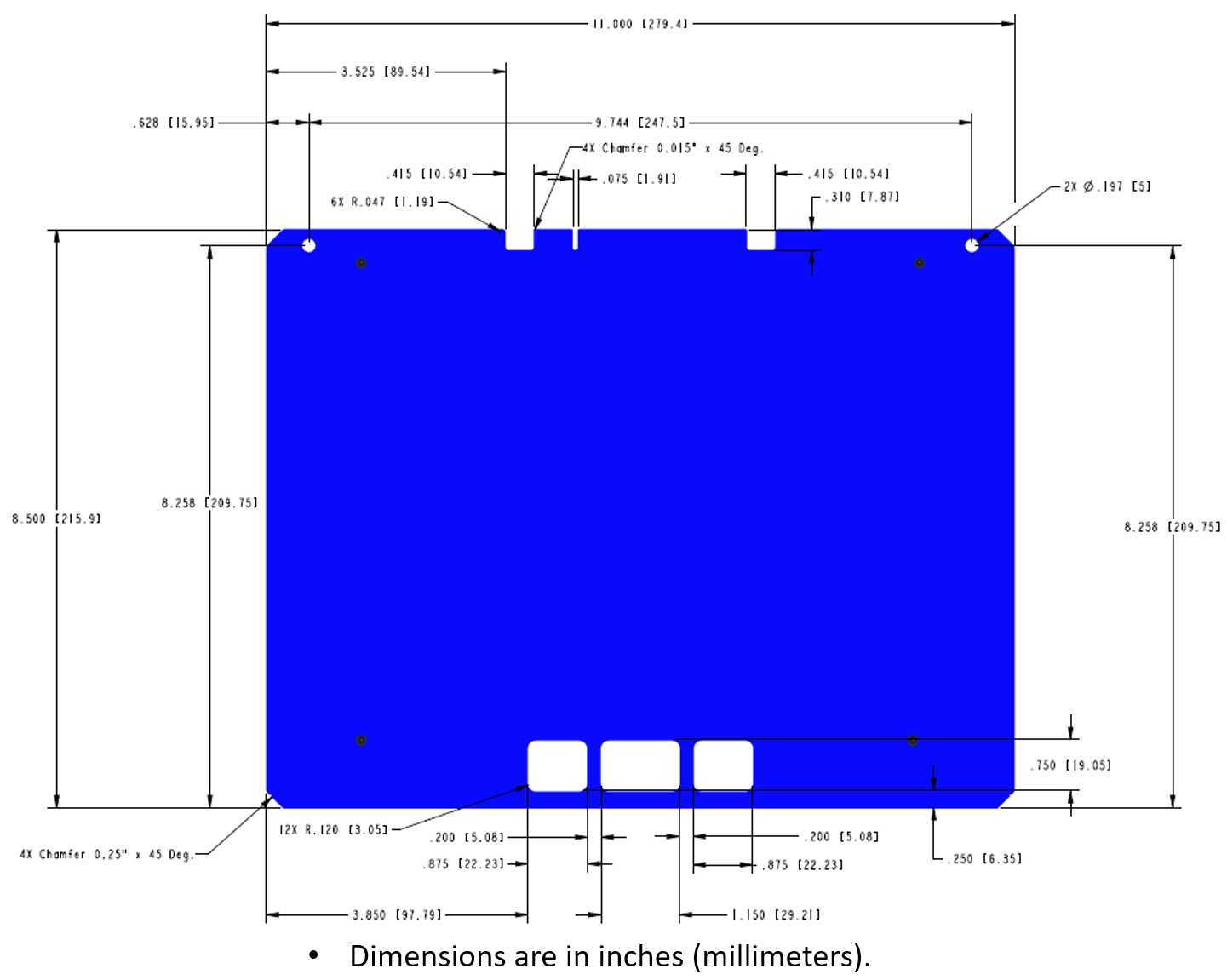


Figure : Primary Side PCB Dimensions

NOTE: An ELVIS III top board does NOT have the same board outline as ELVIS II/II+. There is a keying feature by the gold fingers (wider notches) to support potential future ELVIS variants. Also, the engagement holes at the front of the PCB have been reduced in size. Using the ELVIS II/II+ outline for an ELVIS III top board may result in physical incompatibility with future ELVIS lab stations.

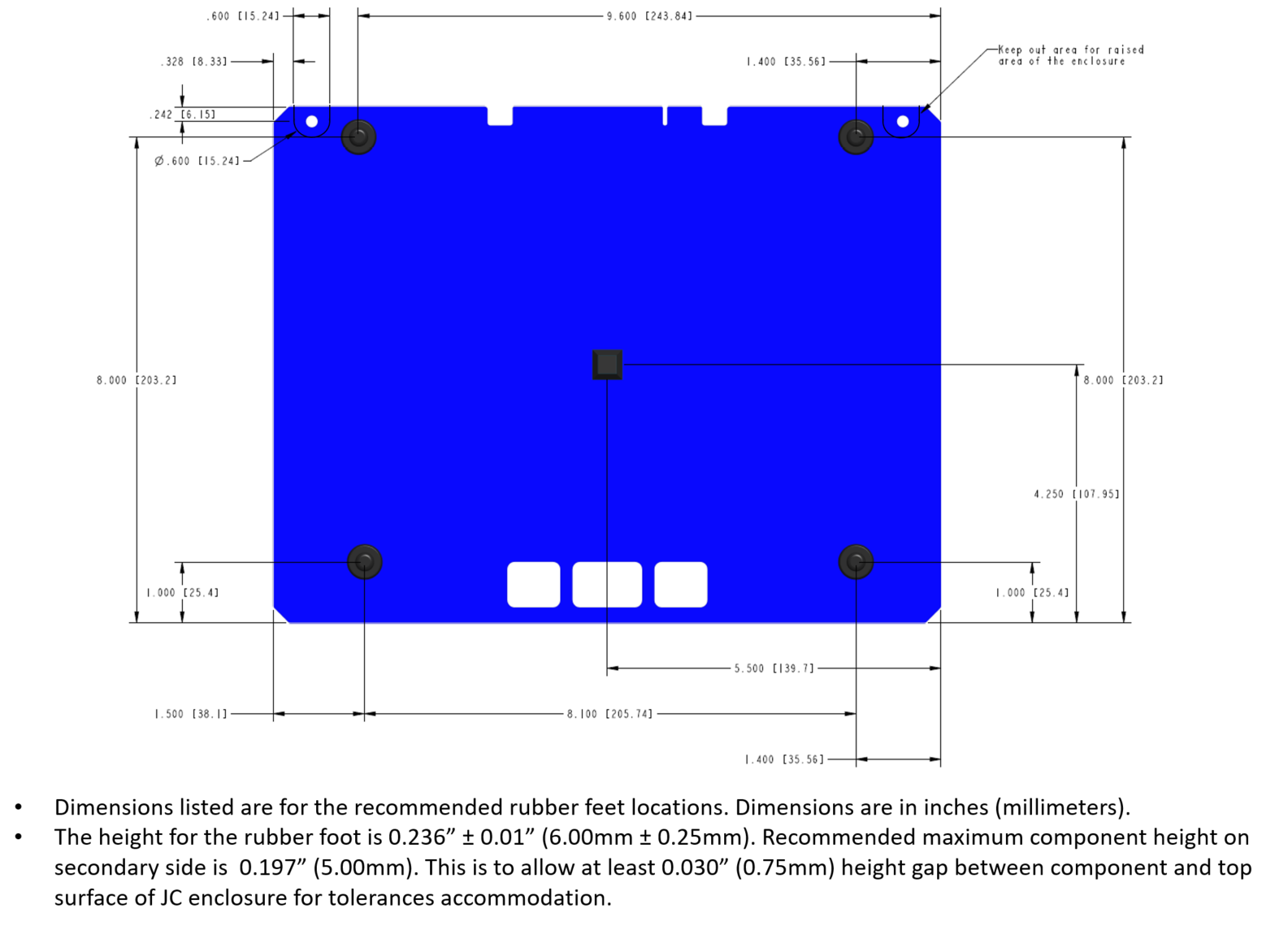


Figure : Secondary Side PCB Dimensions

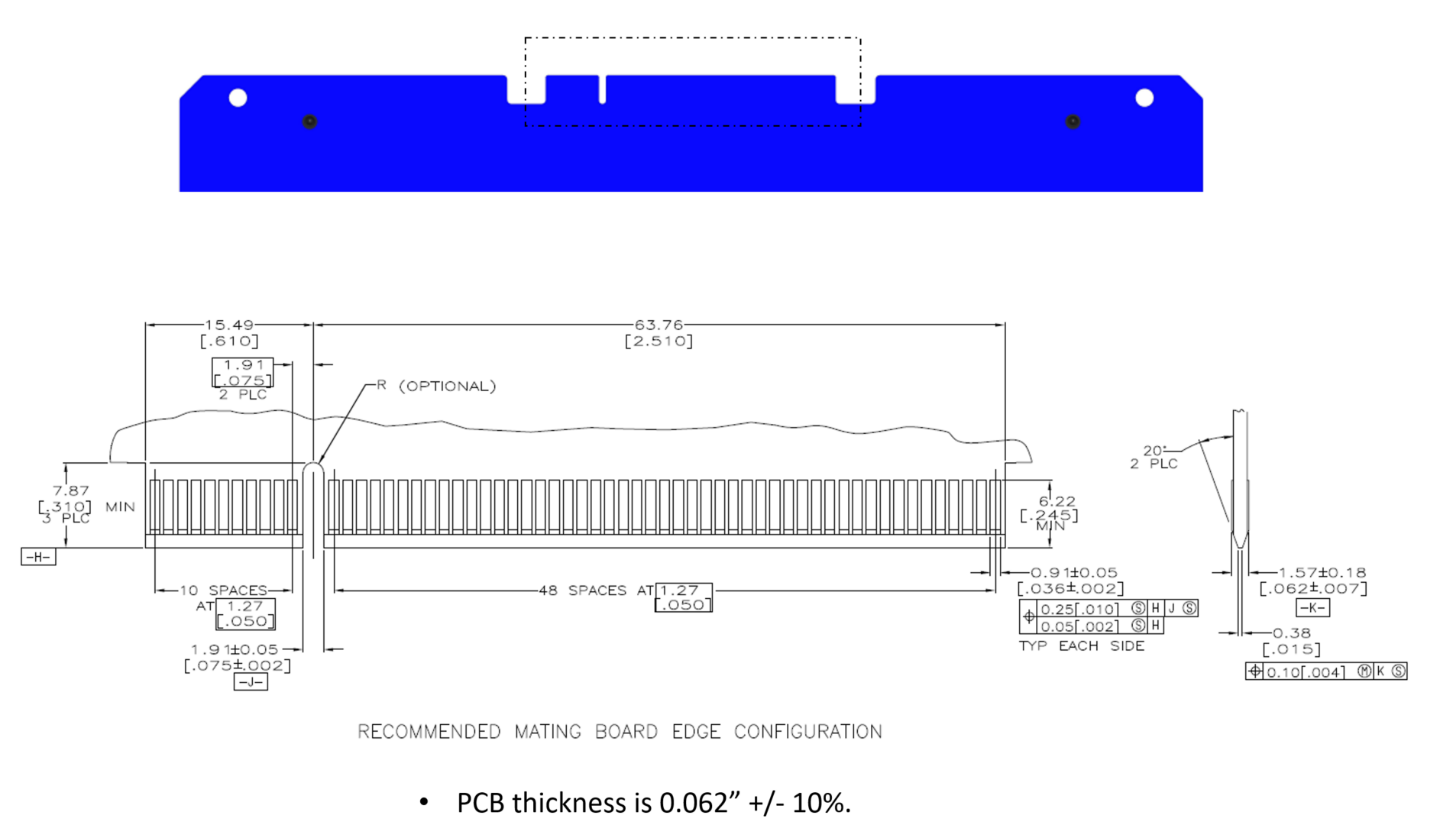


Figure : PCB Cross Section and Edge Connector Dimensions

Components, shields, and other mechanical parts should not be located such that they block user view of the edge connector and adjacent keys during installation of the application board. Any movable components that may extend over this region during operation (such as a robotic arm) should be at least 10mm above the PCB surface.

Consideration should also be given to any connectors or features located along the sides of the application board that may obstruct features on ELVIS III. For example, right angle connectors located on the left hand side of the application board should be located to minimize any overlap of the mating cables with the LEDs in the ELVIS III enclosure.

To accelerate the design process and reduce the risk of errors, a sample project is provided for Multisim and Ultiboard. These electronic design files include a basic schematic and layout for a generic top board, eliminating manual entry of the mechanical dimensions and connector information. The Multisim and Ultiboard templates are attached to this manual and will be provided natively in the software.

## Top Board Appearance and Branding

Third-party partners creating top boards for ELVIS III are required to adhere to consistent branding guidelines that dictate the general appearance of the board.

All boards from a single partner should follow the same color palette; which includes a main color for the silkscreen and an accent color. Before developing a product based on a certain branding scheme or colors, a partner must get approval from NI on their branding guidelines.

In case the partner’s brand does not have a unique color palette or branding requirements, the boards should then follow the NI Compass 2.0 color guidelines (provided separately, as required).

The board should include the partner logo on the silkscreen which will be on the bottom right-hand side of the board so that it is diagonal from the National Instruments logo on ELVIS III.

The product name should be agreed approved by the NI Marketing Team. Partners should include a vanity URL on each board, provided by NI, which will be ni.com/teach/<Name > where <Name> is a short identifier that will drive customers directly to the related teaching resources on ni.com/teach. This name will be defined in collaboration between NI and the partner. ELVIS III application boards will not have the National Instruments logo.

# Thermal Considerations

Top boards should be designed to direct any significant power dissipation upwards and outwards to avoid overheating the ELVIS III. It is a good design practice to thermally isolate high power dissipating devices (like motors, multi-watt load resistors, etc.) from the top board surface. One common approach is to use standoffs or simple brackets to elevate the devices slightly above the board, as the standoffs and brackets are normally poor conductors of heat back into the board. If higher power components (such as H-bridges, high current linear regulators, etc.) must be located on the top board itself, better thermal performance can generally be obtained by locating them on the top surface.

The thermal design of ELVIS III allows for notable thermal margins for typical applications and top boards. For a well-designed high-power top board, thermal margins can still be met even if the application is simultaneously exercising all ELVIS III resources at their documented limits. In addition to following best design practices, a conservative validation of thermal margins can be performed by measuring the case temperature of the ELVIS III. This should be done with the top board in place while operating the system under expected worst case conditions for that application. Testing must take into account room temperature, and either perform the testing at the maximum ambient rating of the product or extrapolate the results to that ambient temperature. The case temperature should be measured across the entire top surface underneath the top board. The case temperature should not exceed [need to determine this value from Rev B analysis and testing].

# Edge Connector Signals

The RIO based IO is provided through the edge connector located towards the rear of the top board. This includes the analog input, analog output, digital I/O, fixed power supplies, board identification features, and a USB 2.0 host port.

## Pinout

|  |  |  |
| --- | --- | --- |
| Pin # | Side A (Secondary) | Side B (Primary) |
| 1 | +15V | -15V |
| 2 | +15V | -15V |
| 3 | +5V | DGND |
| 4 | +5V | DGND |
| 5 | +5V | DGND |
| 6 | DGND | DGND |
| 7 | B/DIO0 | B/DIO1 |
| 8 | B/DIO2 | B/DIO3 |
| 9 | B/DIO4 | B/DIO5 |
| 10 | B/DIO6 | B/DIO7 |
| 11 | DGND | DGND |
| 12 | USB D+ | USB\_VBUS |
| 13 | USB D- | DGND |
| 14 | B/DIO8 | B/DIO9 |
| 15 | B/DIO10 | B/DIO11 |
| 16 | B/DIO12 | B/DIO13 |
| 17 | B/DIO14 | B/DIO15 |
| 18 | DGND | DGND |
| 19 | SPI\_MOSI | RESERVED |
| 20 | SPI\_MISO | RESERVED |
| 21 | SPI\_CS\_n | RESERVED |
| 22 | SPI\_CLK | RESERVED |
| 23 | DGND | DGND |
| 24 | +3.3V | B/DIO17 |
| 25 | B/DIO16 | B/DIO19 |
| 26 | B/DIO18 | A/DIO19 |
| 27 | A/DIO18 | A/DIO17 |
| 28 | A/DIO16 | PRESENCE |
| 29 | A/DIO14 | A/DIO15 |
| 30 | A/DIO12 | A/DIO13 |
| 31 | A/DIO10 | A/DIO11 |
| 32 | A/DIO8 | A/DIO9 |
| 33 | A/DIO6 | A/DIO7 |
| 34 | A/DIO4 | A/DIO5 |
| 35 | A/DIO2 | A/DIO3 |
| 36 | A/DIO0 | A/DIO1 |
| 37 | DGND | DGND |
| 38 | RESERVED | RESERVED |
| 39 | AGND | AGND |
| 40 | B/AI0 | B/AI4 |
| 41 | B/AI1 | B/AI5 |
| 42 | B/AI2 | B/AI6 |
| 43 | B/AI3 | B/AI7 |
| 44 | AGND | AGND |
| 45 | A/AI3 | A/AI7 |
| 46 | A/AI2 | A/AI6 |
| 47 | A/AI1 | A/AI5 |
| 48 | A/AI0 | A/AI4 |
| 49 | RESERVED | RESERVED |
| 50 | KEYWAY | KEYWAY |
| 51 | KEYWAY | KEYWAY |
| 52 | RESERVED | RESERVED |
| 53 | RESERVED | RESERVED |
| 54 | B/AO0 | RESERVED |
| 55 | AGND | RESERVED |
| 56 | RESERVED | AGND |
| 57 | RESERVED | B/AO1 |
| 58 | RESERVED | RESERVED |
| 59 | AGND | AGND |
| 60 | A/AO0 | A/AO1 |
| 61 | RESERVED | RESERVED |
| 62 | RESERVED | RESERVED |

Figure 5: Top Board Pinout

## Signal Descriptions

|  |  |  |
| --- | --- | --- |
| Signal Name | Direction | Description |
| A/AI0:7 | IN | Bank A analog input channels 0 through 7. Includes a dedicated ADC, gain stage, mode selection, and channel multiplexer. |
| B/AI0:7 | IN | Bank B analog input channels 0 through 7. Includes a dedicated ADC, gain stage, mode selection, and channel multiplexer. |
| A/AO0:1 | OUT | Bank A analog output channels 0 and 1. Each channel includes a dedicated DAC. |
| B/AO0:1 | OUT | Bank B analog output channels 0 and 1. Each channel includes a dedicated DAC. |
| A/DIO0:19 | BIDIR | Bank A digital input/output channels 0 through 19. Each channel can operate independently of every other channel. DIO 16:17 are multiplexed with a UART controlled by the RIO processor. |
| B/DIO0:19 | BIDIR | Bank B digital input/output channels 0 through 19. Each channel can operate independently of every other channel. DIO 16:17 are multiplexed with a UART controlled by the RIO processor. |
| +15V | OUT | Fixed voltage supply rail for general purpose use. |
| -15V | OUT | Fixed voltage supply rail for general purpose use. |
| +5V | OUT | Fixed voltage supply rail for general purpose use. |
| +3.3V | OUT | Fixed voltage supply rail for general purpose use, and is also typically used to power the board identification SPI FLASH. |
| DGND | REF | Digital ground, should be used as the reference for the digital signals and fixed voltage power supplies. |
| AGND | REF | Analog ground, should be used as the reference for the analog input and output signals. |
| PRESENCE | IN | Used to detect if an ELVIS III top board is installed. Must be tied directly to DGND. |
| SPI\_CS\_n | OUT | Active low chip select for the board identification non-volatile storage. |
| SPI\_CLK | OUT | Serial clock for the board identification non-volatile storage. |
| SPI\_MOSI | OUT | Data to the slave for the board identification non-volatile storage. |
| SPI\_MISO | IN | Data to the master for the board identification non-volatile storage. |
| USB D+/D- | BIDIR | Differential pair for USB 2.0 host port. |
| USB\_VBUS | OUT | Voltage supply rail for USB 2.0 host port. |
| RESERVED | N/A | These pins should not be connected on an ELVIS III top board, and may contain signals used for other purposes. |
| KEYWAY | N/A | The edge connector is notched across these locations to allow the top board to plug into the keyed top board connector on ELVIS III. |

Figure 6: Top Board Signal Descriptions

## Multisim and Ultiboard Design Templates

User defined templates for circuit schematics and PCB layouts provide a comprehensive starting point to engineers to begin designing their circuits and printed circuit boards. With a template, various elements such as connector placement, critical devices and board layout can be pre-configured to aid in the design process.  Templates include all the basic content needed to help speed design creation of a custom ELVIS topboard in the Multisim environment. The Multisim and Ultiboard templates are attached to this manual and will be provided natively in the software.

A partner is not obligated to create their board designs in Multisim and Ultiboard and have the option to choose a different board layout tool of choice, however, NI will not provide a design template for such case.

## Hardware Capabilities

### Analog Input

The analog input section consists of two identical banks of analog inputs. Each bank includes an ADC, gain stage, mode selection, and channel multiplexer. The gain stage determines the input range for the conversion (±10V, ±5V, ±2V, ±1V). The mode selection controls whether the input is configured for single ended or differential measurement. A single ended measurement measures the difference between the selected signal and ground. A differential measurement measures the difference between the selected signal and its associated signal pair. The differential pairs are AI0/AI4, AI1/AI5, AI2/AI6, and AI3/AI7. The channel multiplexer is used to select the active channel.

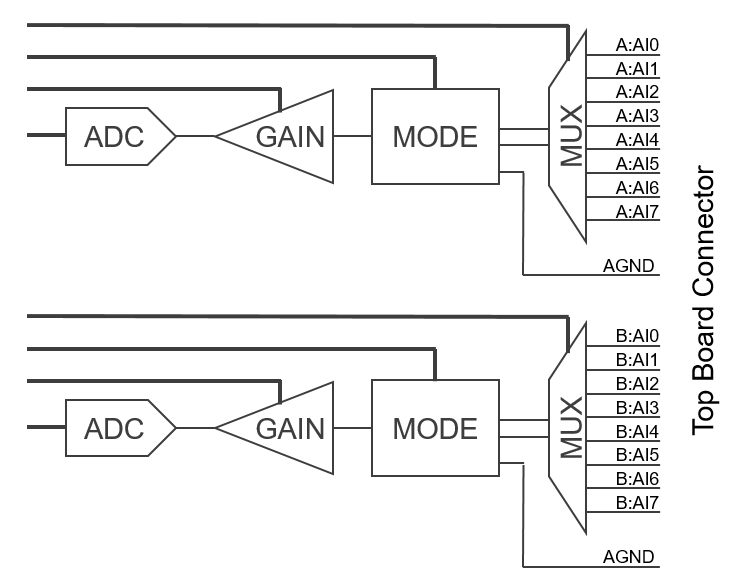


Figure 7: Analog Input Circuitry

At a hardware level the gain, mode, and channel selections must be configured and the analog signal path allowed to settle sufficiently before initiating a conversion. When acquiring data from multiple channels, the configuration must be rewritten to the hardware after each conversion in preparation for the next conversion. Allowing additional time when scanning between channels can increase measurement accuracy, which is dependent upon not only the configuration but the impedance and voltage levels of the input sources. When acquiring data from a single channel the configuration is only written the first time and persists for subsequent conversions, unless explicitly changed by the user. There are no hardware dependencies between the two banks of analog inputs, and conversions on one bank have no impact on the other. The banks can also be operated synchronously, providing simultaneous sampling on two channels at a time (one from each bank).

These low-level operations are partially abstracted from the user by the LabVIEW FPGA API. Higher level APIs and applications are built on top of the LabVIEW FPGA API.

### Analog Output

The analog output section consists of four identical channels of analog output. Each channel consists of a DAC and output buffer with a fixed ±10V output range. There are no configuration settings for the output channels.

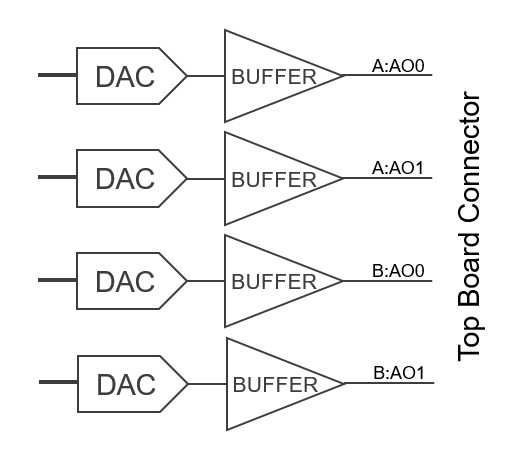


Figure 8: Analog Output Circuitry

There are no hardware dependencies between the four channels, and updates on one channel have no impact on the other channels. The channels can also be operated synchronously, providing simultaneous updates on any combination of the four channels. The channels are named and organized as two banks of outputs to simplify representation in the higher-level APIs and documentation, but at the lowest level of hardware and software there is no difference among the outputs.

### Digital Input/Output

The digital input/output section consists of 40 channels of digital I/O. The basic I/O capabilities are identical across all 40 channels. Each has independent control over its input, output, and output enable. Four of the channels are multiplexed with two processor-based UARTS. When the processor enables a UART, the corresponding digital channels are taken over by the UART and unavailable to the digital I/O API. Each UART consists of a transmit and receive line on DIO17 & 16, respectively. The channels are named and organized as two banks of digital I/O to simplify representation in the higher-level APIs and documentation, but at the lowest level of hardware and software there is no difference among the channels.

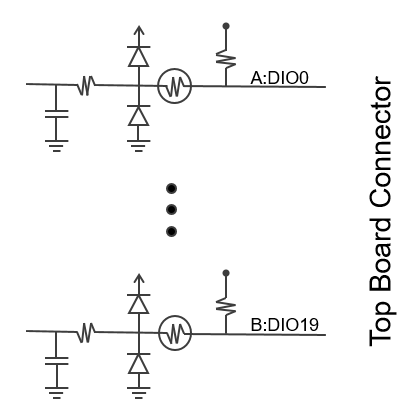


Figure 9: Digital Input/Output Circuitry

Each channel includes an RC filter, protection circuitry, and a programmable pull resistor. Once an ELVIS III top board is detected, they are all automatically configured as pull up resistors.

### USB Host Port

A USB 2.0 host port is provided which includes the USB differential data pair and USB power rail (limited to 900mA). This port is intended for captive use on the top board, as opposed to the general-purpose USB host ports provided on ELVIS III itself. Example uses include onboard flash storage, communication interfaces such as RS-232 or CAN, or a camera. If a general-purpose USB connection is desired for arbitrary use, a USB hub should be integrated into the top board to ensure that full USB specifications are maintained at the USB connector.

### Board Identification

Board identification consists of two phases. First, the presence of the top board is determined through the Presence pin. ELVIS III can detect if this pin is tied hard to digital ground, power, or left floating (corresponding to an ELVIS III top board, ELVIS II/II+ top board, or no installed top board). If an ELVIS III is detected, the second phase is started. ELVIS III will then read the non-volatile identification information from the top board, which is usually stored in a SPI EEPROM or similar. Top boards that are licensed by National Instruments include such information as Vendor ID, Product ID, Product Name, and more. Top boards that are not licensed by National Instruments must include the generic information block specified below, which does NOT include vendor or product information, and the top board will be identified as a generic top board to upper level software.

#### Timing Diagram

All top boards must adhere to the following timing diagram. ELVIS III drives SPI\_CS\_n, SPI\_CLK, and SPI\_MOSI to the top board and the top board dives MISO back to the carrier, with the SPI\_MOSI and SPI\_MISO timing specified relative to SPI\_CLK. The top board samples SPI\_MOSI on the rising edge of SPI\_CLK, and drives SPI\_MISO on the falling edge of SPI\_CLK. ELVIS III drives MOSI on the falling edge of SPI\_CLK, as well as sampling SPI\_MISO on the falling edge of SPI\_CLK. Refer to the read and write sequence diagrams in the following sections for when data is expected to be valid on MOSI and MISO.

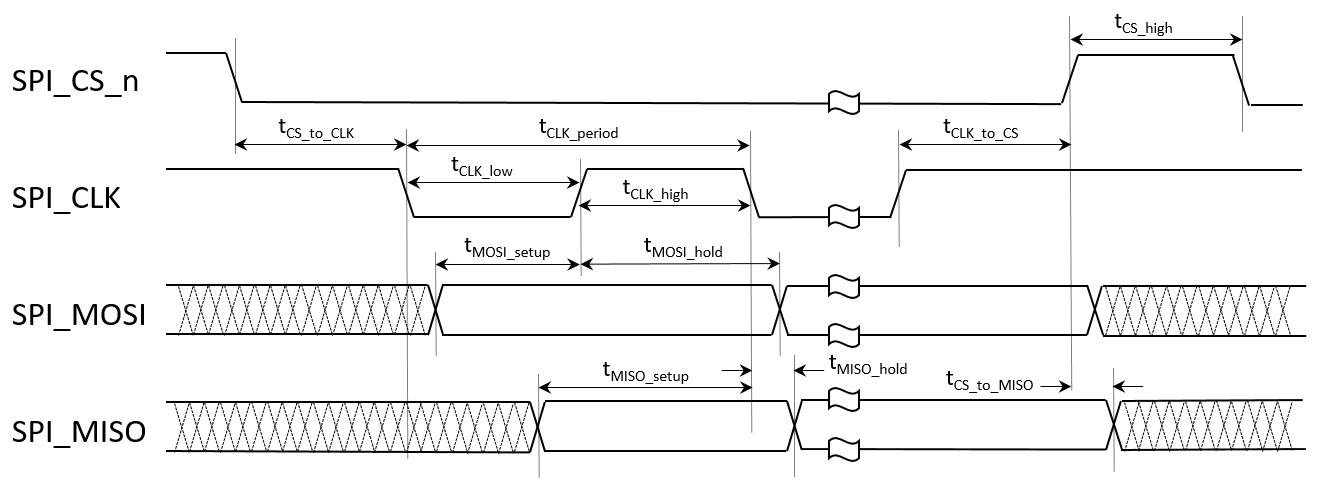


Figure : Board ID Timing Diagram

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Min | Max | Comment |
| tCLK\_period | 990ns | 1010ns | Driven by ELVIS III, nominally 1MHz clock |
| tCLK\_low | ≥450ns |  | Driven by ELVIS III |
| tCLK\_high | ≥450ns |  | Driven by ELVIS III |
| tCS\_high | ≥450ns |  | Driven by ELVIS III |
| tCS\_to\_CLK | ≥450ns |  | Driven by ELVIS III |
| tCLK\_to\_CS | ≥450ns |  | Driven by ELVIS III |
| tMOSI\_setup | ≥225ns |  | Driven by ELVIS III |
| tMOSI\_hold | ≥225ns |  | Driven by ELVIS III |
| tMISO\_setup | ≥225ns |  | Must be met by top board |
| tMISO\_hold | ≥0ns | ≤450ns | Must be met by top board |
| tCS\_to\_MISO | ≥0ns |  | Must be met by top board |

Figure : Board ID Timing Parameters

#### Required Read Support

All ELVIS III top boards must support the standard read sequence. This read sequence supports common SPI EEPROM devices. Top boards are not required to store the Board ID information in a SPI EEPROM, however, whatever device they use (SPI\_EEPROM, microcontroller, CPLD, FPGA, etc.) must adhere to the read sequence shown below.

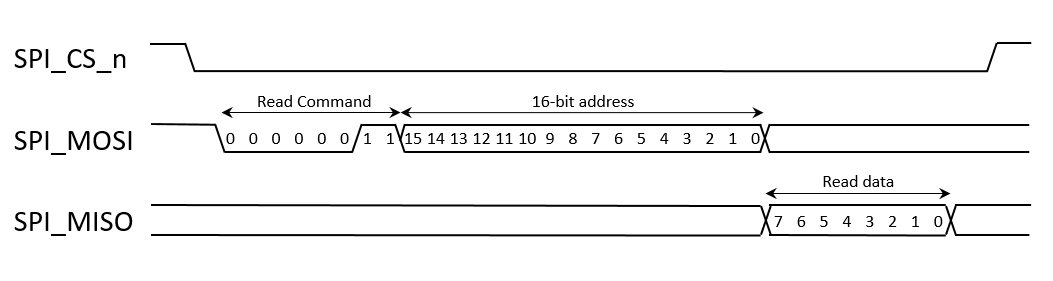


Figure : Required Board ID Read Sequence

#### Optional Write Support

Portions of the identification and description information may need to be written in manufacturing or in the field. Examples of these may include the module serial number or date of manufacture. A standard write sequence is defined to provide a common mechanism for writing this information. The standard write sequence is comprised of three parts: enabling the device for writing, initiating the write, and waiting for the write to complete. No other read or write cycles are permitted while a given write cycle is in progress, and these three parts must be performed in order.

Enabling the device for writing consists of a single transaction where the carrier drives SPI\_CS\_n low, sends an 8-bit enable command, then drives SPI\_CS\_n high again. Initiating the write consists of a single transaction where the carrier drives SPI\_CS\_n low, sends an 8-bit write command and a 16-bit address, writes 1 byte to the module, then drives SPI\_CS\_n high again. Waiting for the write to complete consists of repeated transactions where the carrier drives SPI\_CS\_n low, sends an 8-bit read status command, reads a status byte from the module, then drives SPI\_CS\_n high again. If the least significant (last) bit of the status register is a 1, the write cycle is still in progress. The device shall take no longer than 100ms to complete a write cycle and clear the status bit.

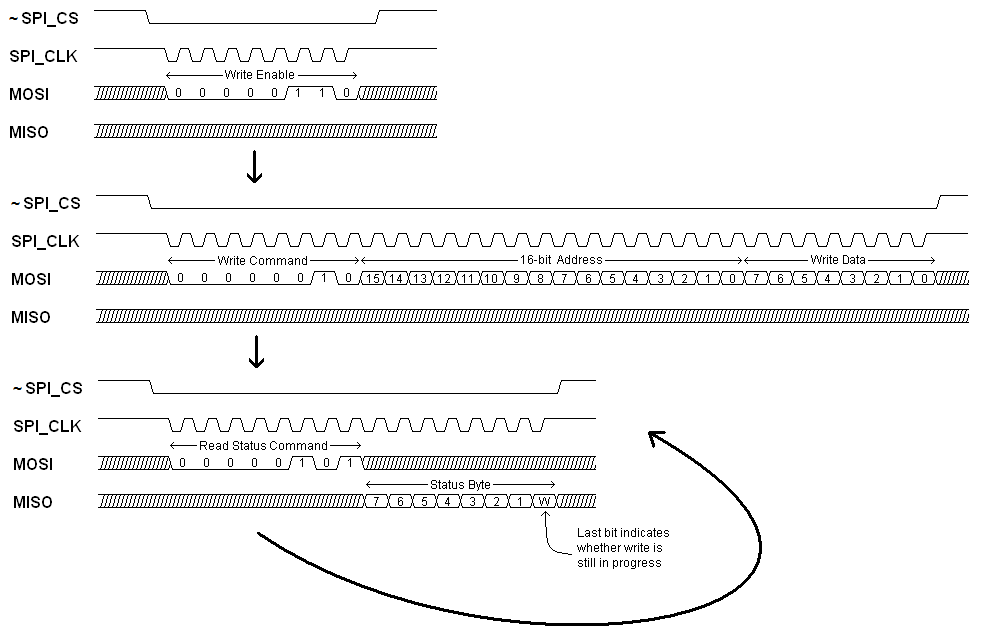


Figure : Optional Board ID Write Sequence

#### Example Implementation

A representative component that adheres to the timing diagram and the read and write sequences is the Microchip 25LC080D. A representative schematic for implementing the Board ID circuitry is shown in Figure 14.

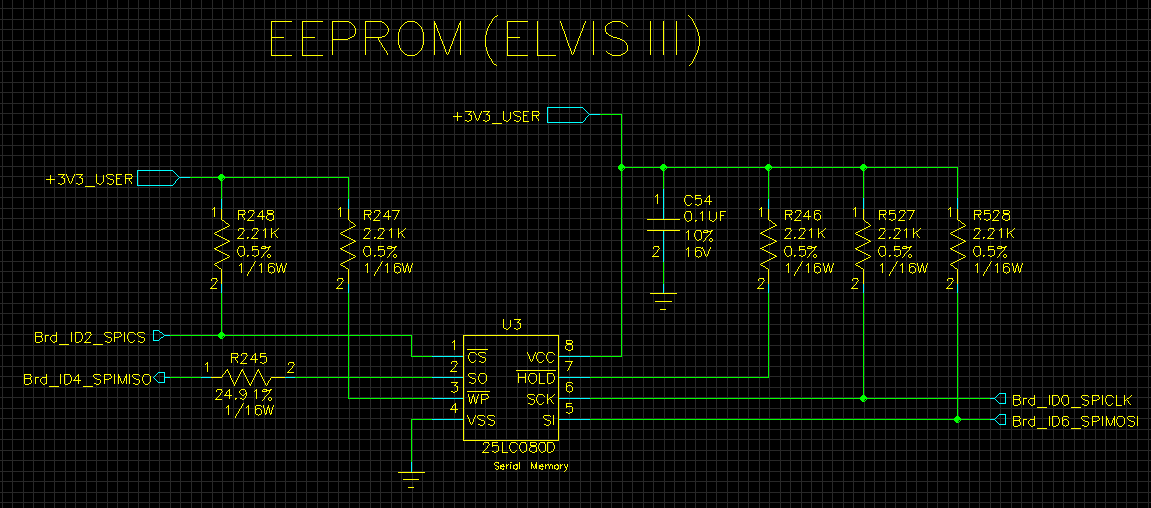


Figure : Example Board ID Schematic

#### Board ID Contents

The Board ID contents consist of two portions: a required binary portion and an optional XML portion. The binary portion includes information that allows for the positive identification of the application board model. The XML portion provides additional information about the board to improve the user experience and supportability of the platform. There is a high level LabVIEW RT API for reading the information. There is a low level LabVIEW RT API for reading and writing the information. The low level API is intended for writing the information in manufacturing or for field updates of the information.

|  |  |  |
| --- | --- | --- |
| Address | Short Description | Size in Bytes |
| 0x0000 | Map Revision | 1 |
| 0x0001 – 0x0002 | Vendor ID | 2 |
| 0x0003 – 0x0004 | Product ID | 2 |
| 0x0005 – 0x0006 | CRC | 2 |
| 0x0007 – 0xnnnn | XML Data | Variable |

Figure : Board ID Register Map (for Map Revision 0x1)

|  |  |
| --- | --- |
| Field Name | Description |
| Map Revision | Used to identify the format of the Board ID contents. Allows software to know how to interpret the remaining contents. Current Map Revision: 0x1 |
| Vendor ID | Used to uniquely identify the manufacturer of the application board. Value assigned by National Instruments to licensed application board vendors. Unlicensed vendors creating boards for non-commercial use must set this value to 0x02. |
| Product ID | Used to identify the model of the application board. Product IDs are assigned by the vendor, and are not unique across vendors. An application board is uniquely identified through the combination of Vendor ID and Product ID. |
| CRC | Used to verify the integrity of the preceding binary fields. Algorithm described below. |
| XML Data | Optional, but recommended information stored in UTF-8 format. Several tags have been predefined by National Instruments for standardized information. The prefix “NI.” has been reserved for standardized information. Vendors may define their own tags but they must not start with “NI.” |

Figure : Board ID Field Descriptions

|  |  |
| --- | --- |
| XML Tag | Description |
| <NI.SN>12345678</NI.SN> | Application board serial number. Application boards of a given model are expected to have unique serial numbers. |
| <NI.Ver>A.1</NI.Ver> | Application board revision. Used to track different versions of the same model. Recommended format is [Major Revision].[Minor Revision], although any format is allowed. Major Revision is usually a letter (A, B, C, …) and is often used to denote a user visible change (notably improved specifications, additional feature, etc.). Minor revision is usually a number (1, 2, 3, …) and usually denotes an implementation change not visible to the user (alternate components, cost reductions, etc.) |
| <NI.Vendor>A Company</NI.Vendor> | The vendor name stored as a string. |
| <NI.Product>A Product</NI.Product> | The product name stored as a string. |
| <NI.URL>http://abc.com/page</NI.URL> | A link for more product information. |

Figure : Board ID XML Tags

#### Board ID CRC Format

The definition of the CRC algorithm is under development.

# Top Board Software Access

There are multiple software options for accessing and controlling most of the signals on the top board connector. The LabVIEW FPGA API provides the lowest level of access to the IO, offering a high degree of flexibility and performance. The LabVIEW RT API provides higher level access with premade functions, accelerating application development. C/C++ support provides similar high level access to the IO for text based programmers. Finally, soft front panels (SFPs) provide an interactive environment on the host computer, offering a rapid no-programming approach to rapidly accessing and controlling the IO.

## LabVIEW FPGA API Support

LabVIEW FPGA support is provided for the analog inputs, analog outputs, and digital I/O on the top board connector. This is in addition to the LabVIEW FPGA support provided for other ELVIS III resources, such as the user LEDs and user button located on the side of the enclosure. Access is provided through a combination of IO, method, and property nodes. To find detailed descriptions of these nodes, select **Help** -> **LabVIEW Help** in LabVIEW. In the *LabVIEW Help*, navigate to **FPGA Module -> Creating FPGA VIs -> Using FPGA IOs**. If you need to program with the default FPGA personality, you could take the section Default FPGA Shipping Personality for reference.

### Analog Input

#### IO Node

The Analog Input IO Node allows user to have direct access to all the 16 AI channels. There are two banks of scanned AI channels.

#### Method Node to Change the AI Range

The method node is introduced for the Analog Input to make advance analog input programming easier. It requires user to make use of a specific pipeline approach in or der to achieve maximum sampling rate.

### Analog Output

The Analog Output IO Node allows user to have direct control on each of the 4 parallel AO channels.

### Digital Input and Output

The Digital Input and Output IO Nodes allow user to have direct access to the 40 DIO channels. Each channel can run as DI and DO at the same time.

## LabVIEW API Support

### LabVIEW APIs for Top Board Access

LabVIEW provides a set of VIs that you use to develop instrumentation, acquisition, and control applications. These VIs provide access to PCI peripherals and instruments on ELVIS III. To find detailed description of these VIs, press <Ctrl-H> after you drop a VI on the block diagram in LabVIEW to display the **Context Help** window. Click **Detailed Help** in the **Context Help** window to launch the full help. To navigate to or search for other related topics, go to <LV>\help and open elvis3vis.chm. The LabVIEW APIs will be available by private beta.

### Use LabVIEW APIs with Custom FPGA Personality

Refer to [Understanding FPGA Personality](http://digital.ni.com/express.nsf/bycode/ex7cpk) for the peripherals supported by the default shipping FPGA personality. You may encounter three situations when you use LabVIEW APIs with a custom FPGA personality:

#### Situation 1: The peripherals supported by the custom FPGA personality is the subset of the default shipping FPGA personality

1. Define a CSV file which describes the peripherals supported by the FPGA personality. The format of the CSV file is as follows:
   * The first column defines the primary peripheral category
     + AI/AO/DIO/PWM/SPI/Encoder/I2C/UART/AI IRQ/DI IRQ/Timer IRQ/AI (N Samples)/AO (N Samples)/DI (N Samples)/DO (N Samples)
   * The second column defines the secondary peripheral category to distinguish between DI, DO, LED and Button.
     + DI/DO/LED/Button
   * The third column is the channel name of the peripheral, for example, A/AI0, B/AO0, A/DIO0, etc.
   * The forth column is the pinout mapping of the peripheral which will be used in the connection diagram in Express VI, for example, (Pin 3), (Pin 18), etc.
2. Use the internal tool “FPGA XML Generator” to generate the XML file for the FPGA Personality. The file name of the bitfile should be aligned with the bitfile.
3. Copy the bitfile and XML file of the custom FPGA personality to <LV>\vi.lib\myRIO\FPGA\bitfiles.
4. Open the <LV>\ProjectTemplates\MetaData\NILV\_NI ELVIS III\_TemplateMetaData.xml.
5. In the “ProjectTemplate” section of “NI ELVIS III Project", add the symbol of the custom FPGA personality to “SupportedFPGAPersonalities"

#### Situation 2: The peripheral supported by the custom FPGA personality is the subset of default shipping FPGA personality, but with channel(s) added or removed.

Follow all steps described in Situation 1, and then update the Low Level VIs in the Instrument Driver Framework to support new channels.

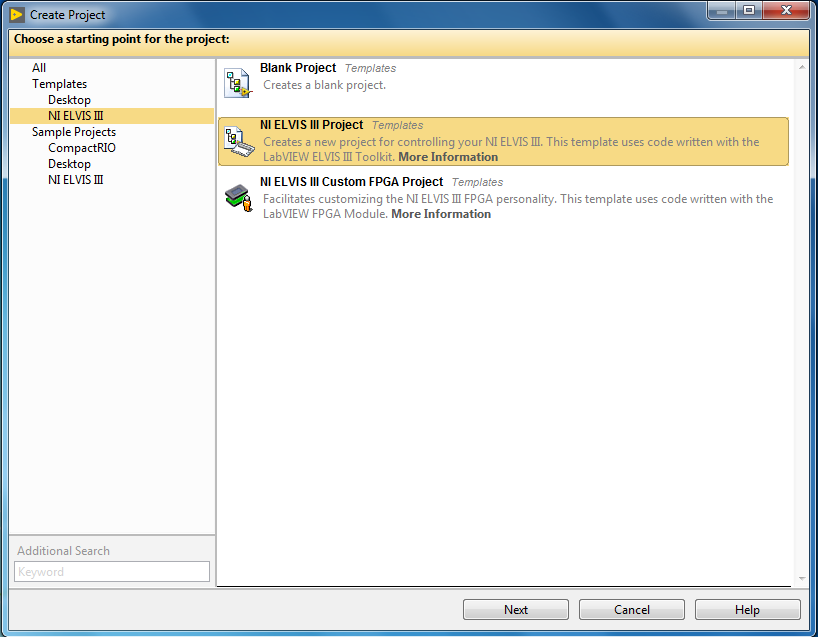
* 1. Go to <LV>\vi.lib\myRIO\Common\Instrument Driver Framework\myRIO\<peripheral name>\typedefs.
  2. Open “<peripheral name> Channels Enum.ctl” and add the new channel(s).
  3. Go to <LV>\vi.lib\myRIO\Common\Instrument Driver Framework\ELVIS III v1.0\<peripheral name>.
  4. Go to the typedefs folder
     + Open the typedef of the peripheral channel name, and add the new channel(s).
     + Create a new “<peripheral name> Channels FPGA Reference.ctl” which describes the FPGA registers of the peripheral.
  5. Go to vis folder
     + Open the Build MUX Configuration VI to add new case structure to support the new channel(s). You could take the existing case structure for reference.
     + Open the Write Read API to add new case structure to access the FPGA resource for the new channel(s). You could take the existing case structure for reference.

#### Situation 3: The custom FPGA personality supports a new peripheral which is not supported by the default shipping FPGA personality

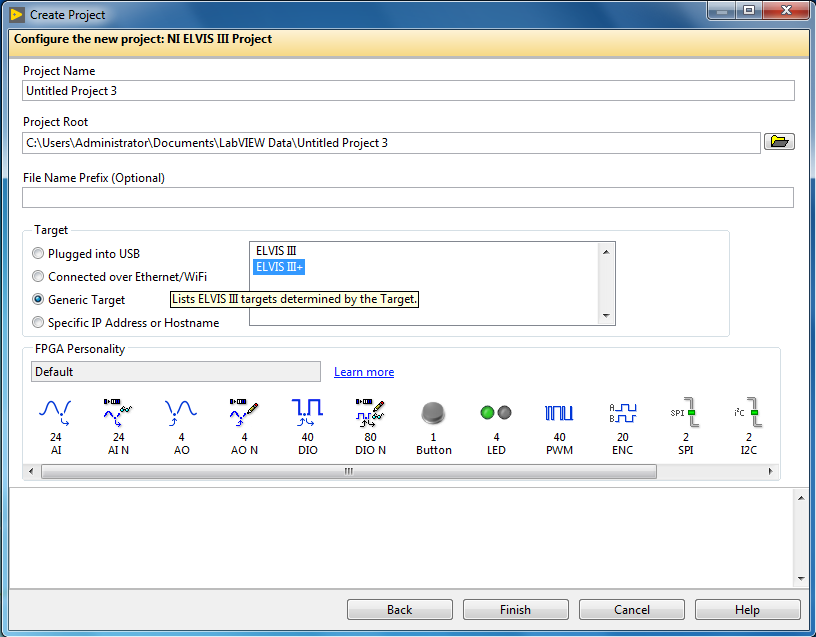
The current infrastructure does not allow the user to do this.

## Migrating NI ELVIS II/II+ Code

1. Go to LabVIEW Project Wizard, and select **NI ELVIS III Project.**



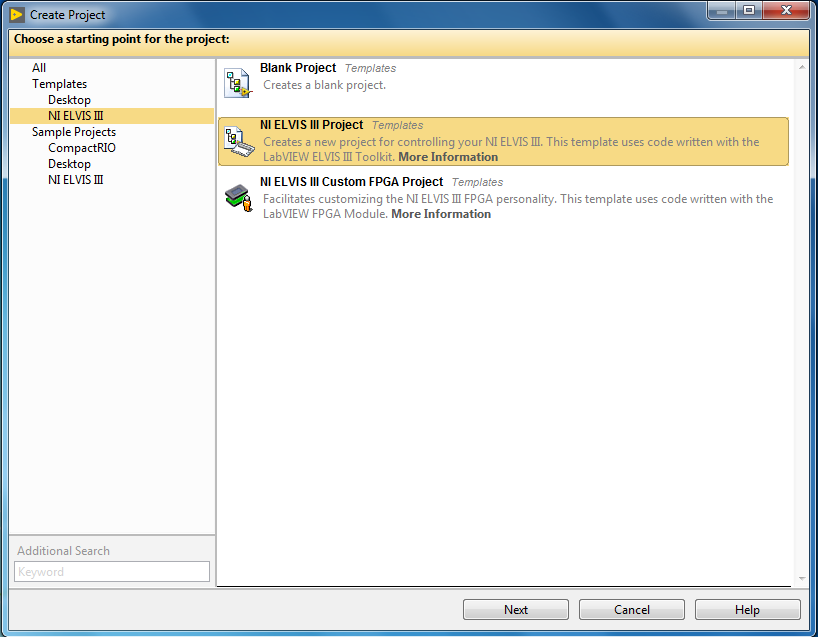
1. Select the target you want to use or select the generic target type in the wizard, and click **Finish.**



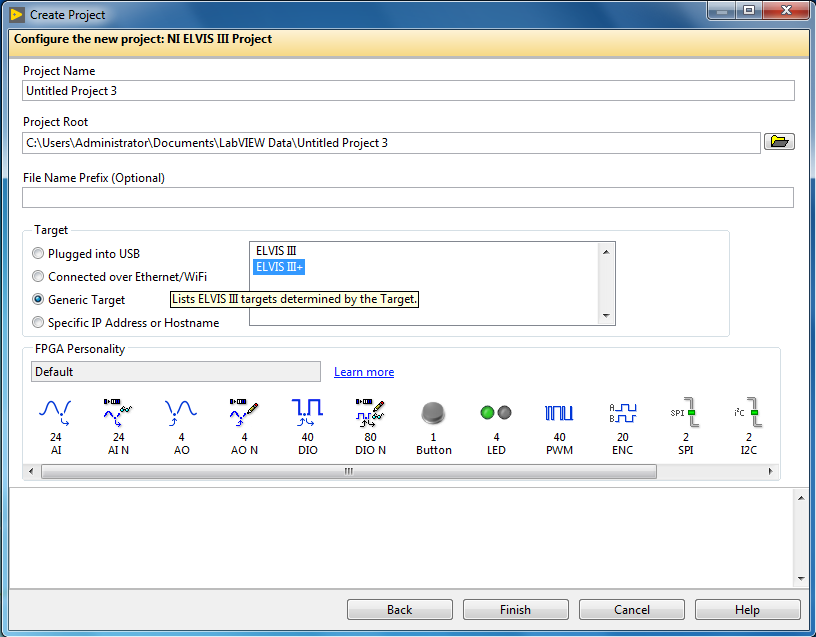
1. Move your existing ELVIS II/II+ or DAQmx VIs to the NI ELVIS III target.
2. Replace the existing ELVIS II/II+ or DAQmx APIs with the NI ELVIS III Instrument APIs. You could find these APIs from the Instrument palette on the elvis palette.

## Migrating myRIO Code

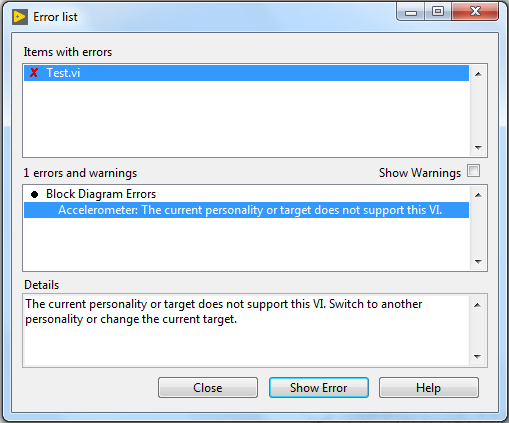
1. Go to LabVIEW Project Wizard, and select **NI ELVIS III Project**.



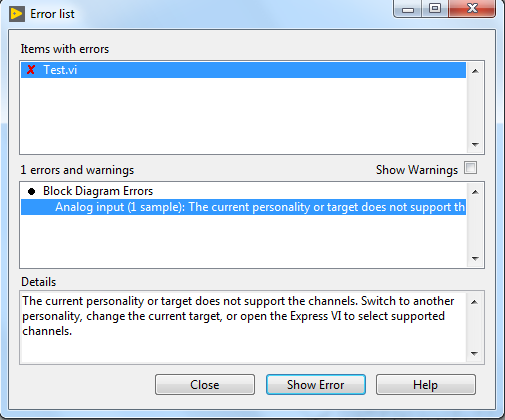
1. Select the target you want to use or select the generic target type in the wizard, and click **Finish.**

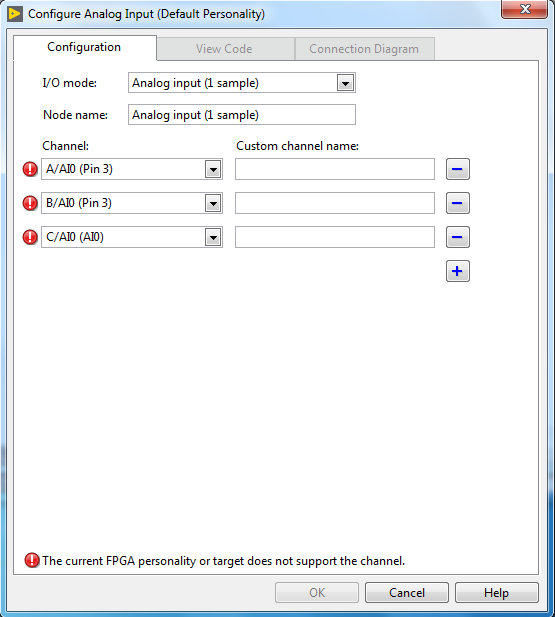


1. Add the existing myRIO VIs to the NI ELVIS III target.
2. If you are using the Express VI, there may be some edit time error messages.
   * If there is an error message about unsupported peripheral, you cannot use this Express VI on the NI ELVIS III.



* + If there is an error message about unsupported channel, you need to specify a new channel supported by NI ELVIS III.





1. If you are using the Low Level API, there may be some run time error messages.
   * If there is an error message about unsupported peripheral, you cannot use this Low Level API on NI ELVIS III
   * If there is an error message about unsupported channel, you need to specify a new channel supported by NI ELVIS III in the Open API.
2. If you are using UART Low Level API, you need to use the Reserve UART API before accessing the UART, and use the Unreserve UART API to free this resource.

## Instrument Launcher

The Instrument Launcher is a toolbar that appears when using ELVIS III. Use it to connect to ELVIS III devices and launch SFPs.



It can appear in:

* Courseware pages that require connection to an ELVIS III device.
* ni.com pages that require connection to an ELVIS III device.
* Direct connections from a computer to an ELVIS III device.

The Instrument Launcher has the following tabs:

* Device - For managing device connection.
* Instruments - For launching the connected device’s instruments (SFPs). This tab is not available until a device is connected.
* Resources - For general resources and resources specific to the current courseware. There are tabbed views within this flyout. Content will vary depending on browser window size, device connection status, and connected hardware. For example, Using ELVIS III, Learning Community, Reference and Support.

### Device Tab

Use to connect to an ELVIS III device.

Click the Device tab and click one of:

* **Connect via USB** – Plug the ELVIS III device into your computer and click Connect.
* **Connect via network** – Enter the code from an ELVIS III that is connected to your network and click Connect. The code displays on the ELVIS III's LCD.

### Instruments Tab

Instruments on the connected ELVIS III device appear here.

Click on an instrument to open its SFP.

### Resources Tab

Links to useful information appear here.

# Teaching Resources Development

## Teaching Resources Quality Guidelines

**Instructional Design**

* Course learning objectives and outcomes.
  + Use observable verbs. For example, “explain” rather than “understand.”
  + Each learning objective should have a condition, audience, behavior, and degree. Refer to this [useful tool](https://cdl.ucf.edu/teach/resources/objective-builder-tool/).
  + Use verbs that are associated with the appropriate degree of mastery in Bloom’s Taxonomy (modified version). Bloom’s wheel is a [useful tool](https://ep.jhu.edu/files/ep-blooms-wheel.pdf).
  + Outline course learning objectives into a taxonomy so that different modules and activities contribute to smaller parts of the overall learning objectives.
  + The course must meet learning objectives.
* Format.
  + Module titles must accurately describe the activity.
  + Entire course and individual modules must include introductions and conclusions.
  + Modules must have a clear hierarchy and flow that is natural for users to navigate.
* Assessments.
  + Assessments must check if learning objectives were achieved.
  + Assessment questions must clearly state expectations of the student.
  + Assessment answers must be accurate on the answer key.
  + If an assessment question asks for a numeric answer, expected units must be clear.
  + Assessment difficulty must appropriate for specified difficulty and prerequisites of the course.
  + Assessment questions must match specified teaching style of the course.
    - Project based – Each project should have clear expectations for the student. Assessments should be used to document the student’s approach and encourage reflection.
    - Traditional lab – Assessment questions should include straight-forward checks for understanding, observations of simulations and experiments, analysis, and reflection. Some questions may be open-ended.
  + If an assessment is included, there must be a corresponding answer key. If an answer is open ended, or if there are several ways a task can be accomplished, this should be documented with clear guidance of what work is acceptable or what is not acceptable.
* Techniques.
  + Include real-world applications and examples whenever possible.

**Content**

* Avoid plagiarism, including mosaic plagiarism – always use proper citations.
  + When using citations in text, bibliographies, and lists of works cited, refer to the *Chicago Manual of Style, Fourteenth Edition* guidelines on author-date text and electronic citations.
  + Use a trusted plagiarism checker, such as Grammarly.
  + When using content written by NI on the NI Website or NI help documents, citations are not required. Informal citations may be helpful, though, so the user can learn more about the subject matter and NI documentation.
    - Follow the [NI Trademarks and Logo Guidelines](http://www.ni.com/legal/trademarks/).
* Theory must be accurate, including statements, equations, examples and historical facts.
* Content must be appropriate for the specified difficulty and prerequisites of the course.
* Use active voice whenever possible.
* Use a professional, clear, and engaging tone. Guidelines:
  + Use technical jargon when appropriate, but be sure to define words when necessary based on the course's difficulty specification.
  + Use simple words and sentences when possible.
  + Avoid slang, clichés, and colloquialisms, especially words and phrases that are regional.
  + Use extravagant and hyperbolic language sparingly.
* Use a voice that speaks directly to the reader. For example, use “you,” not “the user”.
* Use correct spelling and grammar.
* Use correct spelling and capitalization of NI product names. Refer to product documentation.
* Use standardized terms and phrasing throughout the resource. That is, once you refer to a subject using a term, stick to that term.
* Don’t use:
  + i.e. – use “that is.”
  + e.g. – use “for example.”
  + vs. – use “as opposed to” or “versus.”
  + etc. – avoid entirely if possible. Otherwise, use “and so on” sparingly.
* Acronyms.
  + Define non-common-knowledge acronyms at the first occurrence in each topic. After defining an acronym, you can use it alone, without the definition.
  + Don’t use apostrophes when pluralizing acronyms.

**Technical Assets**

* Course preparation.
  + Hardware – Identify required NI, partner, and other hardware, including part numbers, model numbers, and links for purchasing, specifications, and user manuals. When available, refer to existing maintained getting started guides, quick start guides, and similar documents.
  + Software – Identify required platform software, modules, drivers, and toolkits and versions. Provide download links and clear installation instructions and installation order. When available, refer to existing maintained getting started guides, quick start guides, and similar content.
  + All other technical requirements must be identified, for example, operating system, USB requirements.
* Technical instructions.
  + Instructions must be accurate and clear.
  + Start each step with a verb.
  + Use one step for each instruction.
  + Avoid the terms “You should…” or “Please…” since they imply choice when none exists.
  + Match the level of detail in the instructions with the difficulty and prerequisites specified for the course.
  + Include troubleshooting tips for common issues in the instructions.
  + Use imagery and other media when appropriate to supplement instructions.
* LabVIEW code.
  + Design according to the [LabVIEW style checklist](http://zone.ni.com/reference/en-XX/help/371361P-01/lvdevconcepts/checklist/).
  + Test all code for bugs and to ensure it completes the expected task (with hardware if applicable).
  + Test all code in as many different configurations as applicable, such as different operating systems and environments (when using sensors or vision that are sensitive to environmental factors like lighting).
  + Code must be ready to run as-is when opened, including customizations like default values where necessary.
    - Exception – If students are expected to complete code as an activity, code can be supplied unfinished, but tested along with instructions for completing the code.
  + All code must be well documented using numbered callouts and legend tables.
    - Exception – some documentation can be left out of student code if it conflicts with assessments or activities.
  + Provide all code in the most current version of LabVIEW, as well as the two previous LabVIEW versions if drivers support them.
  + Consider testing using the [LabVIEW Analyzer Toolkit](http://search.ni.com/nisearch/app/main/p/bot/no/ap/tech/lang/en/pg/1/sn/catnav:du/q/LabVIEW%20VI%20Analyzer%20Toolkit/?searchType_search_field=default&separator_search_field=-1&facetselection_search_field=ssnav%3Adwl&value_search_field=LabVIEW+VI+Analyzer+Toolkit&i_search_field=0&section_search_field=query).
* Multisim.
  + Test all circuits for bugs and to ensure they complete the expected task (with hardware if applicable).
  + Prepare circuits to be ready to run as-is when opened.
    - Exception – If students are expected to complete a circuit as an activity, it can be supplied unfinished, but tested using instructions.
  + Desktop version – Supply Multisim files in the most recent version of Multisim and the previous version if possible.
  + Multisim Live – Publish circuits publicly, and supply them as embeddables within the course if the student is not expected to edit the circuit. If they are expected to edit the circuit, provide a link to the Multisim Live circuit.
* Delivery.
  + All technical assets must be delivered in a zip file that contains all applicable dependencies (except for Multisim Live circuits) and/or be uploaded to Thinkscape to the appropriate labs and steps.
  + Student technical assets – Do not include any answer keys or answer code, circuits, graphs, data, and so on.
  + Professor technical assets – Include both the student version of the technical assets and any applicable answer keys and answer code, circuits, graphs, data, and so on.

**Media Assets**

* Avoid plagiarism.
  + An image or GIF that was obtained from elsewhere can only be used if:
    - It is in the public domain.
    - The image and its copyright were purchased.
    - The image is protected under creative commons, and usage meets the creative commons conditions.
  + A video that was obtained from elsewhere can only be used if it is a reference link or hyperlink.
    - Do not copy or embed videos from elsewhere.
  + If there is any uncertainty about copyright, do not use the media.
* Consent – National Instruments can provide a release form if required.
* Screenshots.
  + Use the following settings before taking screenshots:
    - Windows 7 instructions:
      1. Select Windows 7 Aero Theme (Right-click desktop > Personalize).
      2. Set background to solid white (Personalization > Desktop background).
      3. Turn off transparency (Personalization > Window color).
      4. Verify ClearType is on (Personalization > Display > Adjust ClearType).
      5. Turn off *Enable Transparent Glass* and *Show shadow under windows* (Computer > Properties > Advanced System Settings > Advanced Tab > Performance > Settings).
      6. Log off and log on again.
    - Windows 10 instructions:
      1. Select Windows 10 Theme (Right-click desktop > Personalize).
      2. Select “Overcast Gray” color (Personalization > Color > bottom left in the palette).
      3. Deselect checkboxes for *Show color for start, taskbar, action center, and title* and *Make Start, taskbar, and action center transparent* (Personalization > Color).
      4. Verify ClearType is on (Windows/start button > type Adjust ClearType text).
      5. Log off and log on again.
  + Screenshots must match the final version of code - if code is updated, screenshots must be updated.
* Image sizing.
  + All images must follow guidelines for size and resolution.
  + For the teaching portal, at least two images per lab must be provided to represent the lab. They should follow these guidelines as closely as possible:
    - 16x9 ratio.
    - Images should ideally be under 100 kB. If there is a lot of detail, the maximum size is 500 kB.
    - Image width should be between 700 and 2,000 pixels.
    - PNG file format is optimal – JPG and JPEG formats are acceptable.
    - Product shots must have transparent or white backgrounds.
* NI imagery.
  + When using icons, use the NI icon library as much as possible. This can be provided upon request.
  + When using NI product diagrams, use screenshots of the specifications or user guide when appropriate, and link to the specifications/user guide document.
  + Request permission and image files before using NI logos, refer to the [NI Logo and Trademarks Guidelines](http://www.ni.com/legal/trademarks/).
* Photos.
  + When a generic product photo is needed, use NI CDG photos supplied by NI when possible.
  + Breadboard circuits must be neat and clear.
* Videos.
  + Videos must be supplied with a script.
  + Videos should be no longer than two minutes each.

**Final Formatting**

* HTML.
  + Use standard template.
* Thinkscape.
  + Use standard template.
  + Test for load time – must be less than two seconds on broadband internet.

**Localization**

* By default, resources must be written in English, using American English styles and notations. For example, use "." instead of "," as a decimal marker. When localizing, update notations to the region.
* Consider the resource a global resource - use global examples, and avoid jargon and phrases that are specific to a region.
* When localizing, consider updating with examples and case studies that are pertinent to the specific region.

**Recommended Installation, Getting Started Guides, and other resources for NI products**

* Find software to download:
  + Primary academic installers: <http://www.ni.com/academic/download>.
  + Alternate resource: <http://www.ni.com/downloads/>.
* Install instructions and troubleshooting:
  + General: <http://www.ni.com/getting-started/install-software/>.
* General Learning Resources:
  + General – Feel free to refer to other existing resources on [ni.com/teach](https://learn.ni.com/teach).
  + Archived courseware – NI also has resources on and off the web that are outdated, but are still great resources.
  + Learn LabVIEW/DAQ/RIO: <http://www.ni.com/academic/students/learn/> .

## Templates for Creating Offline Versions of Curriculum

NI is providing two template documents; one that includes content guidance such as sections, headers, and descriptions that should be used for curriculum development, the other will provide styling and branding guidance of the curriculum. These templates are separate attachments to the MDK.

# Reference

## Default Shipping FPGA Personality

The LabVIEW ELVIS III Toolkit will also ship an FPGA personality as default, and it provides support for the following peripherals:

* Onboard devices (LEDs, button)
* Analog input
* Analog output
* Digital input/output
* Pulse-width modulation (PWM)
* Serial peripheral interface (SPI)
* Encoder
* Inter-integrated circuit (I2C)
* Interrupt request (IRQ)

Each peripheral is controlled through the use of its corresponding registers as outlined in this document.

#### System Control / Function Select

##### Function Select Registers (SYS.SELECTx)

The function select registers control the functionality that is routed to the shared pins. You must enable the desired functionality at run time by setting or clearing the appropriate bits before you use the individual registers. The bit definition of each register for each connector type is given below.

* Tip Changing the register value switches between functions. This may have undesired effects if the connected peripheral is not intended to be connected to the alternate function.

Register list: SYS.SELECTA, SYS.SELECTB

Data type: U64

SYS.SELECTA and SYS.SELECTB select functionality on connectors A DIO [0:19] and B DIO [0:19], respectively. In this register, each DIO is represented by 2 bits from low to high. For example, Bits [0:1] is DIO 0, Bits [2:3] is DIO 1, etc. The functionality of the combination of 2 bits is shown in the following table:

|  |  |
| --- | --- |
| Bits | Functionality |
| 00 | The channel is used as DIO. |
| 01 | The channel is used as PWM. |
| 10 | The channel is used as Encoder. |
| 11 | The channel is used as SPI or I2C. |

For each DIO channel on connectors A and B, it could have different functionalities:

* **DIO**: DIO [0:19] on connectors A and B
* **PWM**: DIO [0:19] on connectors A and B
* **Encoder**: DIO [0:1], DIO [2:3], …, DIO [18:19] on connectors A and B
* **SPI**: DIO [5:7] on connectors A and B
* **I2C**: DIO [14:15] on connectors A and B

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DIO | PWM | Encoder | SPI | I2C | UART |
| DIO 0 | PWM 0 | ENC.A 0 |  |  |  |
| DIO 1 | PWM 1 | ENC.B 0 |  |  |  |
| DIO 2 | PWM 2 | ENC.A 1 |  |  |  |
| DIO 3 | PWM 3 | ENC.B 1 |  |  |  |
| DIO 4 | PWM 4 | ENC.A 2 |  |  |  |
| DIO 5 | PWM 5 | ENC.B 2 | SPI.CLK |  |  |
| DIO 6 | PWM 6 | ENC.A 3 | SPI.MISO |  |  |
| DIO 7 | PWM 7 | ENC.B 3 | SPI.MOSI |  |  |
| DIO 8 | PWM 8 | ENC.A 4 |  |  |  |
| DIO 9 | PWM 9 | ENC.B 4 |  |  |  |
| DIO 10 | PWM 10 | ENC.A 5 |  |  |  |
| DIO 11 | PWM 11 | ENC.B 5 |  |  |  |
| DIO 12 | PWM 12 | ENC.A 6 |  |  |  |
| DIO 13 | PWM 13 | ENC.B 6 |  |  |  |
| DIO 14 | PWM 14 | ENC.A 7 |  | I2C.SCL |  |
| DIO 15 | PWM 15 | ENC.B 7 |  | I2C.SDA |  |
| DIO 16 | PWM 16 | ENC.A 8 |  |  | UART.RX |
| DIO 17 | PWM 17 | ENC.B 8 |  |  | UART.TX |
| DIO 18 | PWM 18 | ENC.A 9 |  |  |  |
| DIO 19 | PWM 19 | ENC.B 9 |  |  |  |

#### Onboard Device Registers

These registers control the onboard LEDs and read the onboard button.

##### LED

Register list: DIO.LED3:0

*Data type*: U8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | LED3 | LED2 | LED1 | LED0 |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

This register controls the state of the onboard LEDs. Each bit corresponds to a single LED. If the bit is set to 1, the LED is lit. If the bit is set to 0, the LED is unlit.

* **Bits [7:4]** - Reserved for future use.
* **Bits [3:0]** - LED3:0

The desired state of onboard LEDs 3 to 0.

##### Button

Register list: DI.BTN

*Data type*: U8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | - | - | - | BTN |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 or 1 |

This register indicates the current state of the onboard button. A value of 1 means the button is pressed. A value of 0 means the button is not pressed. The button is internally debounced so you don’t need to add additional debouncing logic in software.

* Bits [7:1] - Reserved for future use.
* Bit [0] - BTN

The state of the onboard button. The initial value is either 0 or 1, depending on the initial state of this button.

#### AI/AO

##### Analog Counter Register (AI.x.CNT)

Register list: AI.A.CNT, AI.B.CNT

Data type: U8

This register shows how many valid channels there are in AI.X.CNFG and AI.X.VAL.

##### Analog Configuration Registers (AI.X.CNFG)

Register list: AI.A.CNFG, AI.B.CNFG

Data type: Array of U8

This register configures the AI channels to read. The number of the elements in the array depends on the AI.X.CNT register. The value of the configuration is shown in the following table.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | First Tick | - | AI Range | | AI Mode | AI Channel Selection | | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | | |

* **Bits [0:2]**: Specify which AI channel to configure.
  + Channel 0 : 000b
  + Channel 1 : 001b
  + …
  + Channel 7 : 111b
* **Bit 3**: Specify the acquisition mode of the analog input channel
  + DIFF: 0
  + RSE: 1
* **Bits [4:5]**: Specify the voltage range of the analog input channel
  + ±10 V: 00b
  + ±5 V: 01b
  + ±2 V: 10b
  + ±1 V: 11b
* **Bit 6**: Reserved for future use.
* **Bit 7**: First Tick Flag. True means this is the first tick after updating the configuration, and the data of this tick should be acquired.

##### Analog Divisor Registers (AI.X.CNTR, AO.X.DMA\_CNTR)

Register list: AI.A.CNTR, AI.B.CNTR, AO.A.DMA\_CNTR, AO.B.DMA\_CNTR

Data type: U32

This register is the divisor for the analog sample rate. The default onboard clock rate of FPGA is 40 MHz. This register equals the default onboard clock divided by the expected sample rate.

##### Analog Input DMA Enable Registers (AI.X.DMA\_ENA)

Register list: AI.A.DMA\_ENA, AI.A.DMA\_ENA

Data type: Boolean

Each eight analog input channels share one DMA on connector A and B respectively. This register controls whether the DMA is enabled for a specific connector.

##### Analog Ouptput DMA Enable Registers (AO.X.DMA\_ENA)

Register list: AI.A.DMA\_ENA, AI.A.DMA\_ENA

Data type: FXP

Each two analog output channels share one DMA on connector A and B respectively. This register controls whether the DMA is enabled for a specific analog output channel.

|  |  |  |
| --- | --- | --- |
| Bit | 1 | 0 |
| Name | AO1 | AO0 |
| Initial Value | 0 | 0 |

##### Analog DMA IDLE Registers (AI.X.DMA\_IDL, AO.X.DMA\_IDL)

Register list: AI.A.CNT, AI.B.CNT

Data type: Boolean

This register shows whether the DMA is idle.

##### Analog Value Registers (AI.X.VAL, AO.X.VAL)

**Analog Input**

Register list: AI.A.VAL, AI.B.VAL

Data type: Array of FXP

This register contains the value read by the analog input channel. The number of the elements in the array depends on the AI.X.CNT register, and the meaning of each element depends on AI.X.CNFG. These registers should be used only in one sampling acquisition. For N sampling, you should use the DMA directly.

**Analog Output**

Register list: AO.A\_0.VAL, AO.A\_1.VAL, AO.B\_0.VAL, AO.B\_1.VAL

Data type: FXP

This register contains the value written from the analog output channel. These registers should be used only in one sampling acquisition. For N sampling, you should use the DMA directly.

#### DIO

##### Data Direction Registers (DIO.xx.DIR)

Register list: DIO.A\_19:0.DIR, DIO.B\_19:0.DIR

Data type: FXP

This register controls the direction of the DIO channels. Each bit in the register controls the direction of one channel in the bank. For example, bit 0 corresponds to DIO 0, and bit 19 corresponds to DIO 19.

##### Pin Input Registers (DIO.xx.IN)

Register list: DIO.A\_19:0.IN, DIO.B\_19:0.IN

Data type: FXP

This register indicates the value read on the DIO channel. Each bit in the register indicates the value of one channel in the bank. For example, bit 0 corresponds to DIO 0, and bit 19 corresponds to DIO 19. The read value is 1 when a digital high voltage is applied to the pin. The read value is 0 when a digital low voltage is applied to the pin. The read value of output channels is undefined.

##### Pin Output Registers (DIO.xx.OUT)

Register list: DIO.A\_19:0.OUT, DIO.B\_19:0.OUT

Data type: FXP

This register controls the value written on the DIO channel. Each bit in the register controls the value on one channel in the bank. For example, bit 0 corresponds to DIO 0, and bit 19 corresponds to DIO 19. If the bit is set to 1, the pin returns a digital high voltage. If the bit is set to 0, the pin returns a digital low voltage. Output values only take effect when the channel is configured to be an output channel. If the OUT register is written to but the channel is set as an input, there is no effect on the pin. However, if the channel is changed to be an output, the voltage at the pin changes to be the value corresponding to the last value written to the OUT register.

* Note This follows the functionality of the Set Output Data and Set Output Enable FPGA IO Method nodes. Refer to the [LabVIEW Help](http://digital.ni.com/express.nsf/bycode/ex3gpa) for more information about using FPGA I/O.

##### Digital Divisor Registers (DI.X.DMA\_CNTR, DO.X.DMA\_CNTR)

Register list: DI.A.DMA\_CNTR, DI.B.DMA\_CNTR, DO.A.DMA\_CNTR, DO.B.DMA\_CNTR

Data type: U16

This register is the divisor for the digital sample rate. The default onboard clock rate of FPGA is 40 MHz. This register equals the default onboard clock divided by the expected sample rate.

##### Digital Input DMA Enable Registers (DI.X.DMA\_ENA)

Register list: DI.A.DMA\_ENA, DI.B.DMA\_ENA

Data type: Boolean

Each twenty digital input channels share one DMA on connector A and B respectively. This register controls whether the DMA is enabled for all digital input channels on the connector.

##### Digital Output DMA Enable Registers (DO.X.DMA\_ENA)

Register list: DO.A.DMA\_ENA, DO.B.DMA\_ENA

Data type: FXP

Each twenty digital output channels share one DMA on connector A and B respectively. This register controls whether the DMA is enabled for specific digital output channel.

##### Digital DMA Idle Registers (DI.X.DMA\_IDL, DO.X.DMA\_IDL)

Register list: DI.A.DMA\_IDL, DI.B.DMA\_IDL, DO.A.DMA\_IDL, DO.B.DMA\_IDL

Data type: Boolean

This register shows whether the DMA is idle.

#### PWM

##### PWM Configuration Registers (PWM.x.CNFG)

Register list: PWM.A\_[0:19].CNFG, PWM.B\_[0:19].CNFG

Data type: U8

This register configures the functionality of the PWM subsystem as shown in the following table.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | - | MODE | - | INV |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:3] - Reserved for future use.
* Bit [2] - MODE: Counter mode.

The mode of operation of the PWM counter.

* MODE = 0:

The counter operates in no PWM generation mode. The counter counts up to 65535, resets to 0, and repeats. The MAX and CMP registers have no effect on the counter and no PWM output is generated.

* MODE = 1:

The counter operates in PWM generation mode. The counter counts up to the value specified in the MAX register, resets to 0, and repeats. When the value equals the CMP register, a compare match occurs. The behavior of the PWM output on compare match is determined by the INV bit.

* Bit [1] - Reserved for future use.
* Bit [0] - INV : Invert Output

The functionality of this bit depends on the value of the MODE bit. When MODE = 0, the INV bit is not used. When MODE = 1, the INV bit causes the following behavior:

* INV = 0:

Clear the output on compare match, set at min counter value (non-inverting mode).

* INV = 1:

Set the output on compare match, clear at min counter value (inverting mode).

##### PWM Clock Select Registers (PWM.x.CS)

Register list: PWM.A\_[0:19].CS, PWM.B\_[0:19].CS

Data type: U8

This register controls the clock speed of the PWM counter.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | - | CS2 | CS1 | CS0 |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:3] - Reserved for future use.
* Bits [2:0] - CS : Clock select

|  |  |  |  |
| --- | --- | --- | --- |
| CS2 | CS1 | CS0 | Clock |
| 0 | 0 | 0 | Off (No clock) |
| 0 | 0 | 1 | 1x (fclk) |
| 0 | 1 | 0 | 2x (fclk / 2) |
| 0 | 1 | 1 | 4x (fclk / 4) |
| 1 | 0 | 0 | 8x (fclk / 8) |
| 1 | 0 | 1 | 16x (fclk / 16) |
| 1 | 1 | 0 | 32x (fclk / 32) |
| 1 | 1 | 1 | 64x (fclk / 64) |

The base clock frequency (fclk) is 40 MHz. Use this frequency when you calculate the value of MAX for the desired frequency. See the frequency generation section below on how to use the CS register.

##### PWM Maximum Count Registers (PWM.x.MAX)

Register list: PWM.A\_[0:19].MAX, PWM.B\_[0:19].MAX

Data type: U16

This register determines the maximum value of the PWM counter. If the MODE bit in the CNFG register is set to 1, the PWM counter counts to MAX, then resets to 0. Otherwise, this register is ignored.

##### PWM Compare Registers (PWM.x.CMP)

Register list: PWM.A\_[0:19].CMP, PWM.B\_[0:19].CMP

Data type: U16

This register sets the compare value, and therefore determines the duty cycle of the PWM. The behavior depends on the value of the MODE and INV bits in the CNFG register.

|  |  |  |
| --- | --- | --- |
| MODE | INV | Output Behavior |
| 0 | 0 or 1 | No output. CMP value is ignored. |
| 1 | 0 | Clear the output when CNTR = CMP. (non-inverting mode) |
| 1 | 1 | Set the output when CNTR = CMP. (inverting mode) |

##### PWM Counter Registers (PWM.x.CNTR)

Register list: PWM.A\_[0:19].CNTR, PWM.B\_[0:19].CNTR

Data type: U16

Range: 0 to 65535

This register indicates the current value of the PWM counter. If the MODE bit in the CNFG register is 0, the counter increments from 0 to 65535, then resets to 0 and repeats. If the MODE bit in the CNFG register is 1, the counter increments from 0 to the value specified in the MAX register, then resets to 0, and repeats. The counter increments at the rate determined by the value of the CS register.

##### PWM Frequency Generation

The NI ELVIS III hardware runs on a 40 MHz clock, which means the time between clock cycles is 25 ns. The NI ELVIS III can generate slower PWM frequencies by counting and changing the output on intervals of rising clock edges. The NI ELVIS III can generate PWM frequencies between 40 Hz and 40 kHz. You must downsample the 40 MHz clock to generate a slower frequency. For example, the following figure shows the generation of 20 MHz and 10 MHz clocks from a 40 MHz clock by changing the output every rising edge or every other rising edge, respectively.



Figure . Generating Slower PWM Frequencies

Slower frequencies must be exactly divisible by the clock period of 25 ns. A 25 MHz clock cannot be generated from the 40 MHz clock; the next slowest frequency is 20 MHz.

The NI ELVIS III PWM counters are unsigned 16-bit integers with a range of 0 to 65535. Therefore, using the 40 MHz clock, the slowest frequency is:

With this method, the achievable frequency range is ~610.35 Hz to 40 MHz, where frequencies whose period can be divided by 25 ns can actually be generated.

The NI ELVIS III hardware provides hardware clock dividers to divide reduce the main frequency and generate even slower frequencies. The hardware clock divider is selected by the PWM.x.CS register. With a clock divider of 2, the new slowest achievable frequency is:

The following formula describes possible frequencies:

where is the base clock frequency, is the desired PWM frequency, N is the clock divider being used, and X is the number of counts before changing the signal.

The value of N is determined by the value written to the PWM.x.CS register, and X is the value written to the PWM.x.MAX register.

* Note Attempts to generate frequencies outside the range of 40 Hz to 40 kHz are not supported.

#### SPI Master

#### SPI Configuration Registers (SPI.x.CNFG)

Register list: SPI.A.CNFG, SPI.B.CNFG

Data type: U16

This register configures the SPI master subsystem. It determines the clock divider, frame length, data order, clock polarity, and clock phase settings.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Name | CS1 | CS0 | - | - | - | - | - | - |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | FLEN3 | FLEN2 | FLEN1 | FLEN0 | DORD | CPOL | CPHA | - |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [15:14] - CS : Clock Select

Selects the desired clock divider to be applied to the SPI clock generator which controls the SPI frequency. The CS bits and the CNT register are used together to determine the speed of the SPI transmission. The possible frequencies are shown below:

|  |  |  |
| --- | --- | --- |
| CS1 | CS0 | Clock |
| 0 | 0 | 1x (fclk) |
| 0 | 1 | 2x (fclk / 2) |
| 1 | 0 | 4x (fclk / 4) |
| 1 | 1 | 8x (fclk / 8) |

The base clock frequency (fclk) is set at 40 MHz. You must use this frequency when calculating the value of the CNT register. See the frequency generation section below on how to use the CS register.

* Bits [13:8] - Reserved for future use.

These bits are reserved for future use and should never be written to. Writing a value to these bits is unsupported.

* Bits [7:4] - FLEN : Frame Length

Sets the length, in bits, of the frame to be transmitted or received. A frame size of 4 to 16 is supported. The value to be written must be one less the desired frame length. Therefore, for a frame size of 8 a value of 7 must be written. Values less than 3 are not supported.

FLEN = Desired Frame Length - 1

* Bit [3] - DORD: Data Order

This bit controls the order in which the bits are transmitted. When DORD is 0, the most significant bit of the data frame is transmitted first. When DORD is 1, the least significant bit of the data frame is transmitted first.

* Bit [2] - CPOL: Clock Polarity

This bit controls the idle state of the SPI clock. When this bit is written to one, SPI.CLK is high when idle. When CPOL is written to zero, SPI.CLK is low when idle. The CPOL functionality is summarized below:

|  |  |  |
| --- | --- | --- |
| CPOL | Leading Edge | Trailing Edge |
| 0 | Rising | Falling |
| 1 | Falling | Rising |

* Bit [1] - CPHA: Clock Phase

This bit controls the functionality of the leading and trailing edges of SPI.CLK on the SPI.SDA line. The directions of the leading and trailing edges are controlled by the value of the CPOL bit. The CPHA functionality is summarized below.

|  |  |  |
| --- | --- | --- |
| CPHA | Leading Edge | Trailing Edge |
| 0 | Sample | Setup |
| 1 | Setup | Sample |

* Bit [0] - Reserved for future use.



Figure . SPI Transfer Format with CPHA = 0

Figure . SPI Transfer Format with CPHA = 1

##### SPI Counter Registers (SPI.x.CNT)

Register list: SPI.A.CNT, SPI.B.CNT

Data type: U16

This register controls the maximum value of the SPI counter. This value and the clock divider setting in the CNFG register determine the speed of the SPI transmission. See the frequency generation section below on how to use the CS register.

##### SPI Execute Registers (SPI.x.GO)

Register list: SPI.A.GO, SPI.B.GO

Data type: Boolean

This register starts an SPI data transfer. You only need to write a TRUE value to this register as the register resets to FALSE after the transfer starts. The data transmitted is taken from the DATO register while the data received is placed in the DATI register. During a transfer, the DATI register is invalid until the operation is complete but the value of the DATO register can be changed while a SPI transfer is in progress. When a transfer is in progress, the value of the GO register is ignored. You must wait till the operation is complete before setting the GO register to TRUE again. The status of the SPI transfer can be determined using the STAT register.

##### SPI Status Registers (SPI.x.STAT)

Register list: SPI.A.STAT, SPI.B.STAT

Data type: U8

The register indicates the status of the SPI subsystem.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | - | - | - | BSY |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:1] - Reserved for future use.
* Bit [0] - BSY

If BSY is 1, the SPI subsystem is transferring a frame. If BSY is 0, the SPI subsystem is idle.

##### SPI Data Out Registers (SPI.x.DATO)

Register list: SPI.A.DATO, SPI.B.DATO

Data type: U16

This register holds the data that is sent to the slave device during the next transmission.

The FLEN bits in the SPI.x.CNTL register determines the length of the data transmitted. Bits in the SPI.x.DATO register outside of the specified frame length are ignored. For example, if the SPI.x.DATO register contains 65535 (0xFFFF) and the frame length is 8 bits, only the lower 8 bits are transmitted.

##### SPI Data In Registers (SPI.x.DATI)

Register list: SPI.A.DATO, SPI.B.DATO

Data type: U16

This register holds the data that is received from the slave device during the last transmission. The FLEN bits in the SPI.x.CNTL register determines the length of the data received. The SPI subsystem only attempts to receive the number of bits specified. If the slave device transmits 9 bits per frame but the frame length is set at 8 bits, the last bit is ignored and the DATI register contains only the 8 bits received. On the next SPI transfer, the slave may try to send the last bit from the previous transmission.

##### SPI Frequency Generation

The NI ELVIS III hardware runs on a 40 MHz clock, which means the time between clock cycles is 25 ns. The NI ELVIS III can generate slower SPI frequencies by counting and changing the output on intervals of rising clock edges. The NI ELVIS III can generate SPI frequencies between 40 Hz and 4 MHz. You must downsample the 40 MHz clock to generate a slower frequency. For example, the following figure shows the generation of 20 MHz and 10 MHz clocks from a 40 MHz clock by changing the output every rising edge or every other rising edge, respectively.



Figure . Generating Slower SPI Frequencies

Slower frequencies must be exactly divisible by the clock period of 25 ns. A 25 MHz clock cannot be generated from the 40 MHz clock; the next slowest frequency is 20 MHz.

The NI ELVIS III SPI counters are unsigned 16-bit integers with a range of 0 to 65535. Therefore, using the 40 MHz clock, the slowest frequency is:

Using this method, the achievable frequency range is ~610.35 Hz to 40 MHz, where frequencies whose period can be divided by 25ns can actually be generated.

In order for generating even slower frequencies, the NI ELVIS III hardware provides a series of clock dividers (N). The clock dividers function as described above, where the base frequency is divided into even numbers (2, 4, 8, etc) and the generated clock is used to increment the counter. With a clock divider of 2, and a U16 counter, the new slowest achievable frequency is:

The possible SPI frequencies that can be generated are based on the following equation:

where is the base clock frequency, is the desired SPI frequency, N is the clock divider being used, and X is the number of counts before changing the signal.

The value of N is determined by the value written to the CS bits in the CNTL register, and X is the value written to the CNT register.

* Note Attempts to generate frequencies outside the range of 40 Hz to 4 MHz are not supported.

#### Encoder

The quadrature encoder block counts the number of steps that an encoder makes along its rotation. The angular change per step is determined by the resolution of the encoder being used. When the encoder is going forward, the count value is incremented. When the encoder is moving backwards, the count value is decremented. There are two modes that are supported by the implemented encoder (ENC) subsystem. In the step and direction mode, the direction signal indicates the direction of rotation where a low signal means forward and a high signal means backward. The count value changes on every rising edge of the step signal. In the quadrature phase mode, the encoder generates two signals called Phase A and Phase B, which are two square waves that are 90 degrees out of phase with each other. In general, when Phase A is leading Phase B, the encoder counter is counting up, and when Phase B leads Phase A, the encoder counter is counting down. The count value is changed on every change of Phase A or Phase B. The following figure shows a waveform with the Phase A and Phase B signals and the equivalent step (clk) and direction (dir) signals.



Figure . A Waveform with Phase A, Phase B, Step (CLK), and Direction (DIR) Signals

##### Encoder Configuration Registers (ENC.x.CNFG)

Register list: ENC.A\_[0:9].CNFG, ENC.B\_[0:9].CNFG

Data type: U8

This register configures the encoder subsystem.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | COVR | CERR | MODE | RST | EN |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:5] - Reserved for future use.
* Bit [4] - COVR: Clear Overflow

Clears all the overflow related flags (UOVR, SOVR, UOERR, SOERR) in the ENC Status Register (STAT). The flags are cleared on the rising edge on this signal, which is when the value goes from 0 to 1. It should be manually reset to 0 after use.

* Bit [3] - CERR: Clear Error

Clears the error flag (ERR) in the ENC Status Register (STAT). The flag is cleared on the rising edge on this signal, which is when the value goes from 0 to 1. It should be manually reset to 0 after use.

* Bit [2] - MODE: Signal Mode

The mode of operation of the ENC block. When MODE is written with a 0, it operates in quad phase mode. When MODE is written with a 1, it operates in step and direction mode.

* Bit [1] - RST: Reset

Resets the value of the ENC counter to 0. The counter remains at 0 as long as this bit has a value of 1.

* Bit [0] - EN: Enable

Enables the ENC block. When it is written with 0, the ENC block is disabled and the count value and direction flag do not change. When it is written with a 1, the block is enabled.

##### Encoder Status Registers (ENC.x.STAT)

Register list: ENC.A\_[0:9].STAT, ENC.B\_[0:9].STAT

Data type: U8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | SOERR | UOERR | SOVR | UOVR | ERR | DIR |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Configures the encoder subsystem for the desired behavior.

* Bits [7:6] - Reserved for future use.
* Bit [5] - SOERR: Signed Overflow Error

Indicates that a signed overflow error has occurred. When this bit is 1, a signed overflow occurred while the SOVR flag is already set as 1. This indicates that a signed overflow occurred before the SOVR flag is cleared and therefore, the state of the SOVR flag cannot be trusted as it is not possible to know how many times overflow has occurred. This bit remains at 1 until it is cleared by writing a 1 to the COVR bit in the CNFG register.

* Bit [4] - UOERR: Unsigned Overflow Error

Indicates that an unsigned overflow error has occurred. The bit is set to 1 when an unsigned overflow occurs while the UOVR flag is already set as 1. This indicates that an unsigned overflow occurred before the UOVR flag was cleared and therefore, the state of the UOVR flag cannot be trusted as it is not possible to know how many times overflow has occurred. This bit remains at 1 until it is cleared by writing a 1 to the COVR bit in the CNFG register.

* Bit [3] - SOVR: Signed Overflow

Indicates that a signed overflow has occurred. The counter value is stored as an unsigned 32-bit value which can represent both a signed or unsigned number. If you want to treat the stored value as a signed number then use this overflow flag and ignore the UOVR flag. When this bit is 1, the counter value has gone from the maximum value (2147483647) to the minimum value (-2147483648) or has gone from the minimum value to the maximum value. When this bit is 0, no overflow has occurred. This bit remains at 1 until it is cleared by writing a 1 to the COVR bit in the CNFG register.

* Bit [2] - UOVR: Unsigned Overflow

Indicates that an unsigned overflow has occurred. The counter value is stored as an unsigned 32-bit value that can represent both a signed or unsigned number. If you want to treat the stored value as an unsigned number, use this overflow flag and ignore the SOVR flag. When this bit is 1, the counter value has gone from the maximum value (4294967296) to 0 or has gone from 0 to the maximum value. When this bit is 0, no overflow has occurred. This bit remains at 1 until it is cleared by writing a 1 to the COVR bit in the CNFG register.

* Bit [1] - ERR: Error

Indicates that an error has occurred when operating in quad phase mode. This bit will never be 1 while operating in step and direction mode. A value of 1 indicates that an error occurs. This is usually caused by the values of both the Phase A and Phase B signals changing at the same time. When this bit is 1, the counter value and direction bit do not update based on the encoder input but hold the last valid value. This bit remains at 1 until it is cleared by writing a 1 to the CERR bit in the CNFG register.

* Bit [0] - DIR: Direction

Indicates the last direction of the last change to the encoder counter value. A value of 0 indicates that the encoder counter was incremented while a value of 1 indicates that the encoder counter was decremented.

##### Encoder Counter Value Registers (ENC.x.CNTR)

Register list: ENC.A\_[0:9].CNTR, ENC.B\_[0:9].CNTR

Data type: U32

Unsigned range: 0 to 4294967296

Signed range: -2147483648 to 2147483647

The number of steps that the encoder has gone through based on the value of the MODE bit in CNFG. In quad phase mode, the counter value increments when the Phase A leads Phase B and decrements when Phase B leads Phase A. In step and direction mode, the counter increments when the direction input is low and decrements when the direction input is high. Both signed and unsigned numbers are stored as unsigned 32-bit values so if the user wants to treat the value as a signed number they must convert it before they use the value.

#### I2C

##### I2C Configuration Registers (I2C.x.CNFG)

Register list: I2C.A.CNFG, I2C.B.CNFG

Data type: U8

This register enables or disables the I2C subsystem.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | - | - | - | MSTREN |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:1] - Reserved for future use.
* Bit [0] - MSTREN: Enable or disable I2C functionality.

##### I2C Slave Address Registers (I2C.x.ADDR)

Register list: I2C.A.ADDR, I2C.B.ADDR

Data type: U8

This register sets the address and transmission direction of the slave device.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | SA6 | SA5 | SA4 | SA3 | SA2 | SA1 | SA0 | R/S |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:1] - SA : Slave address

Specifies the 7-bit address for the slave device that is being communicated with.

* Bit [0] - R/S : Receive/send

Specifies if the next transmission operation to be completed is a send or receive operation.

* 0: Send
* 1: Receive

##### I2C Counter Registers (I2C.x.CNTR)

Register list: I2C.A.CNTR, I2C.B.CNTR

Data type: U8

Specifies the counter value the I2C subsystem must use to generate the clock during a send or receive operation.

The value of the CNTR register can be calculated using the following equation:

where is the desired I2C transmission frequency and is the base clock frequency of the hardware (40 MHz).

For example, for a standard-mode transmission of 100 kbps:

Since

* Note The actual frequency of the I2C clock depends on the rise and fall times of your circuit. Using the previous equation guarantees that the frequency of the generated clock signal complies with the I2C specification for standard and fast modes, regardless of the connected circuit.

##### I2C Data Out Registers (I2C.x.DATO)

Register list: I2C.A.DATO, I2C.B.DATO

Data type: U8

This register holds the data that is sent to the slave device during the next send operation.

##### I2C Data In Registers (I2C.x.DATI)

Register list: I2C.A.DATI, I2C.B.DATI

Data type: U8

This register holds the data that is received from the slave device during the last receive operation.

##### I2C Status Registers (I2C.x.STAT)

Register list: I2C.A.STAT, I2C.B.STAT

Data type: U8

This register indicates the current status of the I2C subsystem.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | BUSBSY | INUSE | DATNAK | ADRNAK | ERR | BSY |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:6] - Reserved for future use.
* Bit [5] - BUSBSY: I2C bus is busy.

Indicates if the I2C bus is currently busy. The bit is set to 1 when the bus is busy. The bit is set to 0 when the bus is free.

* Bit [4] - INUSE: I2C subsystem is in use.

Indicates if the I2C subsystem is currently in use. The bit is set to 1 when the subsystem is in use. The bit is set to 0 when the subsystem is free.

* Bit [3] - DATNAK: Data Not Acknowledge (NAK) received.

Indicates if a NAK is received from the slave after the last data transmission. The bit is set to 1 when a NAK is received. The bit is set to 0 when a NAK is not received, or an ACK is received.

* Bit [2] - ADRNAK: Address Not Acknowledge (NAK) received.

Indicates that a NAK is received from the slave after the last address transmission. The bit is set to 1 when a NAK is received. The bit is set to 0 when a NAK is not received (an ACK is received).

* Bit [1] - ERR: Error

Indicates that an error occurs during the last transmission. This could be either a NAK is received on the last data transmission or on the last address transmission. It is provided for convenience so that both the ADRNAK and DATNAK bits don't have to be checked every time. When the value = 0 no error occurred during the last operation, when the value = 1 an error occurred during the last operation. If the value = 1 the DATNAK and ADRNAK bits must be checked to see the cause of the error.

* Bit [0] - BSY: Busy

Indicates if the I2C subsystem is busy performing an operation. The value is 1 when the subsystem is busy. The value is 0 when the subsystem is not busy.

Table . I2C busbsy/inuse/bsy combinations

|  |  |  |  |
| --- | --- | --- | --- |
| BUSBSY | INUSE | BSY | Interpretation |
| 0 | 0 | 0 | The I2C bus is free and control can be taken by the I2C subsystem. |
| 1 | 0 | 0 | The I2C bus is busy and in use by some other master connected to the bus. (Not Supported) |
| 1 | 1 | 0 | The I2C bus is busy and in use by the I2C subsystem. The subsystem is not busy so the I2C subsystem is either in the TX IDLE, or RX IDLE state. |
| 1 | 1 | 1 | The I2C bus is busy, in use by the I2C subsystem, and the subsystem is executing some operation. This could be a START, REPEATED START, TX, RX, or STOP. |
| All other combinations have no real-world interpretation and should never occur. | | | |

##### I2C Control Registers (I2C.x.CNTL)

Register list: I2C.A.CNTL, I2C.B.CNTL

Data type: U8

This register controls the next operation to be performed by the I2C subsystem. Some of the operations supported by the I2C subsystem can be independent of each other. As such they must be configured to occur before the operation is started. See Table 1 for a list of valid and invalid values for the CNTL register.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | ACK | STOP | START | TX/RX |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:4] - Reserved for future use.
* Bit [3] - ACK: Data Acknowledge Enable

When receiving data from the slave, this bit specifies if an ACK or a NAK must be generated after the data byte is received. When sending data to the slave device, this bit is ignored.

* Note Sending an ACK after the last data byte received (before generating a STOP condition) violates the I2C standard.

See field decoding in Table 9.

* Bit [2] - STOP: Generate the STOP condition.

Specifies if the I2C subsystem generates a STOP condition after completing the operation. When the STOP condition is generated, control of the I2C bus is released.

* Note When receiving data from the slave, the STOP bit and the ACK bit must never be true at the same time.

See field decoding in Table 9.

* Bit [1] - START: Generate the START condition.

Specifies if the I2C controller generates a START or REPEATED START condition. A START condition must be generated when the I2C subsystem does not have control of the bus and wants to get control. A REPEATED START condition must be generated when the I2C subsystem already has control of the bus and wants to either change the addressed slave device or change the direction of the transmission to the same slave device.

* Note When the START bit is TRUE, the TX/RX bit must also be TRUE.

See field decoding in Table 9.

* Bit [0] - TX/RX: Transmit or receive a data byte.

Specifies if the I2C controller sends a data byte to or receive a data byte from the slave device. The direction of the transmission (whether it is a send or receive operation) depends on the value of the R/S bit in the I2C.x.ADDR register. This bit can be set on its own (when in send mode) or in conjunction with the ACK bit (when in receive mode) to continually send or receive data from the slave without having to generate START or STOP conditions.

See field decoding in Table 9.

Table . I2C control registers possible combinations

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| State | R/S | ACK | STOP | START | TX/RX | I2C Operation | |
| IDLE | 0 | X | 0 | 1 | 1 | Generate START, Send Address, Receive Address ACK, Send Data, Receive Data ACK, and go to TX IDLE state. | |
|  | 0 | X | 1 | 1 | 1 | Generate START, Send Address, Receive Address ACK, Send Data, Receive Data ACK, Generate STOP, and return to IDLE state. | |
|  | 1 | 0 | 0 | 1 | 1 | Generate START, Send Address, Receive Address ACK, Receive Data, Send Data NAK, and go to RX IDLE state. | |
|  | 1 | 0 | 1 | 1 | 1 | Generate START, Send Address, Receive Address ACK, Receive Data, Send Data NAK, and return to IDLE state. | |
|  | 1 | 1 | 0 | 1 | 1 | Generate START, Send Address, Receive Address ACK, Receive Data, Send Data ACK, and go to RX IDLE state. | |
|  | 1 | 1 | 1 | 1 | 1 | Illegal. (Master cannot transmit an ACK before generating a STOP.) | |
|  | All other operations are non-operations. | | | | | NOP | |
|  | | | | | | | |
| TX IDLE | X | X | 0 | 0 | 1 | | Send Data (to previously addressed slave), Receive Data ACK, and return to TX IDLE state. |
|  | X | X | 1 | 0 | 0 | | Generate STOP, and go to IDLE state. |
|  | X | X | 1 | 0 | 1 | | Send Data, Receive Data ACK, Generate STOP, and go to IDLE state. |
|  | 0 | X | 0 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Send Data, Receive Data ACK, and return to TX IDLE state. |
|  | 0 | X | 1 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Send Data, Receive Data ACK, Generate STOP, and go to IDLE state. |
|  | 1 | 0 | 0 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Receive Data, Send Data NAK, and go to RX IDLE state. |
|  | 1 | 0 | 1 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Receive Data, Send Data NAK, Generate STOP, and go to IDLE state. |
|  | 1 | 1 | 0 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Receive Data, Send Data ACK, and go to RX IDLE state. |
|  | 1 | 1 | 1 | 1 | 1 | | Illegal. (Master cannot transmit an ACK before generating a STOP.) |
|  | All other operations are non-operations. | | | | | | NOP |
|  | | | | | | | |
| RX IDLE | X | 0 | 0 | 0 | 1 | | Receive Data (from previously addressed slave), Send Data NAK, and return to RX IDLE state. |
|  | X | X | 1 | 0 | 0 | | Generate STOP, and go to IDLE state. |
|  | X | 0 | 1 | 0 | 1 | | Receive Data, Send Data NAK, Generate STOP, and return to IDLE state. |
|  | X | 1 | 0 | 0 | 1 | | Receive Data, Send Data ACK, and return to RX IDLE state. |
|  | X | 1 | 1 | 0 | 1 | | Illegal. (Master cannot transmit an ACK before generating a STOP.) |
|  | 0 | X | 0 | 1 | 1 | | Generate REPEATED START, Send Address, RX Address ACK, Send Data, RX Data ACK, and go to TX IDLE state. |
|  | 0 | X | 1 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Send Data, Receive Data ACK, Generate STOP, and go to IDLE state. |
|  | 1 | 0 | 0 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Receive Data, Send Data NAK, and return to RX IDLE state. |
|  | 1 | 0 | 1 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Receive Data, Send Data NAK, Generate STOP, and go to IDLE state. |
|  | 1 | 1 | 0 | 1 | 1 | | Generate REPEATED START, Send Address, Receive Address ACK, Receive Data, Send Data ACK, and return to RX IDLE state. |
|  | 1 | 1 | 1 | 1 | 1 | | Illegal. (Master cannot transmit an ACK before generating a STOP.) |
|  | All other operations are non-operations. | | | | | | NOP |

##### I2C Execute Registers (I2C.x.GO)

Register list: I2C.A.GO, I2C.B.GO

Data type: Boolean

This register causes the operation specified in the I2C.x.CNTL register to begin. When an operation is written to the CNTL register, it does not start until the GO bit is strobed. The user only has to write a TRUE to this register as the register resets to FALSE after the I2C operation has started. Table 1 shows how to set the CNTL register for the different possible I2C operations.

#### Timer Interrupt

* Note Timer Interrupt reserves the IRQ Number 0 while the other interrupts use values within the range [1, 8].

##### Timer Read Register (IRQ.TIMER.READ)

Register list: IRQ.TIMER.READ

Data type: U32

This register contains the remaining time before the FPGA timer elapses. The FPGA timer triggers an interrupt request (IRQ) when IRQ.TIMER.READ counts down to zero. If IRQ.TIMER.READ is not zero, the FPGA timer counts down to one per microsecond.

##### Timer Write Register (IRQ.TIMER.WRITE)

Register list: IRQ.TIMER.WRITE

Data type: U32

This register attempts to reset the remaining time of the FPGA timer. The value of this register does not take effect until IRQ.TIMER.SETTIME is strobed.

##### Timer Set Time Register (IRQ.TIMER.SETTIME)

Register list: IRQ.TIMER.SETTIME

Data type: Boolean

This register is the toggle to set the FPGA timer with the value in the IRQ.TIMER.WRITE register. The default value is FALSE. The write operation starts when you write TRUE to this register. After one iteration of writing, the register resets to FALSE automatically.

#### Analog Input Interrupt

##### Analog IRQ Threshold Register (IRQ.AI\_xx.THRESHOLD)

Register list: IRQ.AI\_A\_0.THRESHOLD, IRQ.AI\_A\_1.THRESHOLD

Data type: U16

This register sets the value of the analog input threshold. When an analog input signal crosses the threshold, an interrupt is triggered. Each channel has one analog IRQ threshold register. The value is given in bits/volt. You must scale and offset the threshold before you use the value. You can get the scaling weight and offset from the analog scaling weight and analog scaling offset constants. Refer to the Analog Value Registers section for how to apply scaling to the value.

##### Analog IRQ Hysteresis Register (IRQ.AI\_xx.HYSTERESIS)

Register list: IRQ.AI\_A\_0.HYSTERESIS, IRQ.AI\_A\_1.HYSTERESIS

Data type: U16

This register sets the value of hysteresis or the window size. Hysteresis adds a window above or below the analog IRQ threshold to reduce false triggering due to noise. Each channel has one analog IRQ hysteresis register. The value is given in bits/volt. You must scale and offset the hysteresis before you use the value. You can get the scaling weight and offset from the analog scaling weight and analog scaling offset constants. Refer to the Analog Value Registers section for how to apply scaling to the value.

##### Analog IRQ Configuration Register (IRQ.AI\_ XX.CNFG)

Register list: IRQ.AI\_A\_3:0.CNFG

Data type: U8

This register contains the interrupt type and enabling configuration of the analog IRQ, as shown in the following table.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | IRQ.AI\_A\_1.Type | IRQ.AI\_A\_1.ENA | IRQ.AI\_A\_0.Type | IRQ.AI\_A\_0.ENA |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:4] - Reserved for future use.
* Bit [3] - IRQ.AI\_A\_1. Type.

Specifies the interrupt type of the channel. If the bit is set to 1, the AI1 channel on connector A checks AI interrupts on a rising edge of the analog input signal. If the bit is set to 0, the AI1 channel on connector A checks AI interrupts on a falling edge of the analog input signal.

* Bit [2] - IRQ.AI\_A\_1. ENA.

Enables the settings of the analog input interrupt channel. If the bit is set to 1, the AI1 channel on connector A starts checking AI interrupts based on the settings. If the bit is set to 0, the AI1 channel on connector A stops checking AI interrupts. The default value of the bit is 0 when the NI ELVIS III device is powered on.

* Bit [1] - IRQ.AI\_A\_0. Type.

Specifies the interrupt type of the channel. If the bit is set to 1, the AI0 channel on connector A checks AI interrupts on a rising edge of the analog input signal. If the bit is set to 0, the AI0 channel on connector A checks AI interrupts on a falling edge of the analog input signal.

* Bit [0] - IRQ.AI\_A\_0. ENA

Enables the settings of the analog input interrupt channel. If the bit is set to 1, the AI0 channel on connector A starts checking AI interrupts based on the settings. If the bit is set to 0, the AI0 channel on connector A stops checking AI interrupts. The default value of the bit is 0 when the NI ELVIS III device is powered on.

##### Analog IRQ Number Register (IRQ.AI\_xx.NO)

Register list: IRQ.AI\_A\_0.NO, IRQ.AI\_A\_1.NO

Data type: U8

This register specifies the identifier of the interrupt. Each channel has one analog IRQ number register. The IRQ number ranges from 1 to 8 on FPGA. The number is shared with analog, digital and button interrupts.

#### Digital Input Interrupt

##### Digital Enabling Register (IRQ.DIO\_xx.ENA)

Register list: IRQ.DIO\_A\_7:0.ENA

Data type: U8

This register enables the settings of digital input interrupt channels.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | IRQ.DIO\_A\_3.ENA | IRQ.DIO\_A\_2.ENA | IRQ.DIO\_A\_1.ENA | IRQ.DIO\_A\_0.ENA |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:4] - Reserved for future use.
* Bits [3:0] - IRQ.DIO\_A\_3:0.ENA.

Each bit in **Bits [3:0]** controls the settings of one channel. For example, in IRQ.DIO\_A\_3:0.ENA, bit 0 controls A/DIO0, while bit 3 controls A/DIO3. If the bit is set to 1, the channel starts checking DI interrupts based on the settings. If the bit is set to 0, the channel stops checking the interrupt. The default value of the bit is 0 when the NI ELVIS III device is powered on.

##### Digital Rising Register (IRQ.DIO\_xx.RISE)

Register list: IRQ.DIO\_A\_7:0.RISE

Data type: U8

This register enables the digital rising edge interrupt of digital input interrupt channels.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | IRQ.DIO\_A\_3.RISE | IRQ.DIO\_A\_2.RISE | IRQ.DIO\_A\_1.RISE | IRQ.DIO\_A\_0.RISE |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:4] - Reserved for future use.
* Bits [3:0] - IRQ.DIO\_A\_3:0.RISE.

Each bit in **Bits [3:0]** controls one channel. For example, in IRQ.DIO\_A\_3:0.RISE, bit 0 controls A/DIO0, while bit 3 controls A/DIO3. If the bit is set to 1, the channel checks DI interrupts on a rising edge of the digital input signal. If the bit is set to 0, the channel does not check the rising edge of the digital input signal.

##### Digital Falling Register (IRQ.DIO\_xx.FALL)

Register list: IRQ.DIO\_A\_7:0.FALL

Data type: U8

This register enables the digital falling edge interrupt of digital input interrupt channels.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | - | - | - | - | IRQ.DIO\_A\_3.FALL | IRQ.DIO\_A\_2.FALL | IRQ.DIO\_A\_1.FALL | IRQ.DIO\_A\_0.FALL |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* Bits [7:4] - Reserved for future use.
* Bits [3:0] - IRQ.DIO\_A\_3:0.FALL.

Each bit in **Bits [3:0]** controls one channel. For example, in IRQ.DIO\_A\_3:0.FALL, bit 0 controls A/DIO0, while bit 3 controls A/DIO3. If the bit is set to 1, the channel checks DI interrupts on a falling edge of the digital input signal. If the bit is set to 0, the channel does not check the falling edge of the digital input signal.

##### Digital IRQ Number Register (IRQ.DIO\_xx.NO)

Register list: IRQ.DIO\_A\_0.NO, IRQ.DIO\_A\_1.NO, IRQ.DIO\_A\_2.NO, IRQ.DIO\_A\_3.NO

Data type: U8

This register specifies the identifier of the interrupt. Each channel has one digital IRQ number register. The IRQ number ranges from 1 to 8 on FPGA. The number is shared with analog, digital and button interrupts.

##### Digital Count Register (IRQ.DIO\_A\_0.CNT)

Register list: IRQ.DIO\_A\_0.CNT, IRQ.DIO\_A\_1.CNT, IRQ.DIO\_A\_2.CNT, IRQ.DIO\_A\_3.CNT

Data type: U32

This register specifies the number of edges for triggering one interrupt. The interrupt is triggered every time the edges count reaches the number. Each channel has one digital count register.

#### Button Interrupt

##### Button Enabling Register (IRQ.DI\_BTN.ENA)

Register list: IRQ.DI\_BTN.ENA

Data type: Boolean

This register enables the settings of the button interrupt channel. If the bit is set to 1, the channel starts checking interrupts based on the settings. If the bit is set to 0, the channel stops checking the interrupt. The default value of the bit is 0 when the NI ELVIS III device is powered on.

##### Button Rising Register (IRQ.DI\_BTN.RISE)

Register list: IRQ.DI\_BTN.RISE

Data type: Boolean

This register enables the rising edge interrupt of the button channel. If the bit is set to 1, the channel checks interrupts on a rising edge of the button state. If the bit is set to 0, the channel does not check the rising edge of the button state.

##### Button Falling Register (IRQ.DI\_BTN.FALL)

Register list: IRQ.DI\_BTN.FALL

Data type: Boolean

This register enables the falling edge interrupt of the button channel. If the bit is set to 1, the channel checks interrupts on a falling edge of the button state. If the bit is set to 0, the channel does not check the falling edge of the button state.

##### Button IRQ Number Register (IRQ.DI\_BTN.NO)

Register list: IRQ.DI\_BTN.NO

Data type: U8

This register specifies the identifier of the interrupt. The available range of IRQ number is 1~8 on FPGA. The number is shared with analog, digital and button interrupts.

##### Button Count Register (IRQ.DI\_BTN.CNT)

Register list: IRQ.DI\_BTN.CNT

Data type: U32

This register specifies the number of edges for triggering one interrupt. The interrupt is triggered every time the edges count reaches the number.