roboRIO Shipping Personality 1.0 Reference

This document contains reference information about the roboRIO shipping personality.

Contents

NATIONAL INSTRUMENTS

Introduction

The roboRIO shipping personality provides support for the following peripherals:

- Onboard devices (accelerometer, 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)
- Robot signal light (RSL)
- Relay

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

Register Naming Convention

Registers follow a naming scheme as described below:

```
Peripheral Type.Channel Name.Property Name
```

Note When you program in C language, register names must not contain periods, colons, or spaces.

Peripheral Type

Possible values = {ACC, AI, AO, DIO, DI, DO, PWM, I2C, SPI, ENC, IRQ, RSL, RELAY, SYS}

Short Name	Full Name
ACC	Accelerometer
AI	Analog input
AO	Analog output
DIO	Digital input/output
DI	Digital input
DO	Digital output
PWM	Pulse-width modulation
I2C	Inter-integrated circuit
SPI	Serial peripheral interface
ENC	Encoder
IRQ	Interrupt request
RSL	Robot signal light
RELAY	Relay
SYS	System

Note SYS is a reserved value used for a special purpose system register. The system registers may or may not be related to a specific peripheral.

Channel Name

The Channel Name is a combination of the connector designation and its numeric enumeration or range enumeration. An underscore (_) separates the channels connector designation and its enumeration. The enumeration is omitted if only one channel of that type is available on the connector.

Example: AI_0 indicates the channel is the first channel on Connector ANALOG IN. B_7:0 indicates that the register corresponds to channels 0 to 7 on MXP Connector B.

Property Names

The following tables list all possible property names. Some properties are only applicable to a single peripheral while others may be used across multiple peripherals.

Table 1. Inputs (controls)

Short Name	Long Name	Comment
DIR	Direction	Controls the direction of the DIO pins.
OUT	Output	The value of the signal to be set at the DIO output pins.
CNFG	Configuration	Used to configure a peripheral. Usually used to indicate some setting that does not change after initialization or at least changes infrequently.
CNTL	Control	Used to control a peripheral. Usually used for control bits that are used often at run time, such as start or clear bits.
CMP	Compare	The compare value. Used by peripherals that compare one value to another value.
MAX	Maximum	The maximum value for some internal counter.
CS	Clock Select	The desired clock to be used.
ADDR	Address	The address to be used.
DATO	Data Out	The data to be sent to an attached device.
CNT	Count	The count value to be used by an internal counter. Usually used to control the rate of operation of a subsystem or signal generation.
GO	Go	Used to start an operation on a peripheral.
VAL	Value	The value written to a subsystem/onboard device. Also used as an output.

Table 2. Outputs (indicators)

Short Name	Long Name	Comment
IN	Input	The value currently present at the DIO input lines.
STAT	Status	The status of a peripheral.
CNTR	Counter	The value of some counter.
DATI	Data In	The data received from an attached device.
VAL	Value	The value read from a subsystem/onboard device. Also used as an input.
WGHT	Weight	The scaling weight to convert to/from physical units.
OFST	Offset	The offset from zero.
RDY	Ready	The status of a subsystem.
READ	Read	The remaining time before the FPGA timer elapses.
WRITE	Write	The elapse time to be set to the FPGA timer.
SETTIME	Set Time	The toggle to overwrite the elapse time in the FPGA timer.
ENA	Enable	Enables the setting of a peripheral.
RISE	Rise	Enables the rising edge interrupt.
FALL	Fall	Enables the falling edge interrupt.
NO	Number	The identifier of the interrupt.
THRESHOLD	Threshold	The value that triggers the analog input interrupts.
HYSTERESIS	Hysteresis	The window size for threshold to reduce noise.

Host Synchronization Registers (SYS.x.RDY)

Register list: SYS.AI.RDY, SYS.AO.RDY, SYS.ACC.RDY, SYS.AI_SCALE.RDY, SYS.AO_SCALE.RDY, SYS.RDY

Data type: Boolean

The host synchronization registers determine the status of specific hardware subsystems or the entire roboRIO hardware. These registers change to TRUE when the corresponding subsystem is ready. When all subsystems are ready, the value of the SYS.RDY register changes to TRUE. If you use a subsystem before its RDY register is TRUE, you might encounter unexpected behavior.

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.

myRIO Expansion Port B (MXP B)

Register list: SYS.SELECTB

Data type: U8

Use this register to select the function of the pins on MXP Connector B, as shown in the following table.

Bit	7	6	5	4	3	2	1	0
Name	I2C	-	ENC	PWM2	PWM1	PWM0	SPI1	SPI0
Initial Value	0	0	0	0	0	0	0	0

- **Bit [7]** - I2C

  Switches between the DIO and I2C functionality on channels DIO15:14.

  - 1 - I2C enabled, DIO15:14 disabled.
  - 0 - I2C disabled, DIO15:14 enabled.
- **Bit [6]** - Reserved for future use.
- **Bit [5]** - ENC

  Switches between the DIO and Encoder functionality on channels DIO12:11.

  - 1 - Encoder enabled, DIO12:11 disabled.
  - 0 - Encoder disabled, DIO12:11 enabled.
- **Bits [4:2]** - PWM [2:0]

  Switches between the DIO and PWM functionality on channels DIO10, DIO9, and DIO8, respectively.

  - 1 - PWM enabled, DIO disabled.
  - 0 - PWM disabled, DIO enabled.

- **Bits [1:0]** - SPI

  Switches between the DIO and SPI functionality of channels DIO7:5.

  - 11 - SPI enabled, DIO7:5 disabled.
  - 10 - Transmit only SPI: MOSI, CLK enabled, MISO disabled. DIO7, DIO5 disabled, DIO6 enabled.
  - 01 - Receive only SPI: MISO, CLK enabled, MOSI disabled. DIO7:6 disabled, DIO5 enabled.
  - 00 - SPI disabled, DIO7:5 enabled.

roboRIO DIO Port

Register list: SYS.SELECTDIO

Data type: U8

Use this register to select the function of the pins on Connector DIO, as shown in the following table.

Bit	7	6	5	4	3	2	1	0
Name	-	-	-	ENC4	ENC3	ENC2	ENC1	ENC0
Initial Value	0	0	0	0	0	0	0	0

- **Bits [7:5]** - Reserved for future use.
- **Bits [4:0]** - ENC [4:0]

  Switches between the DIO and Encoder functionality of channels DIO [1:0], DIO [3:2], DIO [5:4], DIO [7:6] and DIO [9:8], respectively.

  - 1 - Encoder enabled, DIO disabled.
  - 0 - Encoder disabled, DIO enabled.

Onboard Device Registers

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

LEDs (DO.LED 5:0)

Data type: U8

Bit	7	6	5	4	3	2	1	0
Name	-	-	Mode LED Red	Mode LED Green	Comm LED Red	Comm LED Green	Radio LED Red	Radio LED Green
Initial Value	0	0	0	0	0	0	0	0

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

- **Bits [7:6]** - Reserved for future use.
- **Bits [5:0]** - RADIO LED (Green/Red), COMM LED (Green/Red) and MODE LED (Green/Red)

  The desired state of onboard RADIO LED, COMM LED and MODE LED.

Button (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.

Accelerometer Value Registers (ACC.x.VAL)

Register list: ACC.X.VAL, ACC.Y.VAL, ACC.Z.VAL

Data type: U16

These registers contain the values read from the onboard accelerometer. The values are provided in raw values and must be scaled to get the correct values in g-force.

Accelerometer Scaling Weight

The value of the accelerometer scaling weight is given in (unsigned) bits/g-force, which is the number of bits that changes for one g-force change in the reading. The value can be changed to g-force/bit by inverting the value of the accelerometer scaling weight and then multiplying by the signed integer value of ACC.x.VAL. The value of the accelerometer scaling weight is a constant, which applies to all accelerometer channels. Its value is 256.

The following table gives examples of calculating the acceleration using the accelerometer scaling weight and ACC.x.VAL.

Table 3. Example acceleration calculations

Register Value	Acceleration (g-force)
0	0
1	$\frac{1}{\text{accelerometer scaling weight}}$
2	$\frac{1}{\text{accelerometer scaling weight}} \times 2$
65536	$\frac{1}{\text{accelerometer scaling weight}} \times -1$

Note When you program in C language, register names must not contain periods, colons, or spaces.

Analog Value Registers (AI.xx.VAL or AO.xx.VAL)

Register list:

Analog Input
Unsigned values: AI.B_0.VAL, AI.B_1.VAL, AI.B_2.VAL, AI.B_3.VAL, AI.AI_0.VAL, AI. AI _1.VAL, AI. AI _2.VAL, AI.AI_3.VAL
Analog Output
Unsigned values: AO.B_0.VAL, AO.B_1.VAL

Data type: U16

This register contains the value read by the analog input channel or written from the analog output channel. Each channel has one VAL register. The value is given in bits/volt. You must scale and offset the value to volts (input) or scale and offset the value from volts (output) before you use the value. If a channel supports negative voltages, when reading from an input, the value read must be converted to a signed value before applying any scaling, and when writing to an output, the value must be converted to an unsigned value after applying the appropriate scaling. See Tables 6 and 7 for examples of how to apply scaling to the value.

Analog Scaling Weight

The value of analog scaling weight represents the change in the voltage for every one bit change in the value. Each channel has one analog scaling weight and the value is given in (unsigned) nanovolts. For MXP Connector B, and Connector ANALOG IN, the value of analog scaling weight is 1220703. Refer to Tables 6 and 7 for examples of how to use the analog scaling weight for scaling.

Analog Scaling Offset

The value of analog scaling offset represents the offset between the input/output value and the actual voltage. Each channel has one analog scaling offset and the value is given in (unsigned) nanovolts. Therefore, you must convert the value read from each channel to a signed value before you use the value. The value of analog scaling offset is 0. Refer to Table 6 and 7 for examples of how to use the analog scaling offset for scaling.

Table 4 and 5 list the weight and offset constants for all the Analog Value Register.

Table 4. Analog input scaling constant

Channel	WEIGHT (nanovolts)	OFFSET (nanovolts)	Sign
AI.B_0	1220703	0	Unsigned (U16)
AI.B_1	1220703	0	Unsigned (U16)
AI.B_2	1220703	0	Unsigned (U16)
AI.B_3	1220703	0	Unsigned (U16)
AI.AI_0	1220703	0	Unsigned (U16)
AI.AI_1	1220703	0	Unsigned (U16)
AI.AI_2	1220703	0	Unsigned (U16)
AI.AI_3	1220703	0	Unsigned (U16)

Table 5. Analog output scaling constant

Channel	WEIGHT (nanovolts)	OFFSET (nanovolts)	Sign
AO.B_0	1220703	0	Unsigned (U16)
AO.B_1	1220703	0	Unsigned (U16)

Table 6. Example calculations for scaling AI In values

Value	Formula	AI.B_0
Weight (volts)	$\frac{\text{(Double)WEIGHT}}{10^9}$	$\frac{\text{(Double)1220703}}{10^9} = 0.001220703$
Offset (volts)	$\frac{\text{(Double)(I32)OFFSET}}{10^9}$	$\frac{\text{(Double)(I32)0}}{10^9} = 0$
Voltage	((Sign)Register Value × WEIGHT) + OFFSET	((U16)AI.A_0.VAL×0.001220703)+0

Table 7. Example calculations for scaling AO In values

Value	Formula	AO.B_0
Weight (volts)	$\frac{\text{(Double)WEIGHT}}{10^9}$	$\frac{\text{(Double)1220703}}{10^9} = 0.001220703$
Offset (volts)	$\frac{\text{(Double)(I32)OFFSET}}{10^9}$	$\frac{\text{(Double)(I32)0}}{10^9} = 0$
Register value	$\text{(U16)}(Sign)\frac{\text{(Voltage-Offset)}}{\text{Weight}}$	$\text{AO.A\_0.VAL} = \text{(U16)}\frac{\text{(Voltage-0)}}{0.001220703}$

Note Output values do not take effect until AO.SYS.GO is set to TRUE.

Analog Input Ready Register (SYS.AI.RDY)

Register list: SYS.AI.RDY

Data type: Boolean

This register is FALSE when the FPGA code starts and changes to TRUE after the AI subsystem has been initialized. This register can be used to determine when the values available in the AI VAL registers are valid. If the VAL registers are read before this value is TRUE, you might encounter expected behavior.
Note: The SYS.RDY register indicates when all subsystems, including AI, are ready. Most applications should check the SYS.RDY register instead of this register.

Analog Output Set Register (AO.SYS.GO)

Register list: AO.SYS.GO

Data type: Boolean

This register causes values written to the AO.xx.VAL registers to take effect. Values written to the Analog Output VAL registers don't take effect until the AO.SYS.GO bit is strobed. You only need to write TRUE to this register as the register resets to FALSE after the write operation starts and the output is maintained until this register is again set to TRUE. Values for all registers are set for a single strobe of the GO register. Care should be taken to not change a VAL register if you do not want the output voltage to change when GO is strobed.

Analog Output Status Register (AO.SYS.STAT)

Register list: AO.SYS.STAT

Data type: Boolean

This register toggles every time the analog output write operation completes. If the value is read before the GO register is toggled, it can be used to determine when the write operation completes by waiting for the value to change from the initial value.

DIO

The DIO channels are divided into 8 channel banks. For example: MXP Connector B has 16 channels, DIO0 to DIO 15. They are divided as follows.

- B\DIO7:0 (ConnectorB/DIO7 to ConnectorB/DIO0)
- B\DIO15:8 (ConnectorB/DIO15 to ConnectorB/DIO8)

There are three registers to access each DIO channel bank.

Note When you program in C language, register names must not contain periods, colons, or spaces.

Data Direction Registers (DIO.xx.DIR)

Register list: DIO.B_7:0.DIR, DIO.B_15:8.DIR, DIO.DIO_7:0.DIR, DIO.DIO_9:8.DIR, DIO_SPI_CS3:0.DIR

Data type: U8

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, in DIO.B_7:0.DIR bit 0 controls B/DIO0, while bit 7 controls B/DIO7. If the bit is set to 1, the channel is an output. If the bit is set to 0, the channel is an input.

Pin Input Registers (DIO.xx.IN)

Register list: DIO.B_7:0.IN, DIO.B_15:8.IN, DIO.DIO_7:0.IN, DIO.DIO_9:8.IN, DIO_SPI_CS3:0.IN

Data type: U8

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, in DIO.B_7:0.IN bit 0 corresponds to B/DIO0, while bit 7 corresponds to B/DIO7. 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.B_7:0.OUT, DIO.B_15:8.OUT, DIO.DIO_7:0.OUT, DIO.DIO_9:8.OUT, DIO_SPI_CS3:0.OUT

Data type: U8

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, in DIO.B_7:0.OUT bit 0 corresponds to B/DIO0, while bit 7 corresponds to B/DIO7. 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.

For example, if Channel B/DIO0 is set as an input but left unconnected and the IN register is read, bit 0 reads a value of 1. If a 0 is written to bit 0 of the OUT register, there is no effect on the hardware and bit 0 of the IN register is still 1. However, if the channel is changed to an output, the value read on bit 0 of the IN register immediately changes to a 0 and a low voltage returns at the pin.

Note This follows the functionality of the Set Output Data and Set Output Enable FPGA IO Method nodes. Refer to the *LabVIEW Help* for more information about using FPGA I/O.

PWM

Note When you program in C language, register names must not contain periods, colons, or spaces.

PWM Configuration Registers (PWM.x.CNFG)

Register list: PWM.B_0.CNFG, PWM.B_1.CNFG, PWM.B_2.CNFG, PWM.PWM_0.CNFG, PWM.PWM_1.CNFG, PWM.PWM_2.CNFG, PWM.PWM_3.CNFG, PWM.PWM_4.CNFG, PWM.PWM_5.CNFG, PWM.PWM_6.CNFG, PWM.PWM_7.CNFG, PWM.PWM_8.CNFG, PWM.PWM_9.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.B_0.CS, PWM.B_1.CS, PWM.B_2.CS, PWM.PWM_0. CS, PWM.PWM_1. CS, PWM.PWM_2. CS, PWM.PWM_3. CS, PWM.PWM_4. CS, PWM.PWM_5. CS, PWM.PWM_6. CS, PWM.PWM_7. CS, PWM.PWM_8. CS, PWM.PWM_9. 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 (f_{clk})
0	1	0	2x (f_{clk} / 2)
0	1	1	4x (f_{clk} / 4)
1	0	0	8x (f_{clk} / 8)
1	0	1	16x (f_{clk} / 16)
1	1	0	32x (f_{clk} / 32)
1	1	1	64x (f_{clk} / 64)

The base clock frequency (f_{clk}) 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.B_0.MAX, PWM.B_1.MAX, PWM.B_2.MAX, PWM.PWM_0. MAX, PWM.PWM_1. MAX, PWM.PWM_2. MAX, PWM.PWM_3. MAX, PWM.PWM_4. MAX, PWM.PWM_5. MAX, PWM.PWM_6. MAX, PWM.PWM_7. MAX, PWM.PWM_8. MAX, PWM.PWM_9. 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.B_0.CMP, PWM.B_1.CMP, PWM.B_2.CMP, PWM.PWM_0. MAX, PWM.PWM_1. MAX, PWM.PWM_2. MAX, PWM.PWM_3. MAX, PWM.PWM_4. MAX, PWM.PWM_5. MAX, PWM.PWM_6. MAX, PWM.PWM_7. MAX, PWM.PWM_8. MAX, PWM.PWM_9. MAX

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.B_0.CNTR, PWM.B_1.CNTR, PWM.B_2.CNTR, PWM.PWM_0. CNTR, PWM.PWM_1. CNTR, PWM.PWM_2. CNTR, PWM.PWM_3. CNTR, PWM.PWM_4. CNTR, PWM.PWM_5. CNTR, PWM.PWM_6. CNTR, PWM.PWM_7. CNTR, PWM.PWM_8. CNTR, PWM.PWM_9. 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 roboRIO hardware runs on a 40 MHz clock, which means the time between clock cycles is 25 ns. The roboRIO can generate slower PWM frequencies by counting and changing the output on intervals of rising clock edges. The roboRIO 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.

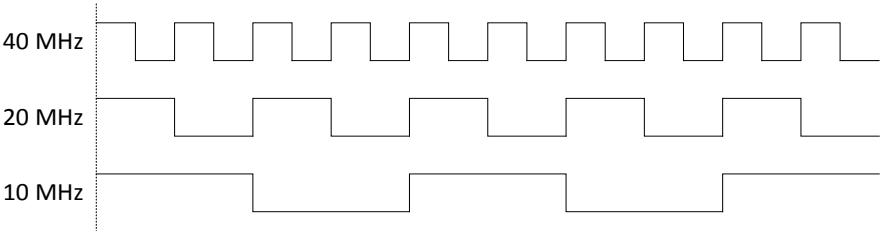


Figure 1. Generating Slower PWM Frequencies

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

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

$$\frac{1}{25\text{ ns} \times 65536} \cong 610.35\text{ Hz}$$

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 roboRIO 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:

$$\frac{1}{50\text{ ns} \times 65536} \cong 305.17\text{ Hz}$$

The following formula describes possible frequencies:

$$f_{PWM} = \frac{f_{clk}}{N\ (X+1)}$$

where f_{clk} is the base clock frequency, f_{PWM} 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

Note When you program in C language, register names must not contain periods, colons, or spaces.

SPI Configuration Registers (SPI.x.CNFG)

Register list: SPI.B.CNFG, SPI.SPI.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 (f_{clk})
0	1	2x (f_{clk} / 2)
1	0	4x (f_{clk} / 4)
1	1	8x (f_{clk} / 8)

  The base clock frequency (f_{clk}) 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.

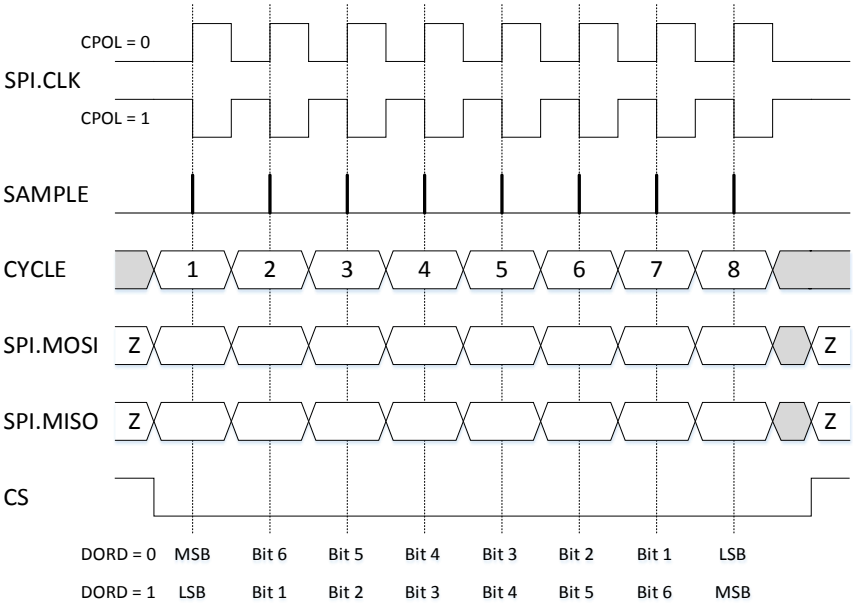


Figure 2. SPI Transfer Format with CPHA = 0

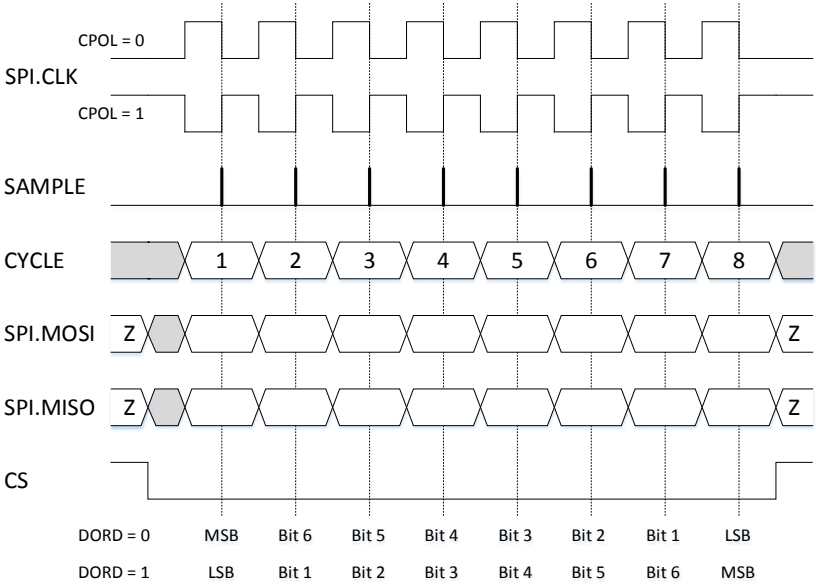


Figure 3. SPI Transfer Format with CPHA = 1

SPI Counter Registers (SPI.x.CNT)

Register list: SPI.B.CNT, SPI.SPI.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.B.GO, SPI.SPI.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.B.STAT, SPI.SPI.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.B.DATO, SPI.SPI.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.B.DATO, SPI.SPI.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 roboRIO hardware runs on a 40 MHz clock, which means the time between clock cycles is 25 ns. The roboRIO can generate slower SPI frequencies by counting and changing the output on intervals of rising clock edges. The roboRIO 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.

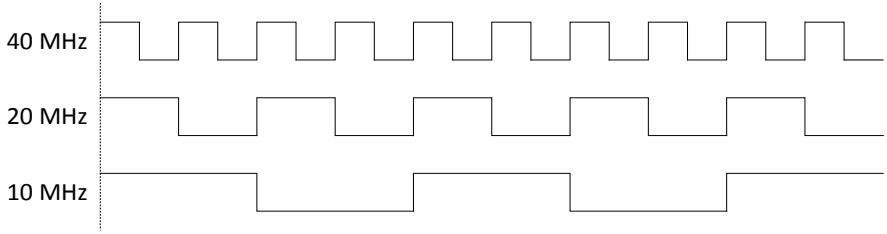


Figure 4. Generating Slower SPI Frequencies

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

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

$$\frac{1}{25\text{ ns} \times 65536} \cong 610.35\text{ Hz}$$

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 roboRIO 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:

$$\frac{1}{50\text{ ns} \times 65536} \cong 305.17\text{ Hz}$$

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

$$f_{SPI} = \frac{f_{clk}}{2 \times N \times (X+1)}$$

where f_{clk} is the base clock frequency, f_{SPI} 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.

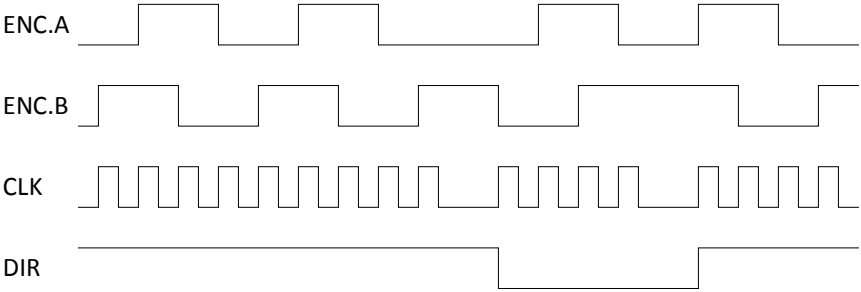


Figure 5. A Waveform with Phase A, Phase B, Step (CLK), and Direction (DIR) Signals

Note When you program in C language, register names must not contain periods, colons, or spaces.

Encoder Configuration Registers (ENC.x.CNFG)

Register list: ENC.B.CNFG, ENC.DIO_0.CNFG, ENC.DIO_1.CNFG, ENC.DIO_2.CNFG, ENC.DIO_3.CNFG, ENC.DIO_4.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.B.STAT, ENC.DIO_0.STAT, ENC.DIO_1.STAT, ENC.DIO_2.STAT, ENC.DIO_3.STAT, ENC.DIO_4.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.B.CNTR, ENC.DIO_0.CNTR, ENC.DIO_1.CNTR, ENC.DIO_2.CNTR, ENC.DIO_3.CNTR, ENC.DIO_4.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

Note When you program in C language, register names must not contain periods, colons, or spaces.

I2C Configuration Registers (I2C.x.CNFG)

Register list: I2C.B.CNFG, I2C.I2C.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.B.ADDR, I2C.I2CADDR

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.B.CNTR, I2C.I2C.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:

$$f_{SCL} = \frac{f_{clk}}{(2 \times CNTR) - 26}$$

where f_{SCL} is the desired I2C transmission frequency and f_{clk} is the base clock frequency of the hardware (40 MHz).

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

$f_{SCL} = 100$ kHz

$f_{clk} = 40$ MHz

Since $f_{SCL} = \frac{f_{clk}}{(2 \times CNTR) - 26}$

$$CNTR = \frac{f_{clk}/f_{SCL} + 26}{2}$$

$$CNTR = \frac{40\ MHz/100\ kHz + 26}{2}$$

CNTR = 213

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.B.DATO, I2C.I2C.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.B.DATI, I2C.I2C.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.B.STAT, I2C.I2C.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 8. 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.B.CNTL, I2C.I2C.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 9. 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.B.GO, I2C.I2C.GO

Data type: Boolean

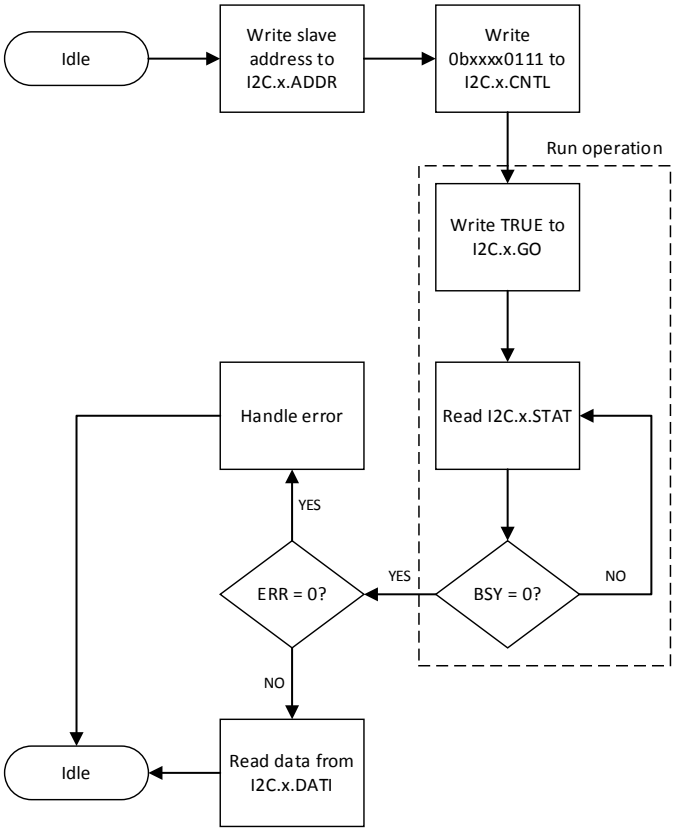
This register causes the operation specified in the I2C.x.CNTL register to begin. When an operation is written to the CNTL register, it does not start until the GO bit is strobed. The

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.

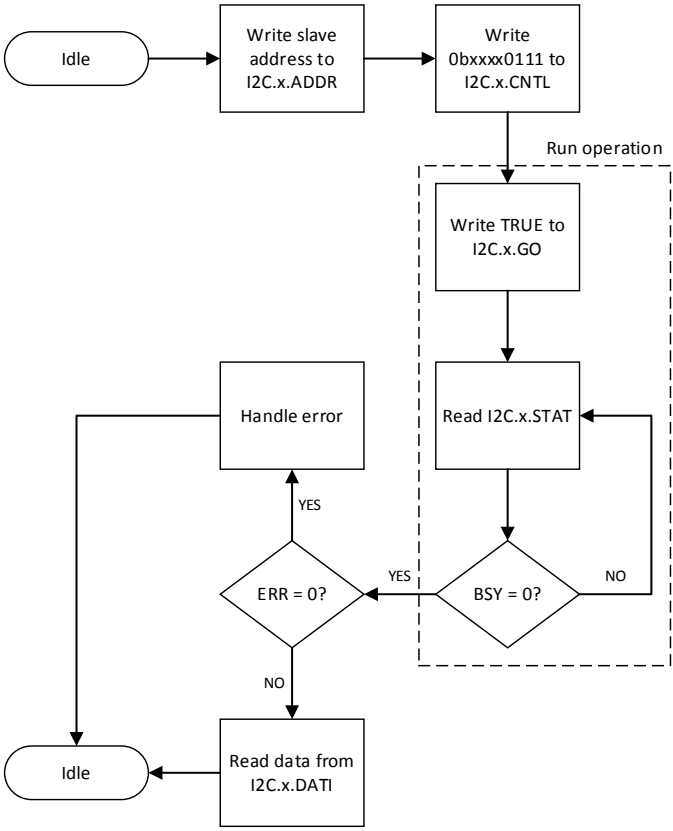
I2C Sequence Flowcharts

The following figures show the sequence of events required to use the I2C peripheral.

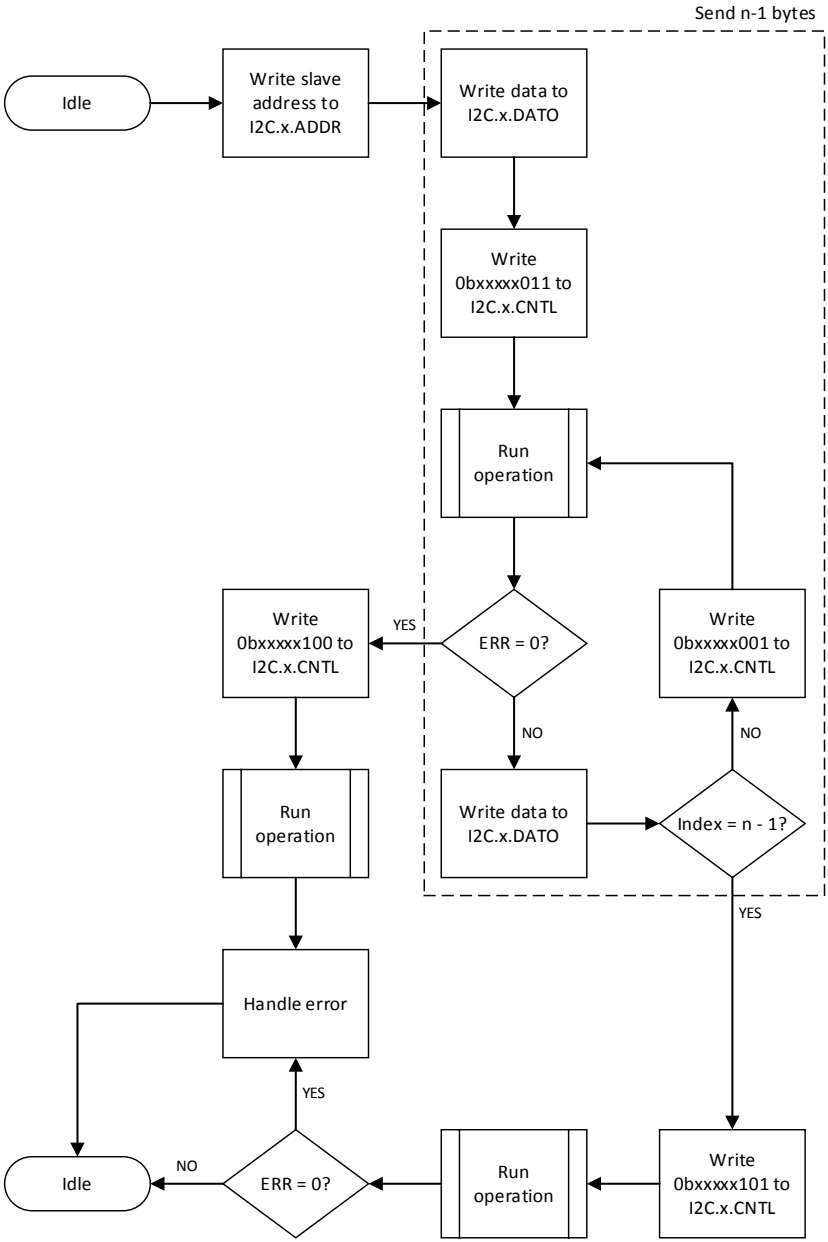
Sending a Single Byte



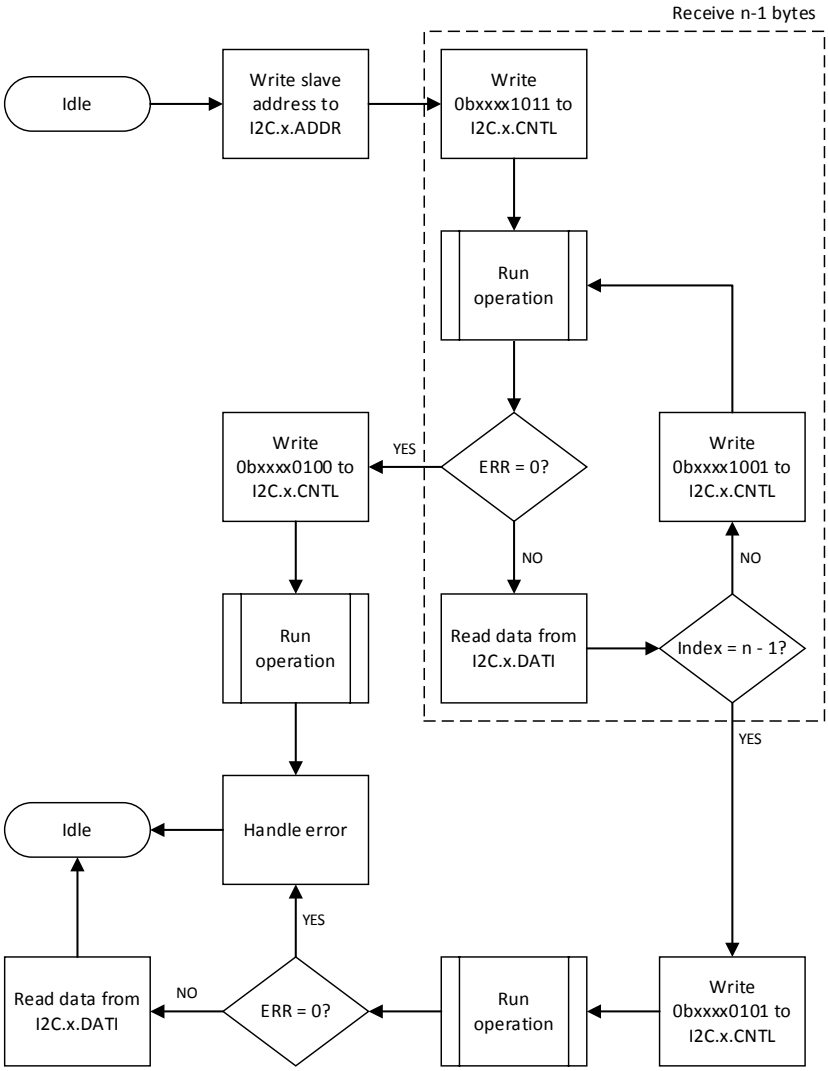
Receiving a Single Byte



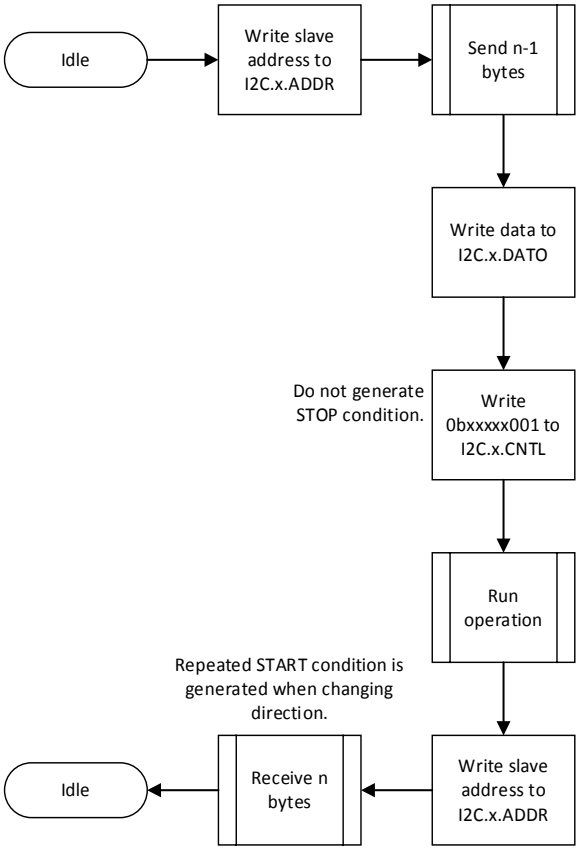
Sending Multiple Bytes



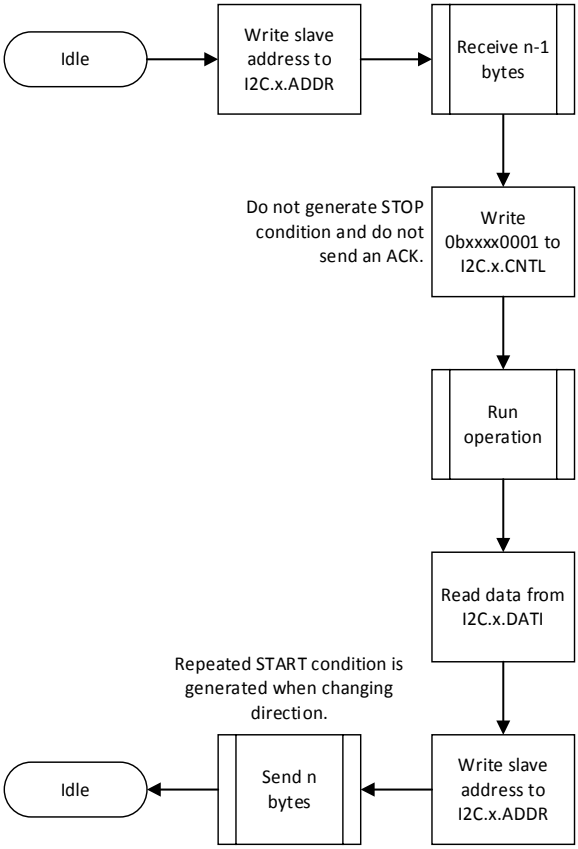
Receiving Multiple Bytes



Sending Multiple Bytes then Receiving Multiple Bytes



Receiving Multiple Bytes then Sending Multiple Bytes



IRQ

Note When you program in C language, register names must not contain periods, colons, or spaces.

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_B_0.THRESHOLD, IRQ.AI_B_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_B_0.HYSTERESIS, IRQ.AI_B_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_B_3:0.CNFG)

Register list: IRQ.AI_B_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_B_1.Type	IRQ.AI_B_1.ENA	IRQ.AI_B_0.Type	IRQ.AI_B_0.ENA
Initial Value	0	0	0	0	0	0	0	0

- **Bits [7:4]** - Reserved for future use.
- **Bit [3]** - IRQ.AI_B_1. Type.

  Specifies the interrupt type of the channel. If the bit is set to 1, the AI1 channel on connector B checks AI interrupts on a rising edge of the analog input signal. If the bit is set to 0, the AI1 channel on connector B checks AI interrupts on a falling edge of the analog input signal.
- **Bit [2]** - IRQ.AI_B_1. ENA.
  Enables the settings of the analog input interrupt channel. If the bit is set to 1, the AI1 channel on connector B starts checking AI interrupts based on the settings. If the bit is set to 0, the AI1 channel on connector B stops checking AI interrupts. The default value of the bit is 0 when the roboRIO device is powered on.
- **Bit [1]** - IRQ.AI_B_0. Type.

  Specifies the interrupt type of the channel. If the bit is set to 1, the AI0 channel on connector B checks AI interrupts on a rising edge of the analog input signal. If the bit is set to 0, the AI0 channel on connector B checks AI interrupts on a falling edge of the analog input signal.
- **Bit [0]** - IRQ.AI_B_0. ENA

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

Analog IRQ Number Register (IRQ.AI_xx.NO)

Register list: IRQ.AI_B_0.NO, IRQ.AI_B_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_B_7:0.ENA)

Register list: IRQ.DIO_B_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_B_3.ENA	IRQ.DIO_B_2.ENA	IRQ.DIO_B_1.ENA	IRQ.DIO_B_0.ENA
Initial Value	0	0	0	0	0	0	0	0

- **Bits [7:4]** - Reserved for future use.
- **Bits [3:0]** - IRQ.DIO_B_3:0.ENA.

Each bit in **Bits [3:0]** controls the settings of one channel. For example, in IRQ.DIO_B_3:0.ENA, bit 0 controls B/DIO0, while bit 3 controls B/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 roboRIO device is powered on.

Digital Rising Register (IRQ.DIO_B_7:0.RISE)

Register list: IRQ.DIO_B_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_B_3.RISE	IRQ.DIO_B_2.RISE	IRQ.DIO_B_1.RISE	IRQ.DIO_B_0.RISE
Initial Value	0	0	0	0	0	0	0	0

- **Bits [7:4]** - Reserved for future use.
- **Bits [3:0]** - IRQ.DIO_B_3:0.RISE.

Each bit in **Bits [3:0]** controls one channel. For example, in IRQ.DIO_B_3:0.RISE, bit 0 controls B/DIO0, while bit 3 controls B/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_B_7:0.FALL)

Register list: IRQ.DIO_B_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_B_3.FALL	IRQ.DIO_B_2.FALL	IRQ.DIO_B_1.FALL	IRQ.DIO_B_0.FALL
Initial Value	0	0	0	0	0	0	0	0

- **Bits [7:4]** - Reserved for future use.
- **Bits [3:0]** - IRQ.DIO_B_3:0.FALL.

Each bit in **Bits [3:0]** controls one channel. For example, in IRQ.DIO_B_3:0.FALL, bit 0 controls B/DIO0, while bit 3 controls B/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_B_0.NO, IRQ.DIO_B_1.NO, IRQ.DIO_B_2.NO, IRQ.DIO_B_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_B_XX.CNT)

Register list: IRQ.DIO_B_0.CNT, IRQ.DIO_B_1.CNT, IRQ.DIO_B_2.CNT, IRQ.DIO_B_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 roboRIO 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.

RSL

Note When you program in C language, register names must not contain periods, colons, or spaces.

Pin Output Registers (DO.RSL.OUT)

Register list: DO.RSL.OUT

Data type: Boolean

This register enables the RSL channel. When you write a TRUE value to this register, the RSL output returns the roboRIO power supply voltage (7-16 V), and the RSL LED turns on to indicate that an output voltage exists on the RSL port. When you write a FALSE value to this register, the RSL output returns the RSL GND and the RSL LED is off.

Note The reference of the RSL output is the RSL GND. The RSL GND is not directly connected to the GND of other ports on the roboRIO.

Note When you program in C language, register names must not contain periods, colons, or spaces.

Pin Output Registers (DO.RELAY.OUT)

Register list: DO.REALY.OUT

Data type: U8

Each relay channel contains a forward (PWD) pin and a reverse (REV) pin. This register enables the FWD and REV pins on Connector RELAY.

Bit	7	6	5	4	3	2	1	0
Name	RelayRev3	RelayFwd3	RelayRev2	RelayFwd2	RelayRev1	RelayFwd1	RelayRev0	RelayFwd0
Initial Value	0	0	0	0	0	0	0	0

- **Bits [7:0]** – RelayFwd [0:3], RelayRev [0:3]

Each bit in **Bits [7:0]** controls one pin. For example, bit 0 controls RELAY/FWD0, while bit 3 controls RELAY/REV1. If the bit is set to 1, the pin returns 5 V. If the bit is set to 0, the pin returns 0 V.