

Users Reference Manual

XIO-RLY8

External Eight Channel Relay Output Board
with Qwiic / Stemma QT I²C interface

Model No.	100-7749	
Doc. No.	M7749	Rev: 2.00 6/9/26



649 School Street / Pembroke, MA 02359 USA / Tel: (781) 293-3059

www.scidyne.com

DISCLAIMER:

This document contains proprietary information regarding SCIDYNE and its products. The information is subject to change without notice. SCIDYNE makes no warranty of any kind with regard to this material, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. SCIDYNE shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material. No part of this document may be duplicated in any form without prior written consent of SCIDYNE.

WARRANTY:

SCIDYNE warrants this product against defects in materials and workmanship and, that it shall conform to specifications current at the time of shipment, for a period of one year from date of shipment. Duration and conditions of warranty may be superseded when the product is integrated into other SCIDYNE products. During the warranty period, SCIDYNE will, at its option and without charge to Buyer, either repair or replace products which prove defective. Repair or replacement of a defective product or part thereof does not extend the original warranty period.

WARRANTY SERVICE:

For warranty service or repair, this product must be returned to a service facility designated by SCIDYNE. The Buyer must obtain prior approval and a Return Material Authorization (RMA) number before returning any products. The RMA number must be clearly visible on the shipping container. The Buyer shall prepay shipping and insurance charges to the service facility and SCIDYNE shall pay shipping and insurance charges to Buyer's facility for products repaired or replaced. SCIDYNE may, at its discretion, bill the Buyer for return shipping and insurance charges for products received for repair but determined to be non-defective. Additionally, the Buyer shall pay all shipping charges, duties and taxes for

products returned to SCIDYNE from another country.

LIMITATION OF WARRANTY:

The forgoing warranty shall not apply to defects resulting from improper or negligent maintenance by the Buyer, Buyer-supplied products or interfacing, unauthorized modifications or misuse, operation outside the published specifications of the product or improper installation site preparation or maintenance, or the result of an accident. The design and implementation of any circuit using this product is the sole responsibility of the Buyer. SCIDYNE does not warrant the Buyer's circuitry or malfunctions of SCIDYNE products that result from the Buyer's circuitry. In addition, SCIDYNE does not warrant any damage that occurs as a result of the Buyer's circuit or any defects that result from Buyer-supplied products. This Warranty does not cover normal preventative maintenance items such as fuse replacement, lamp replacement, resetting of circuit breakers, cleaning of the Product or problems caused by lack of preventative maintenance, improper cleaning, improper programming, or improper operating procedures. No other warranty is expressed or implied. SCIDYNE specifically disclaims the implied warranties of merchantability and fitness for a particular purpose. Some states do not permit limitation or exclusion of implied warranties; therefore, the aforesaid limitation(s) or exclusion(s) may not apply to the Buyer. This warranty gives you specific legal rights and you may have other rights which vary from state to state.

CERTIFICATION:

Testing and other quality control techniques are utilized to the extent SCIDYNE deems necessary to support this warranty. Specific testing of all parameters is not necessarily performed, except those mandated by government requirements.

**30-DAY PRODUCT
EVALUATION POLICY:**

SCIDYNE offers a no-risk trial for initial, low quantity, evaluation purchases. Items purchased for evaluation can be returned within 30 days of purchase for a full refund less shipping charges. The Buyer must obtain a Return Material Authorization (RMA) number before returning any products. The entire package, including hardware, software, documentation, discount coupons and any other accessories supplied must be returned intact and in new and working condition. This policy will not be honored for packages that are not returned complete and intact. The Buyer shall prepay shipping and insurance charges to SCIDYNE. To expedite the return process, the RMA number must be clearly visible on the shipping container. SCIDYNE will cancel the invoice, refund by check or issue credit to your credit card within 10 days after receipt of returned merchandise.

LIFE SUPPORT POLICY:

Certain applications may involve the risks of death, personal injury or severe property or environmental damage ("Critical Applications").

SCIDYNE products are not designed, intended, authorized or warranted to be suitable for use in life-support applications, devices or systems or other critical applications without the express written approval of the president of SCIDYNE.

Table of Contents

Conventions and Terminology used in this publication	2
Safety and Usage Conventions	2
Terminology.....	2
Introduction.....	2
Key Features	2
Component Identification	3
Initial Setup.....	4
Operating Power Source	4
External Power (Default)	4
Host Supplied Power.....	4
I2C Address	4
Changing the I2C address	5
Set Factory Default Address	6
Hardware Details	7
I2C Interface	7
Pull-Up resistors (optional).....	7
Relay Outputs.....	7
Software Details.....	8
Example Software	8
Arduino	8
Raspberry Pi.....	8
Output Register	8
I2C protocol	8
regRW - Register Read / Write.....	9
regBitSet - Register Bit Set.....	9
regBitReset - Register Bit Reset	9
I2C Bus Speed and Latency	9
Appendix - A Specifications.....	10
Appendix - B Arduino Software Library	11
Library Inclusion and Instantiation.....	11
setup()	11
setBit(), resetBit()	12
readBit()	12
byteWrite().....	13
byteRead().....	13
getProductInfo().....	14
User Notes.....	15

Conventions and Terminology used in this publication

Safety and Usage Conventions

Note:



Provides important information and useful tips that will assist in the understanding and operation of this product.

Caution:



Calls attention to a procedure, practice, or condition that could possibly cause equipment damage or bodily injury.

Danger:



Calls attention to a procedure, practice, or condition that is likely to cause extensive equipment damage, severe bodily injury, or death if not observed.

Terminology

Logic Conditions

Unless otherwise noted, logic signals are designated as TRUE being logic '1' or 'High' (i.e.; Positive Voltage, +5Vdc or +3.3Vdc) and FALSE is being logic '0' or 'Low' (i.e.; 0Vdc). Signal names with an asterisk (*) postscript or overlined are inverted or active low.

Numbering Systems

Computerized equipment often requires its numeric data to be represented in different forms depending on the audience and information being conveyed. Decimal numbers are typically used for end-user data entry and display while internally these values are converted and manipulated in native binary. Hexadecimal numbers are often used by programmers as an intermediate level between binary and decimal notations.

Base	Name	Format (MS ←---→ LS)
2	Binary	0b10111001 or 1011 1001 ₂
10	Decimal	185
16	Hexadecimal	0xB9 or B9 ₁₆ or HB9

Multi-Byte Word Formats

In this document, multi-byte values use big-endian format (e.g., 0x1234), where the most significant byte (0x12) is stored first and at the lowest memory address, and the least significant byte (0x34) at higher addresses. Actual storage may vary by system architecture (big- or little-endian).

Host

A device acting as Master in a Master / Slave bus configuration.

Introduction

The XIO-RLY8 is an eight-channel Relay Output Board employing the industry standard Qwiic / Stemma QT pluggable interconnection system. It can be used with any device that supports I2C communications such as Arduino, Raspberry Pi, ESP32, small Microcontrollers, and FPGAs. In addition, the I2C hardware has been engineered to operate at either 3.3V or 5V making the XIO-RLY8 highly compatible.

Key Features

- Eight 10A Form-C relay outputs
- Relays provide inherent galvanic isolation
- Two 4-wire Qwiic / Stemma QT bus connectors
- I2C communications
- Single address, user configurable as 0x08 - 0x77
- Built-in Set and Reset bit operations
- LEDs show board operation at a glance
- Compatible with 3.3V and 5V logic
- Screw-Terminals accept 14-24 AWG field wiring
- External or Bus powered

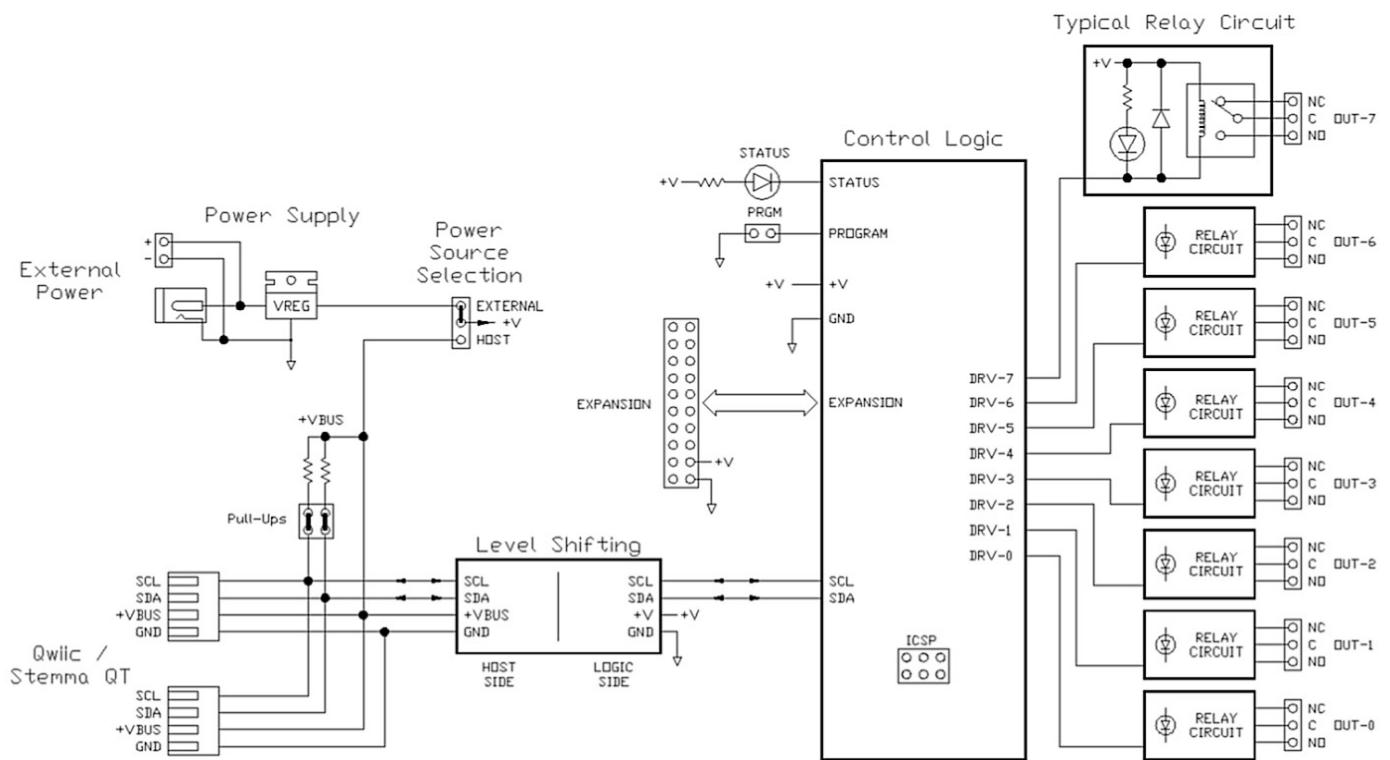


Figure 1 – XIO-RLY8 Simplified Block Diagram

Component Identification

To properly apply the XIO-RLY8 it is necessary to become familiar with its various components. The following figure and accompanying text briefly describe their locations and functions. Subsequent sections of this manual explain their purpose in greater detail.

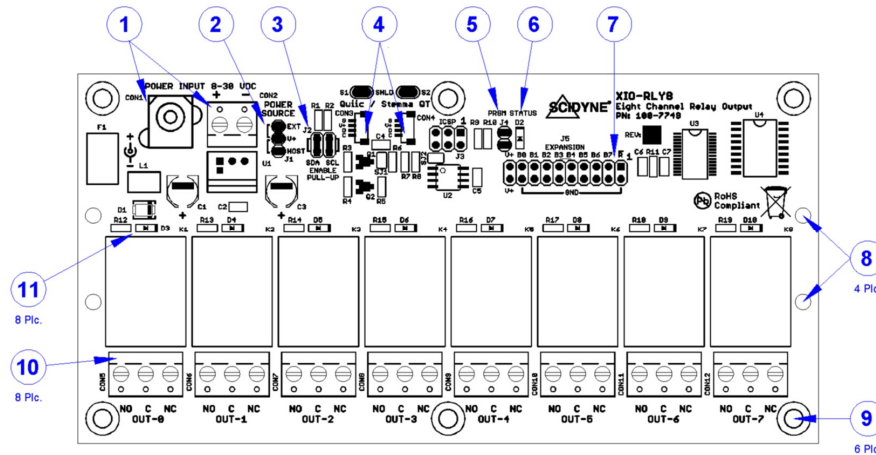


Figure 2 – XIO-RLY8 Component Identification

1 **Power Input (CON1, CON2)**

External power, 8 – 30Vdc

CON1: Barrel Jack accepts standard 2.1mm x 5.5mm plug, positive center.

CON2: Screw terminals, observe polarity marked on board.

2 **Power Source (J1)**

This jumper determines the source of power for the XIO-RLY8.

EXT: (Default Position) A suitable power supply must be used and connected to either CON1 or CON2, See item #1 above.

HOST: Power is supplied by the Host via the Qwiic / Stemma connectors. The Host must supply adequate voltage and current for the XIO-RLY8 to operate reliably.

3 **Pull-Up Enable (J2)**

These jumpers enable optional Pull-Up resistors on the I2C SDA and SCL signals. Only one set of Pull-Up resistors should be used on any I2C bus segment and generally at the furthest wired distance from the Host. The Pull-Ups are referenced to Host +V.

4 **Qwiic⁽¹⁾ / Stemma QT⁽¹⁾ (CON3, CON4)**

Connects the XIO-RLY8 to the Host and other Qwiic / Stemma QT peripherals.

5 **PRGM (J4)**

Temporarily shorting these pins during power-up will place the XIO-RLY8 into a special programming mode or loads factory defaults.

6 **STATUS LED (D2)**

This LED glows steadily during normal operation or blinks while in programming and load factory defaults modes.

7 **Optional Expansion connector (J5)**

This unpopulated 2x10 header is for custom and future uses. Do not make connections to these locations.

8 **DIN Rail clip mounting holes**

These 4 holes accept optional DIN Rail Clip #121-0014 allowing the XIO-RLY8 to be mounted on DIN Rail.

9 **Standoff mounting holes**

Use these 6 holes when mounting the XIO-RLY8 with Stand-Offs. Use 4-40 or M3 Pan-Head machine screws.

10 **Relay Output Screw-Terminals (CON5 – CON12)**

The relay contacts for each channel are accessible using these screw terminals. Observe the Normally Open (NO), Common (C), and Normally Closed (NC) designation when wiring. Accepts wires to 14 AWG.

11 **Output Status LEDs (D3 – D10)**

Operation of each channel is shown on these LEDs. An illuminated LED indicates a channel's corresponding relay is energized, i.e.; Normally Open is connected to Common.

1) Qwiic and Stemma QT are peripheral hardware buses based on I2C communications and popularized by SparkFun, Adafruit, and other third parties.

Initial Setup

The XIO-RL8 must be properly configured to work in your application. This includes configuring the Operating Power Source and possibly changing its I2C address.

Operating Power Source

Power can be externally supplied power or provided by the Host through the Qwiic / Stemma QT bus. The selection is made by placing a jumper at the corresponding position of J1. The default and preferred method is to use an external power source.



For reliable operation the power source must be clean, noise free, and quick settling at power on. The XIO-RLY8 requires approximately 4 watts of power when all relays are activated.

External Power (Default)

External power is applied to the XIO-RLY8 using the Screw-Terminals CON2, or Barrel Jack CON1. The barrel jack is a standard 2.1mm x 5.5mm style plug with a positive center. Any external supply must be in the range of 8 - 30VDC and provide approximately 4 watts of power.

Host Supplied Power

Place a jumper at the HOST position of J1 when powering from the Host via the Qwiic / Stemma QT bus through connectors CON3 or CON4. Host supplied power should only be used when the Qwiic / Stemma QT bus is operating at 5Vdc.

I2C Address

The XIO-RLY8 uses the industry standard Qwiic / Stemma QT bus for communications with a Host. The underlying I2C protocol requires each device on the bus to have a unique address. The factory default address for the XIO-RLY8 is 0x30. This address can be changed to any free I2C address in the range of 0x00 to 0x7F. However, the new address must not be used by any other device or interfere with the functionality of the I2C protocol.



The I2C specification reserves several addresses for specific functions such as initiating communications with all devices on the bus or signaling special bus formats. To retain these functions the following addresses should be avoided.

<u>Reserved Address</u>	<u>Usage</u>
0x00:	General Call Address, used for broadcasting messages to all devices
0x01:	Reserved for CBUS compatibility, Start Byte
0x02:	Reserved for different bus formats
0x03:	Reserved for future purposes
0x04 – 0x07:	High-Speed Master Code
0x78 – 0x7B:	10-bit Slave Addressing Prefix
0x7C – 0x7F:	Reserved for future purposes, Device ID

Changing the I2C address

A programming utility is included with the XIO-RLY8 example software that facilitates changing factory default address. The procedure below applies to either the Arduino or Raspberry Pi versions of the software. It is written in general terms and should be adapted to meet each user's particular situation, i.e.; which Host being used, performed on a bench or in the field, Host or Externally powered, ... etc.

Materials Needed:

- ☐ Arduino or Raspberry Pi
- ☐ Programming software code. Provided with XIO-RLY8 software examples
Arduino: XIO-RLY8_programmer.ino Raspberry Pi: XIO-RLY8_programmer.py
- ☐ Shorting Jumper, or small metallic flat-head screwdriver, or paper clip
- ☐ Qwiic / Stemma QT breakout cable
- ☐ (optional) Power Supply, 8-30Vdc. For powering the XIO-RLY8 externally.

Procedure:

1. Turn the Power Off to the XIO-RLY8 and Host. It is recommended to disconnect all external loads being driven by the XIO-RLY8.
2. Disconnect the XIO-RLY8 from all other I2C devices on the bus.
3. Connect only the Host and XIO-RLY8 using a Qwiic / Stemma QT cable.

Some Host boards, such as the Arduino UNO R4 WiFi, have a Qwiic connector built-in making the connection straight-forward.

Other Host will require a Qwiic breakout cable with separately accessible wires to make the SDA, SCL and Power interconnections.

Qwiic / Stemma QT Connector pinout CON3, CON4		
Signal		Wire Color (Typ.)
GND	①	Black
V+ ⁽¹⁾	②	Red
SDA	③	Blue
SCL	④	Yellow

⁽¹⁾ IMPORTANT: Connect V+ to the same power supply as the Host operating Voltage.



The four signal connections between the Host and XIO-RLY8 must match exactly. Failure to do so may result in hardware failure. Verify all wiring before applying power.

4. Make note of the positions for the Power Source selection Jumper (J1) and Enable Pull-Up jumpers (J2). This will allow easy reconfiguration after completing the procedure.
5. Install a jumper on the Power Source jumper between +V and HOST. Install jumpers at the SDA and SCL positions of the Enable Pull-up jumpers (J2).

6. Gently short and hold the PRGM (J4) pins together with a conductive metallic object such as a shorting jumper, a small flat-headed screwdriver tip, or paper clip. Be careful not to come in contact with any surrounding circuitry.
7. Power up the Host and XIO-RLY8. Release the PRGM short within 5 Seconds. The programming mode is indicated by the STATUS LED slowly blinking.
8. Depending on the Host being used, load the appropriate programming software.

Arduino

Software: XIO-RLY8_programmer.ino

Note: The software uses Wire by default. If necessary, modify the line of code where indicated to use Wire1 instead. The need to make this change depends on the particular Arduino and I2C bus being used.

Raspberry Pi

Software: XIO-RLY8_programmer.py

Note: The python software uses the native I2C by default. Bit-Banged I2C can be used instead by modifying a line of code where indicated. Make sure the SCL and SDA signals are connected to the RPi's GPIO pins referenced in the software.

9. Run the software. Follow On-Screen instructions and progress using either the Arduino IDE Serial Monitor or a Raspberry Pi terminal window.
10. The program will report on the current XIO-RLY8 I2C address and prompt for a new address. Enter a Decimal value between 8 and 120.
11. The program will confirm if the reprogramming operation was successful.
12. Power down. Return the Power Selection jumper (J1) and Enable Pull-Up jumpers (J2) to their original positions. Attach all Qwiic cables as they were. Reconnect any loads that were previously disconnected.
13. The XIO-RLY8 will now operate at the I2C address entered in step #9
14. Be sure to make corresponding address changes in the Host application software.

Set Factory Default Address

Setting Factory Defaults overwrites any previously modified parameters to a known configuration.

<u>Parameter</u>	<u>Factory Default Value</u>
I2C Address	0x30

Procedure

1. Turn off power to the XIO-RLY8.
2. Gently short and hold the PRGM (J4) pins together with a conductive metallic object such as a shorting jumper, a small flat-headed screwdriver tip, or paper clip. Be careful not to come in contact with any other surrounding circuitry.
3. Power up the XIO-RLY8 while keeping the short applied for more than 10 Seconds.
4. Release the short. The STATUS LED will repeatedly flash in a Blink-Blink-Pause pattern indicating the Factory Defaults have been loaded.
5. Cycle the XIO-RLY8 power to exit the Factory Default mode.

Hardware Details

I2C Interface

For fast and easy setup two industry standard Qwiic / Stemma QT connectors are provided. They are wired in parallel eliminating the need for any upstream vs. downstream cabling considerations. The pinout is shown on the right.

Built-in level-shifters automatically convert the Host I2C bus voltage to levels appropriate for the XIO-RLY8. This ensures compatibility with Hosts operating at either 3.3V or 5V. **Never intermix the two voltage levels on the same bus.**

Qwiic / Stemma QT Connector pinout CON3, CON4		
Signal		Wire Color (Typ.)
GND	①	Black
V+ ⁽¹⁾	②	Red
SDA	③	Blue
SCL	④	Yellow

⁽¹⁾ IMPORTANT: Connect V+ to the same supply as the Host operating Voltage.

Pull-Up resistors (optional)

I2C often requires pull-up resistors for reliable operation. These are typically installed at the device wired the furthest on the bus from the Host. The XIO-RLY8 can optionally enable 2.0K Ω pull-up resistors on the SDA and SCL signals, referenced to the Hosts V+ bus power supply. To enable the resistors install two shorting-jumpers on the J2 header.

Relay Outputs

The XIO-RLY8 provides eight sealed SPDT (Form C) relay outputs rated for currents up to 10 amperes. In addition, the relays inherently provide galvanic isolation and Break-Before-Make operation. This ensures a relay's Normally-Closed and Normally-Open contacts will not be mechanically connected at the same time.

A LED associated with each relay is illuminated when that channel is activated.

To ease wiring constraints, and permit flexible mixing of AC and DC signals, the Normally-Closed (NC), Normally-Open (NO) and Common (COM) contacts from each relay are accessible separately via Screw-Terminals, CON5 – CON12. Each screw terminal accepts wire sized to 14 AWG.



- 1. Relay outputs are not fuse protected. Attempting to pass current in excess of the rated limit can cause permanent damage to the relays and/or circuit board.*
 - 2. Inductive loads often generate voltages with enough potential to produce arcing between a relay's contacts. This effect can cause electrical noise problems in some situations and inevitably reduces the lifetime of the contacts. When driving inductive loads, it is suggested that an appropriate "Snubber" network be used across the contacts to dissipate any inductive spikes that may occur.*
-

Software Details

Example Software

The XIO-RLY8 can be controlled by any Host that supports I2C. Two popular devices are Arduino and Raspberry Pi, and their variants. Software examples for both platforms are available and can be downloaded from the XIO-RLY8 product webpage at scidyne.com/hpb7749.html.

Arduino

The Arduino examples include a library containing useful routines for controlling individual relay channels and retrieving product information. Appendix-B of this manual describes the Arduino library usage in detail.

Raspberry Pi

The Raspberry Pi examples are written in Python and C/C++ languages with a mix of native I2C and Bit-Banged versions.

Output Register

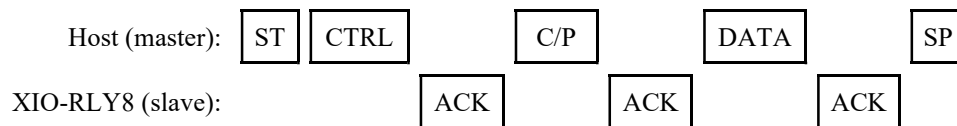
All software that controls the XIO-RLY8 relies on an 8-bit output register, regardless of the host system. Each bit in this register maps directly to a relay channel: bit 0 controls relay channel 0, bit 1 controls channel 1, and so on.

Register bits that are set to logic “1” will activate its associated relay. Conversely, register bits that are logic “0” will deactivate its associated relay. The output register is cleared at Power-On resulting in all relays being deactivated, i.e.; Normally-Closed and Common relay contacts will be connected.

I2C protocol

The I2C protocol implemented on the XIO-RLY8 offers several ways to control the output register including bit and byte operations. A Host initiates and directs the overall I2C communications.

A typical I2C write message



Where:

ST	I2C Start condition, initiated by Host hardware when SDA goes low while SCL is High
CTRL	Control Byte, bit[7:1] = slave address, bit[0] = data direction, 1 = Read / 0 = Write
C/P	Command / Parameter Byte, contents vary by usage
DATA	Data Byte, contents vary by usage
ACK or $\overline{\text{NAK}}$	Acknowledgement, $\overline{\text{Negative Acknowledgement}}$
SP	I2C Stop condition, initiated by Host hardware when SDA goes high while SCL is High

regRW - Register Read / Write

The simplest method to modify the output register by performing a byte write using the **register Read / Write** command (regRW). All eight bits are written simultaneously as outlined in this sequence.

1. Open I2C communications to XIO-RLY8 at its I2C address, i.e.; 0x30 default address
2. Write the Command/Parameter byte with a value of 0x20, i.e.; regRW command
3. Write the desired Output Register data byte value where each bit controls a relay; Bits that are “1” activate its corresponding relay, bits that are “0” deactivate its corresponding relay.
4. Close I2C communications

regBitSet - Register Bit Set

regBitReset - Register Bit Reset

The regRW command is useful when quickly establishing all relay states at once. However, using it to control individual relays becomes cumbersome. A Host would need to maintain a copy of all relay states and perform logical OR and AND operations to modify individual bits before writing a new output register value.

A cleaner, more efficient, method is to use the XIO-RLY8’s built-in regBitSet and regBitReset commands. These two commands allow relays to be individually controlled without affecting the state of adjacent channels.

1. Open I2C communications to XIO-RLY8 at its I2C address, i.e.; 0x30 default address
2. Write the Command/Parameter byte with a value to activate or deactivate the desired relay channel

Relay Channel :	0	1	2	3	4	5	6	7
To activate relay :	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47
To deactivate relay :	0x60	0x61	0x62	0x63	0x64	0x65	0x66	0x67

3. Write the data byte value = 0x01.
4. Close I2C communications

I2C Bus Speed and Latency

The XIO-RLY8 communicates at standard (100kHz) and Fast (400kHz) I2C bus speeds. After each communication message the XIO-RLY8 requires a certain period of time to process the command, typically within a few milliseconds. Hosts that are rapidly issuing consecutive messages to the XIO-RLY8 or running particularly fast may need to insert a short timing delay between messages.

Appendix - A Specifications

Specifications subject to change without notice

General:

Description: Eight channel relay output board with Qwiic / Stemma QT I2C interface
Power requirement: Jumper determines source, “Power-On / STATUS” LED
EXT: 7-30 VDC @ 100mA typical. Screw terminal or 2.1mm x 5.5 mm barrel jack
HOST: +5Vdc $\pm 5\%$ @ 100mA typical, supplied by Qwiic / Stemma QT connection

Mechanical:

Dimensions: 3.00"W x 6.00"L x 1.35"H
Weight: 8oz
Mounting: Mounts using Standoffs, DIN Rail Clips, Mounting Track, or legacy SNAPTRACK®

Communications:

Connections: Two four wire Qwiic / Stemma QT connections, 1.0mm JST style
Addressing: Requires one unique 7-bit I2C address, Factory default address is 0x30
User programmable to 0x00 – 0x7F, some addresses are reserved and should be avoided
Pull-Ups: Optional jumper enabled 2.0K Pull-Ups on SDA and SCL signals referenced to HOST V+
Bus Voltage: Level-Translator circuitry support 3.3V or 5.0 V Hosts
Protocol / Software: Use standard I2C library functions. Supplied with software examples for Arduino and Raspberry Pi
Speed: Standard (100kbps), Fast (400kbps)
Latency: 5ms maximum, command received to execution/completion

Relay Outputs:

General: Eight SPDT (Form C) sealed electro-mechanical relays, Break-Before-Make operation, “Active” LED for each channel
Connections: Three position Screw-Terminal for each channel. Accepts up to 14 AWG wire
Power handling: DC: 8 Ampere @ 30Vdc maximum (resistive load)
AC: 10 Ampere @ 125V_{RMS} maximum (resistive load)
Switching capacity: 100ma, 5Vdc minimum, 1200 VA, 240W maximum, 250Vac, 125Vdc
Contact resistance: 100m Ω maximum, AgSnO₂ contacts
Operate time: 10ms maximum (activate or release)
Bounce time: Activate: 0.6ms Release: 7.2ms
Switching frequency: 1,800 operations per hour under rated load
Service life: 10,000,000 operations per relay minimum
Isolation: Board to Output: 250V DC or AC
Between adjacent outputs channels: 250V maximum

Environmental:

Operating temperature: -20°C to 70°C Non-condensing relative humidity: 5% to 95%
Compliance: RoHS, Lead-Free, Conflict-Mineral Free
Product origin: Designed, Engineered, and Assembled in U.S.A. by SCIDYNE® Corporation using domestic and foreign components.

Appendix - B Arduino Software Library

A Software library has been developed to simplify applying the XIO-RLY8 when it is used with an Arduino. The library provides several useful and proven routines to control the relays on a byte or individual channel (bit) basis. Use the Add ZIP Library feature within the Arduino IDE to install the compressed .zip library file. After installation, view the example sketches which further illustrate the use of the library's routines. The library routines are described below.

Function: **Library Inclusion and Instantiation**

Syntax: `#include <SCIDYNE_XIORLY8.h>`

`SCIDYNE_XIORLY8    objectName;`

Inputs: objectName: User defined unique name for one instance of the SCIDYNE_XIORLY8 library.

Outputs: None

Description: Include and instantiate the SCIDYNE_XIORLY8 library. A separate object must be instantiated for each XIO-RLY8 that will be controlled by library routines. When using several XIO-RLY8 boards, the multiple libraries can be instantiated using different object names or an array of the same named library object i.e.; `objectName[3];`

To be accessible by all routines within a program instantiation should be done at the global level i.e.; outside of the Arduino `setup()` or `loop()` routines.

Example: `#include <SCIDYNE_XIORLY8.h>    // Include the XIORLY8 library`

`SCIDYNE_XIORLY8    myXIORLY8;    // Create an object of the library named myXIORLY8`

Function: **setup()**

Syntax: `setup ( Address, wireLib );`

Inputs: Address : This is the I2C address of the XIO-RLY8 to be controlled by the SCIDYNE_XIORLY8 library.

 wireLib: Use Wire or Wire1 depending on which Host I2C port the XIO-RLY8 will be connected to. Not all Host boards support Wire1.

Outputs: None

Description: Associates the address of a XIO-RLY8 with the library object and associates Wire or Wire1 as the means for I2C communications.

Example: `// Associate myXIORLY8 with I2C address 0x30 and use Wire for I2C communications`

`SCIDYNE_XIORLY8    myXIORLY8;    // Create an object of the library named myXIORLY8`

`myXIORLY8.setup( 0x30, Wire );`

Function: **setBit(), resetBit()**

Syntax: setBit(bitNumber, bitPattern);
resetBit(bitNumber, bitPattern);

Inputs: bitNumber: The starting bit (i.e.; Relay Channel) where operations will begin
bitPattern: Bits that are logic “1” will affect the corresponding relay. Bits that are logic “0” do not affect relay.

Outputs: None

Description: Control individual or multiple relays. Use setBit() to activate relays, use resetBit() to deactivate relays. Beginning at the relay channel identified by bitNumber, activate or deactivate the corresponding relay(s) for each logic “1” in bit pattern, evaluated right-to-left. Any relay whose corresponding bit pattern is logic “0” is unaffected.

Example:

```
/******  
 * Control Relay #5 – Method 1, by bitNumber  
 */  
myXIORLY8.setBit( 5, 1 );    // Activate relay #5  
delay(1000);                 // Wait 1 second for observation (optional)  
myXIORLY8.resetBit( 5, 1 ); // Deactivate relay #5  
  
/******  
 * Control Relay #5 – Method 2, by bitPattern  
 */  
myXIORLY8.setBit( 0, 0b00100000 ); // Activate relay #5  
delay(1000);                       // Wait 1 second for observation (optional)  
myXIORLY8.resetBit( 0, 0b00100000 ); // Deactivate relay #5  
  
/******  
 * Control multiple relays  
 */  
myXIORLY8.setBit( 0, 0b00100110 ); // Activate relay #5, #2, #1  
delay(1000);                       // Wait 1 second for observation (optional)  
myXIORLY8.resetBit( 0, 0b00100010 ); // Deactivate relay #5, #1. Relay #2 remains activated.
```

Function: **readBit()**

Syntax: bool result readBit(bitNumber);

Inputs: bitNumber : The bit to read

Outputs: Logic “0” if bit is reset (i.e.; relay is deactivated), Logic “1” if bit is set (i.e; Relay is activated).

Description: Returns the state of the bit identified by bitNumber. Valid values are 0 – 7 which correspond to the relay channels.

Example:

```
/******  
 * Read state of relay 6  
 * A returned logic “1” indicates relay is activated  
 */  
bool state;  
state = myXIORLY8.readBit( 6 ); // Read state of relay #6
```

Function: **byteWrite()**

Syntax: portWrite(byte);

Inputs: byte = The 8-bit value that will be written to the relays.

Outputs: None

Description: Writes an 8-bit value to the relays. Bit#0 controls Relay#0, Bit#1 controls Relay#1 and so on. A bit that is logical “1” will activate its corresponding relay. A bit that is logical “0” will deactivate its corresponding relay.

Example:

```
/******  
 * Control all relays at once  
 */  
myXIORLY8.byteWrite( 0b00000000 ); // Deactivate all the relays  
myXIORLY8.byteWrite( 0b11100000 ); // Activate relay #7, #6, #5
```

Function: **byteRead()**

Syntax: Returned_Value = byteRead();

Inputs: None

Outputs: Unsigned 8-bit value, each bit represents the state of one Relay

Description: Reads the state of the relays. Returns an 8-bit value where each bit represents the states of its corresponding relay. Bit#0 represents the state of Relay#0, Bit#1 represents the state of Relay #1 and so on. Any bit being logic “1” represents an active relay. Any bit being logic “0” represents a deactivated relay.

Example:

```
/******  
 * Read relays states  
 */  
uint8_t relayStatus = myXIORLY8.byteRead(); // Read state of the relays
```

Function: **getProductInfo()**

Syntax: length = getProductInfo (&bufferAddress);

Inputs: &bufferAddress: The address of a character buffer

Outputs: buffer is filled with the product information string including a terminating NUL character.
length = Number of characters in string less the terminating NULL.

Description: Writes a Null terminated product information string into a character buffer pointed to by &bufferAddress. Returns the number of characters in string, not including the terminating NULL.

Note: It is recommended to size the buffer for at least 25 bytes to accommodate future versions of this function.

Product String Format: PPPP.VV.YYYYMMDD

Where: PPPP = Product or Firmware document number

 VV = Version Number

 YYYYMMDD = Release date; YYYY = Year, MM = Month, DD = Day

Example:

```
/******
```

```
 * Read the product string
```

```
 */
```

```
char buffer[25]; // Will be populated with the Product String including NULL
```

```
int length;      // Holds the string length (Optional)
```

```
length = myXIORLY8.getProductInfo( &buffer ); // Get product String and length
```

***** Junk *****

Open I2C communications to XIO-RLY8 at its I2C address, i.e.; 0x30 default address



Write the Command/Parameter byte with a value of 0x20, i.e.; regRW command



Write the desired Output Register data byte value where each bit controls a corresponding relay; Bits that are “1” activate the relay, bits that are “0” deactivate the relay.



Close I2C communications

Relay Channel	byte value
0	0x40
1	0x41
2	0x42
3	0x43
4	0x44
5	0x45
6	0x46
7	0x47

Open I2C communications to XIO-RLY8 at its I2C address, i.e.; 0x30 default address



Write the Command/Parameter byte with a value to activate the desired relay

Set command value :	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47
Relay Channel :	0	1	2	3	4	5	6	7



Write the desired Output Register data byte value where each bit controls a corresponding relay; Bits that are “1” activate the relay, bits that are “0” deactivate the relay.



Close I2C communications



For even deeper understanding a companion document is available that describes the protocol in great detail.

SCIDYNE I2C Protocol Technical Documentation

regBit Commands

Relay Channel	regBitSet Activate Relay Command byte	regBitReset Deactivate Relay Command byte
0	0x40	0x60
1	0x41	0x61
2	0x42	0x62
3	0x43	0x63
4	0x44	0x64
5	0x45	0x65
6	0x46	0x66
7	0x47	0x67