USER'S REFERENCE MANUAL

DIO24-ARD

24 Bit Digital I/O Interface

Shield for Arduino and Compatible Devices

Model No. 100-7692
Doc. No. M7692 Rev: 3.0 11/8/22

SCIDYNE®

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

www.scidyne.com

Previous revision: V2.0

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 Duration and conditions of warranty may be superseded when the product is integrated into other SCIDYNE products.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 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 Throughout 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

Arduino Through-out this document the term Arduino will be used generically to mean an actual Arduino board or any compatible device which the DIO24-ARD is plugged in to.

Logic Conditions

Unless otherwise noted, logic signals are designated as TRUE (Set) and FALSE (Clear). Names with an asterisk (*) postscript are inverted or active low. Unless otherwise noted TRUE is considered logic '1' (+5Vdc or +3.3Vdc) and FALSE is considered logic '0' (0Vdc).

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 are shown as 0x1234 where 12 represents the most-significant byte and 34 is the least significant byte. Depending on your particular system the values could be internally stored as little-endian or big-endian.

Introduction

The DIO24-ARD is a 24-Bit (channel), non-isolated, digital Input/Output interface specifically designed to provide the high sink current drive capability required by many peripheral components. It attaches to an Arduino by means of the standard R3 connector footprint and uses +5V for operation. Communications between the Arduino and DIO24-ARD is through the SPI (Serial-Peripheral-Interface) bus, via the six pin ICSP connector, and one user designated Arduino digital output acting as a board SELECT signal.

Within the context of this document the term channel refers to an Input/Output stage which connects the DIO24-ARD to external devices. The term bit refers to one internal logical signal associated with a channel. A port is a group of related eight bits / channels.

The 24 channels appear across three 8-bit ports, A, B, and C, as shown in the block diagram below. Each channel may be individually operated as ether an input or output under software control. Channels configured as outputs can passively source 1.0ma and actively sink 85ma, sufficient sink current to reliably drive solid-state and mechanical relays. External components attach to the DIO24-ARD by means of a 50 conductor IDC flat-ribbon cable type connector. The pin-out of this connector is compatible with industry standard solid-state I/O module racks. The DIO24-ARD is also designed to connect to commonly encountered components including LED's, Switches, and Relays.

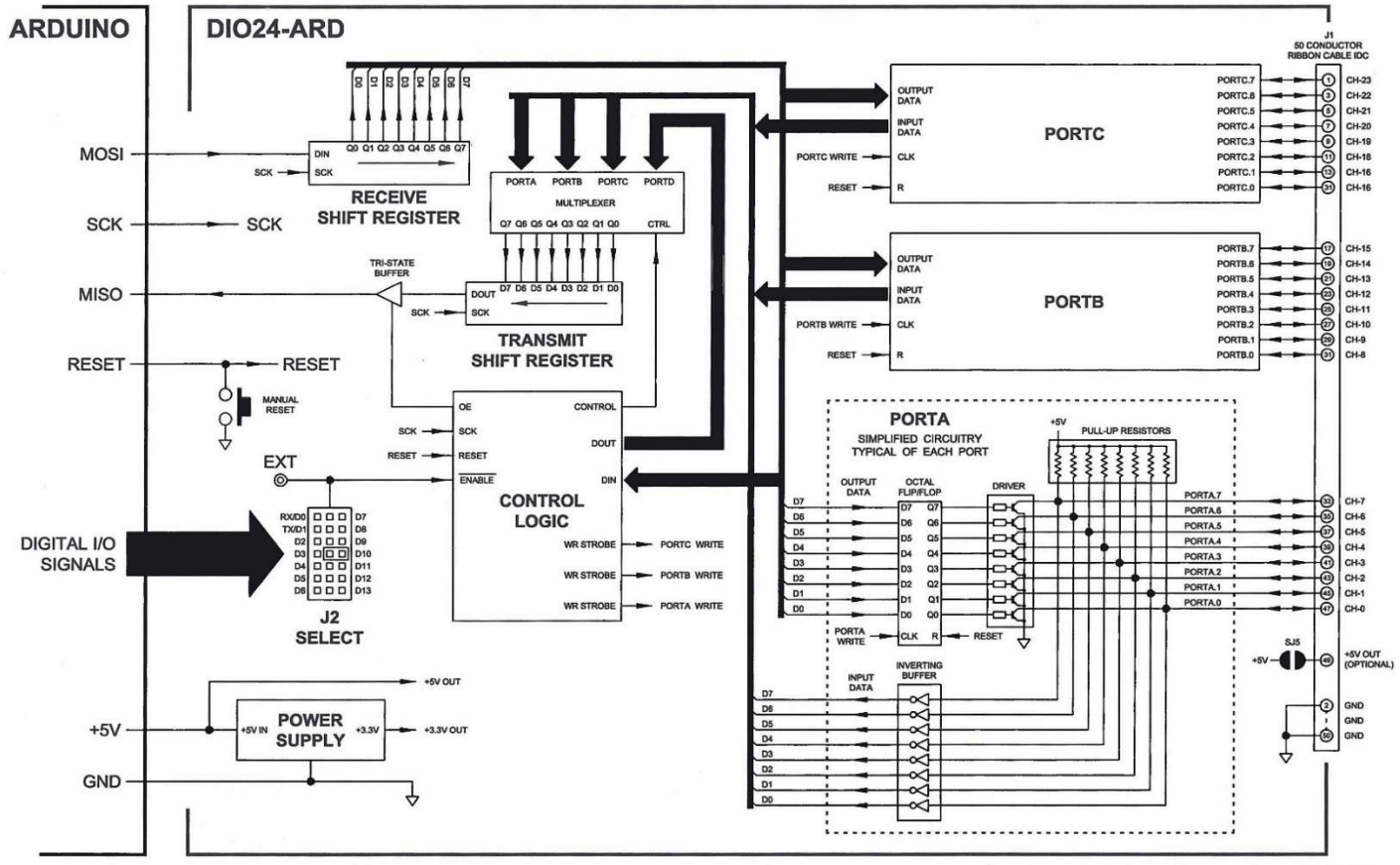


Figure 1 - DIO24-ARD Block Diagram

Before placing the DIO24-ARD into service it is helpful to become familiar with the location and purpose of some important components. Below is a figure showing their locations followed by a brief description of each.

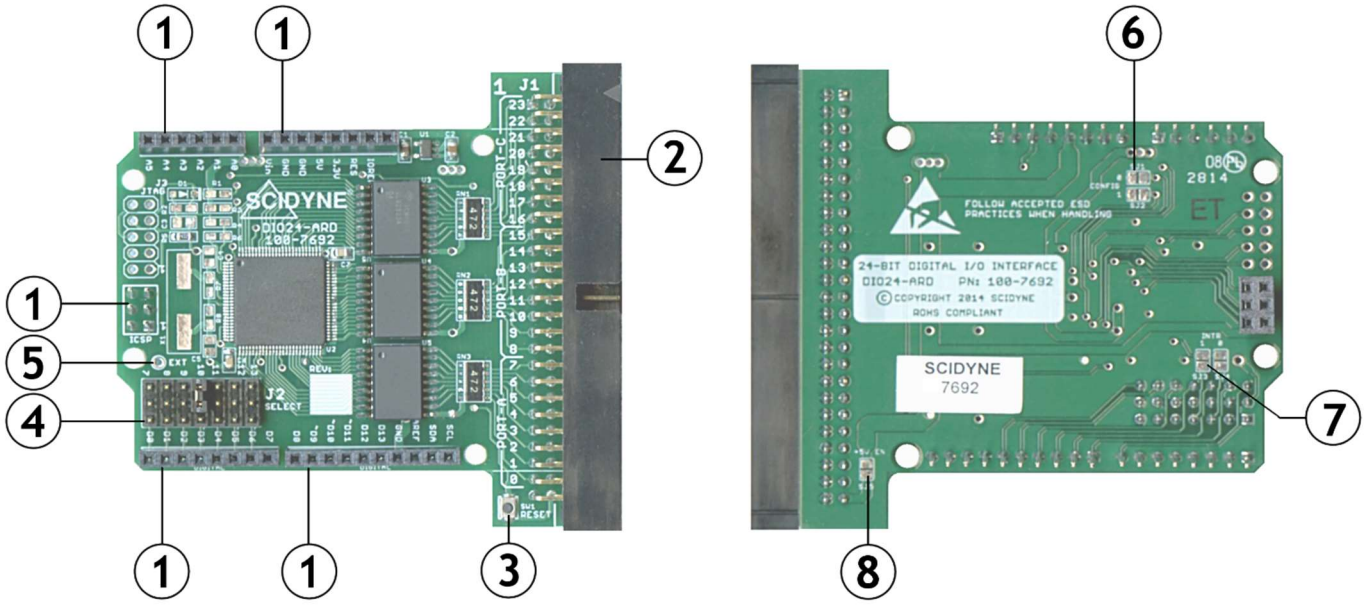


Figure 2 - Component Identification

(1) Arduino Headers

These are the five standard Arduino R3 headers. The DIO24-ARD features stack-through connections permitting multiple Arduino shields to be placed on top of one another.

(2) J1, I/O Connector

This 50 pin IDC flat-ribbon cable connector is used to attach external devices to the DIO24-ARD. All even numbered pins 2 - 50 are connected to signal ground. Each of the odd numbered pins 1 - 47 connect to I/O signals where Pin#1 is I/O channel #23, Pin#3 is I/O channel #22, and so on continuing to Pin#47 which is I/O channel #0.

Pin #49 is not used but reserved as an Arduino supplied +5V source for externally connected devices. See item (8) **+5V ENable** for more details and important cautions.

(3) RESET Push Button

Momentarily pressing this button will reset the entire Arduino system.

(4) J2, SELECT

A shunt placed between the center row and one of the 14 outside positions selects which Arduino digital I/O signal will be used as the designated enable for the DIO24-ARD during SPI communications. The remaining 13 digital I/O signals are unaffected and available for other purposes. The silk screen text printed along the outside edge of J2 correspond to the Arduino digital I/O signals. Only one shunt must be installed. The corresponding digital signal must be configured as an output and cannot be used for any other function in an Arduino system.

(5) EXTernal SELECT Input

A wire can be soldered here and the other end driven by a digital signal external to the DIO24-ARD, such as one of the higher numbered digital I/O signals originating from an Arduino MEGA2560 board, or even an available analog input properly reconfigured for digital output operation. See the "Choosing a SELECT signal" section of this manual for details on using this feature.

(6) Custom Configuration, CONFIG-0, CONFIG-1

These two Solder Jumpers, SJ1 and SJ2, are used for customized versions of the DIO24-ARD. To maintain future compatibility, they should be left in the factory default un-soldered, OPEN, position.

(7) Interrupts, INTR-0, INTR-1

These two solder jumpers, SJ3 and SJ4, are provided for customized versions of the DIO24-ARD. To maintain future compatibility, they should be left in the factory default un-soldered, OPEN, position.

(8) +5V ENable

Soldering these two pads of SJ5 together will route the Arduino supplied +5V to pin #49 of connector J1.

J1 pin #49 is not used but reserved as an Arduino supplied +5V source for externally connected devices. Most present versions of Arduino hardware are incapable of providing an adequate amount of current to operate a fully loaded I/O module rack. It is therefore recommended that the +5V Enable Solder Jumper SJ5 (8) remain in the factory default un-soldered, OPEN, position, and any external circuitry be powered by an external +5V power supply. In addition, keeping SJ5 (8) open also prevents the possibility of damage caused by an external power source back-feeding +5V in to the DIO24-ARD and Arduino system.

Hardware Details

External devices attach to the DIO24-ARD through J1, a 50 pin IDC flat-ribbon type connector. The Pin Out follows the industry standard for 8, 16 and 24 position solid-state I/O racks and is compatible with products offered by numerous manufactures.

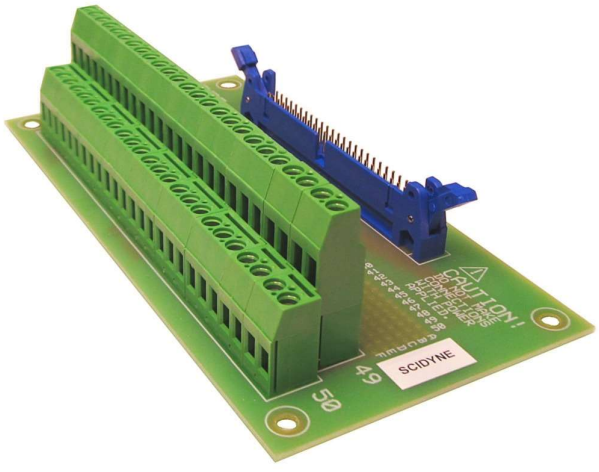
Notes:

1) All even numbered pins 2 - 50 are connected to signal ground.

2) Pin #49 is intentionally not connected by default. It can be optionally configured to supply Arduino produced +5V to externally connected device by shorting solder jumper SJ5.

Most present versions of Arduino hardware are incapable of providing an adequate amount of current to operate a fully loaded I/O module rack. It is therefore recommended that the +5V Enable Solder Jumper SJ5 remain in the factory default un-soldered, OPEN, position, and any external circuitry be powered by a separate +5V power supply. In addition, keeping SJ5 open also prevents the possibility of damage caused by an external power source back-feeding +5V in to the DIO24-ARD and Arduino system.

The DIO24-ARD is also intended to interface with switch contacts, relays, LEDs and many other common peripheral devices. A companion 50-position Screw-Terminal board, SCIDYNE Part Number: **100-7625/50**, is available separately to make field wiring to these components straightforward.



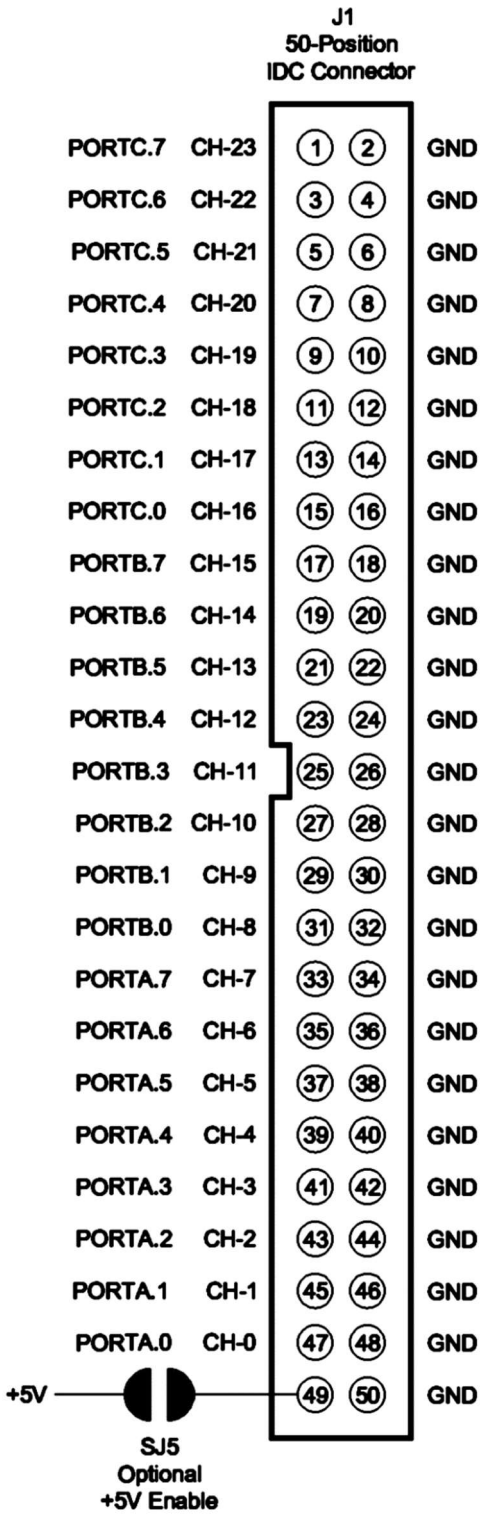


Figure 3 - J1 Connector

I/O stage operation

Each of the 24 I/O channels use an identical circuit design. The arrangement allows each channel to be independently operated as either an Input or an Output under software control as illustrated and described below.

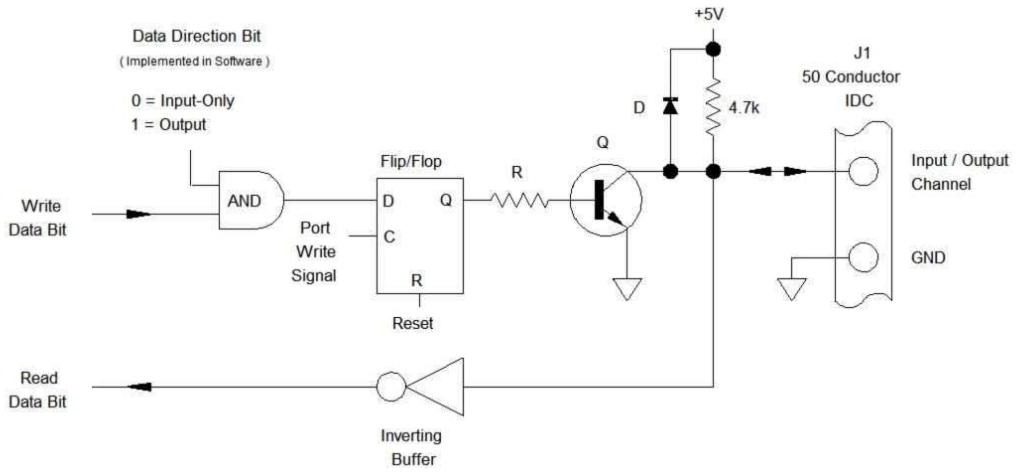


Figure 4 - Simplified I/O Channel Circuitry

Input Mode:

Upon a system reset, the flip/flop is cleared deactivating transistor Q and placing it in a high impedance state. The I/O channel is pulled high (+5V) by virtue of the 4.7k ohm resistor. Its value has been chosen to supply reasonable source current while the I/O stage functions as an output yet be capable of being overridden by external devices when used as an input. The level of the I/O channel is also routed to the input of the inverting buffer. The inverted state of the I/O channel is read back by the Arduino during SPI communications with the DIO24-ARD. As described, the I/O channel being pulled high causes a logic "0" to be read for this channel. If an external device were to pull the I/O channel to ground, then a logic "1" would be read instead. If necessary, the application software can perform an invert operation if the actual state of the I/O channel must be represented.

*Any channel being used exclusively as input-only **MUST ALWAYS** have its data bit written as logic "0" whenever writing to its associated port. This requirement is automatically performed in the background when using the supplied library routines. By virtue of the software AND operation, any Data Direction bit that is "0" will force a "0" to be written to the flip/flop irrespective of the intended Write Data Bit state.*

Output Mode:

The data direction bit associated with the channel must be "1" to allow output write operations. When the I/O stage functions as an output, the 4.7k ohm pull-up resistor supplies up to 1.0ma of source current to external devices whenever the flip/flop is at logic "0". Performing a write operation with the corresponding I/O channel bit at logic "1" sets the flip/flop which activates the transistor. The transistor pulls the I/O channel hard to ground providing a low impedance path capable of sinking 85ma. A read operation of this bit essentially returns the logic state stored in the flip/flop due to the double inversion created by the transistor and buffer.

Example Interface Circuits

The DIO24-ARD is fully compatible with industry standard I/O modules and the J1 connector can directly connect to 8, 16, and 24 position racks. However, the versatility of the DIO24-ARD I/O channel circuitry makes the DIO24-ARD well suited to satisfy a wide variety of other commonly encountered interfacing needs. The following example circuits illustrate the versatility of the DIO24-ARD and a few ways it can be applied.

Reading a Switch or Contact Closure

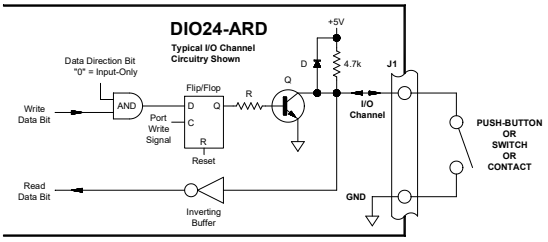


Figure 5

Any channel configured as an input can be used to read the state of a Push-Button, Switch, or Dry Contact. It's important to observe that whenever writing to the PORT associated with an input-only channel, the channel's data bit is always written as logic "0". This assures the Flip/Flop output is always at "0" forcing transistor Q off and keeping the channel exclusively configured as an input. The accompanying library routines take care of this provided the channels Data Direction bit has been initialized to “0”.

When the switch is open the 4.7k pull-up resistor to +5V causes logic "1" to be presented to the inverting buffer. Closing the switch provides a path to ground and makes the input to the inverting buffer logic "0".

If the switch is located a considerable wiring distance away from the DIO24-ARD it is recommended that connections be made using shielded cable to improve noise immunity.

Switch State	Read Data Bit
Switch Open	Logic "0"
Switch Closed	Logic "1"

Driving a Relay Coil

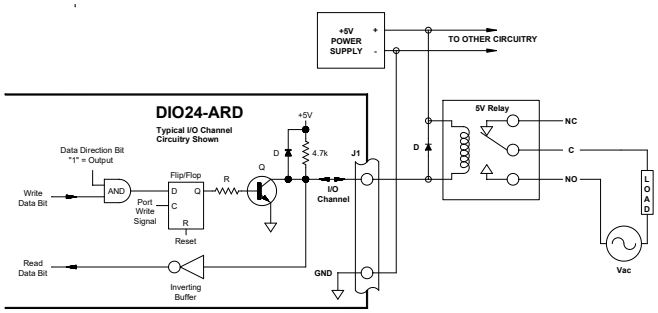


Figure 6

Mechanical relays inherently provide electrical isolation and can control loads far exceeding the voltage and current capability of the driving source. Any number of DIO24-ARD channels can be used to drive 5V mechanical relays. The only stipulations are that a relay coil must be rated for +5V operation and have a DC resistance no lower than 60 ohms. In addition, an external +5V power supply is recommended because the standard Arduino hardware is limited on the amount of 5V power it can provide to operate attached devices, especially if the Arduino is powered via USB.

Writing to the associated PORT with the channel's data bit as "0" resets the Flip/Flop turning off transistor Q resulting in the relay being de-energized. Writing the channel's data bit as "1" sets the Flip/Flop turning on transistor Q which activates the relay by providing a path to ground for the relay coil. Reading the channel bit returns the state of the flip/flop. The diode across the relay coil is recommended to suppress back EMF generated whenever the relay coil is de-energized.

Write Data Bit	Relay State, Connection made	Read Data Bit
Logic "0"	Relay De-Energized, C-to-NC	Logic "0"
Logic "1"	Relay Energized, C-to-NO	Logic "1"

C = Common, NC = Normally Closed contact, NO = Normally Open contact

Driving LED's

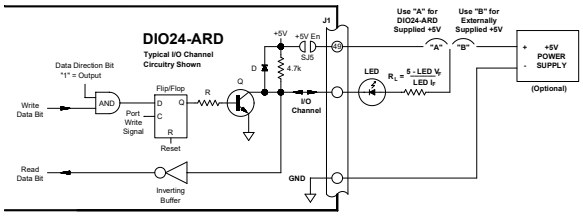


Figure 7

The DIO24-ARD can drive LEDs as shown. Writing to the associated PORT with the channel's data bit as "0" resets the Flip/Flop turning off transistor Q resulting in the LED also being off. Writing the channel's data bit as "1" sets the Flip/Flop turning on transistor Q which consequently illuminates the LED by providing a path to ground. A suitable limiting resistor must be placed in series with the LED to prevent excessive current draw.

Although an external power supply is depicted, the required +5V can be provided by the Arduino instead (via the DIO24-ARD) by shorting solder jumper SJ5. Be sure to consider all the loads that must be supplied when choosing the Arduino as the +5V power source.

Write Data Bit	LED
Logic "0"	Off
Logic "1"	On

Controlling High-Power Loads

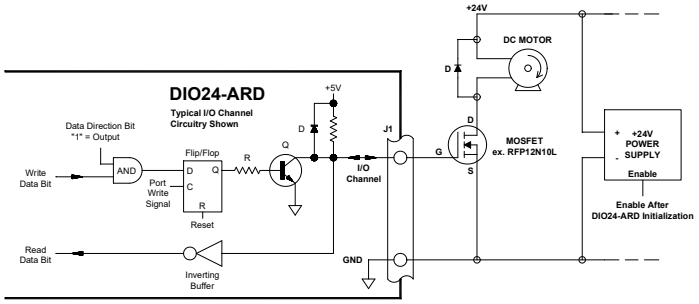


Figure 8

It is sometimes necessary to control loads which require higher voltage, current, or power than the DIO24-ARD itself can provide. By adding an external transistor these requirements are easily satisfied. This example shows one DIO24-ARD channel feeding the gate of an N-Channel MOSFET transistor which then drives a +24V load. Although a DC motor is shown, a solenoid, relay, alarm, LED string, or other high-power load could easily be substituted.

Writing to the associated PORT with the channel's data bit as "0" resets the Flip/Flop, turning off transistor Q which allows the channel output to be pulled to +5V by virtue of the 4.7k pull-up resistor. This activates the MOSFET and consequently the motor. Alternatively, writing the associated PORT channel's data bit as "1" sets the flip/flop, turns on transistor Q, which clamps the MOSFET's gate to ground causing it and the motor to turn off. The diode across the motor is optional but is often included to suppress possible back EMF whenever an inductive load, such as a motor or relay coil, is suddenly de-energized.

Particular design consideration should be given regarding system reset. Since the Flip/Flop is forced to "0" during system reset this simple example results in the motor being activated. In a practical application, the +24V supply would only be enabled after the DIO24-ARD has been properly initialized.

Write Data Bit	Load State
Logic "0"	Motor On
Logic "1"	Motor Off

Sensing Open-Collector Outputs

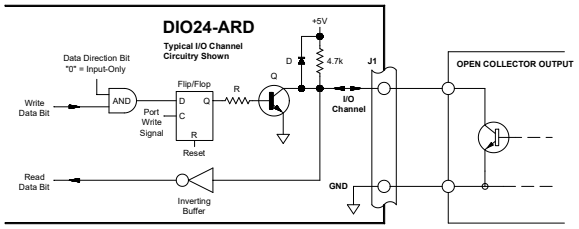


Figure 9

Some devices, such as Hall-effect switches and Gear Tooth sensors, signal their operation using an "Open-Collector Transistor Output". Their transistor output behaves like a switch by providing a low impedance path to ground when activated. Any DIO24-ARD channel configured as an input can be used to sense the operation. It's important to observe that whenever writing to the PORT associated with the channel, that the channel's data bit is always written as logic "0". This assures the channel remains configured as an input by maintaining the Flip/Flop output at "0" and keeping transistor Q in an off, non-conductive, state. The accompanying library routines take care of this provided the channels Data Direction bit has been initialized to "0".

When the device's Open-Collector output is off the 4.7k pull-up resistor to +5V causes logic "1" to be presented to the inverting buffer. Conversely, when the device Open-Collector output is on, it pulls the input to the inverting buffer to ground i.e., logic "0".

If the Open-Collector device is located a considerable wiring distance away from the DIO24-ARD it is recommended that connections be made using shielded cable to improve noise immunity.

Open-Collector Output	Read Data Bit
Transistor Off (Open)	Logic "0"
Transistor On (Closed)	Logic "1"

Software Details

This section covers the software aspects necessary to operate the DIO24-ARD. Where noted the accompanying library and software examples are available for downloading on the ***scidyne.com*** website.

Choosing a SELECT signal

The DIO24-ARD requires one Arduino digital output to be used exclusively as the board SELECT signal. Once properly configured, in hardware and software, the signal will enable the DIO24-ARD on the SPI bus whenever logic "0" and disable it whenever logic "1".

Using onboard SELECT

The J2 SELECT conveniently allows an available Arduino digital signal to be used by placing a shunt between the center row and one of the 14 outside positions. The other 13 digital I/O signals are unaffected and available for other purposes. Position #10 is the Factory Default position for the DIO24-ARD. Several of the Arduino digital signals are pre-assigned or have a de-facto usage and may not be available. The following table provides a brief summary of the standard Arduino digital I/O signals. The information should be carefully compared against your particular Arduino hardware and application.

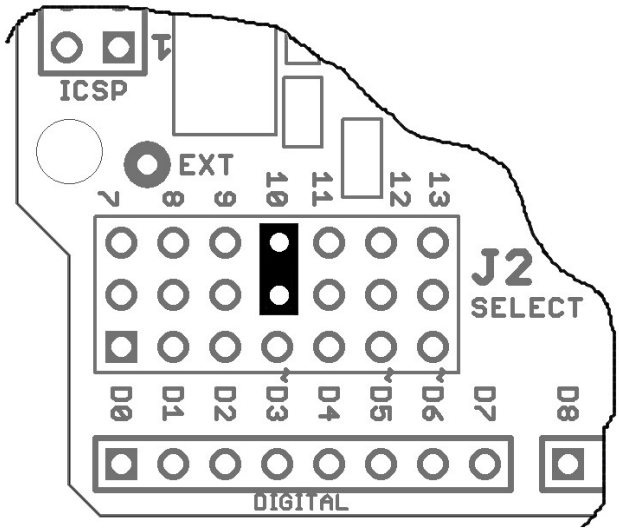


Figure 10 - J2 Select (Factory Default)

Arduino Typical Digital I/O Usage		
Digital I/O Designation	Typical Usage	Comment
0	RX, Serial Communications	Avoid if using serial communications
1	TX, Serial Communications	Avoid if using serial communications
2	Interrupt-0	Possible conflict if interrupt-0 is used
3	Interrupt-1	Possible conflict if interrupt-1 is used
4	Chip Select for SD Memory	Possible conflict if SD Memory is also used
5	Unassigned	Available if not used elsewhere by application
6	Unassigned	Available if not used elsewhere by application
7	Unassigned	Available if not used elsewhere by application
8	Unassigned	Available if not used elsewhere by application
9	Unassigned	Available if not used elsewhere by application
10	SS, Chip Select for EtherNet chip	**DIO24-ARD Factory Default,** Possible conflict if Ethernet is used
11	MOSI	Hard wired to ICSP MOSI signal on Arduino UNO
12	MISO	Hard wired to ICSP MISO signal on Arduino UNO
13	SCK	Hard wired to ICSP SCK signal on Arduino UNO

Using EXTernal SELECT Input

If necessary, the DIO24-ARD can be externally enabled using a digital signal other than those presented on J2 SELECT. This is particularly handy if all of the standard Arduino digital I/O signals are already being used for other purposes, but an external digital signal is available. For example, one of the higher numbered digital I/O signals originating from an Arduino MEGA2560 board.

To use this feature a wire must be soldered at the EXT input pad on the DIO24-ARD. The other end of the wire is then connected to and driven by the external digital signal. In addition, the J2 SELECT shunt must be positioned to prevent a conflict between the external digital signal and the 14 standard digital signals. This is accomplished by installing the J2 shunt between any two adjacent positions in the center row.

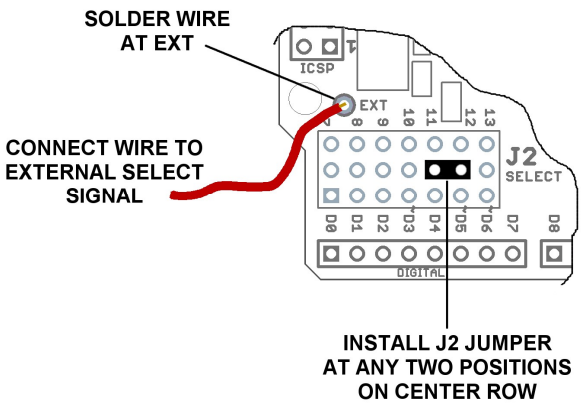


Figure 11 - External Select Input

Any signal used for the External Select must meet the digital input voltage levels described in the specification section of this manual. Exceeding the levels by over-driving or under-driving, even momentarily, could damage the DIO24-ARD and devices attached to it.

DIO24-ARD Arduino Library

SCIDYNE has developed an Arduino library to simplify applying the DIO24-ARD. The library provides several useful and proven routines to manipulate individual bits, write and read ports, and implement software-based Data Direction for each channel. The library is supplied as a compressed .zip file that is installed using the Add ZIP Library feature within the Arduino IDE. After installation, examples sketches are available which further illustrate the use of the library routines.

Function: **Library Inclusion and Instantiation**

Syntax:

```
#include <SCIDYNE_DIO24ARD.h>

SCIDYNE_DIO24ARD      objectName;
```

Inputs: objectName: User defined unique name for one instance of the SCIDYNE_DIO24ARD library.

Outputs: None

Description: Include and instantiate the SCIDYNE_DIO24ARD library. A separate object must be instantiated for each DIO24-ARD that will be controlled by library routines. When using several DIO24-ARD 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_DIO24ARD.h>   // Include the DIO24-ARD library

SCIDYNE_DIO24ARD   DIO24ARD;    // Create an object of the library called DIO24ARD

DIO24ARD.setup( 10, 0xFF, 0xFF, 0xFF );   // Perform Setup
```

Function: **setup()**

Syntax: setup (pinNumber, portA_DDR, portB_DDR, portC_DDR);

Inputs: PinNumber: Arduino digital I/O pin to be used exclusively as the DIO24-ARD select signal for SPI transactions. Each DIO24-ARD requires a different SELECT pin.

portA_DDR: Initialize portA (Channels [7:0]) Data Direction

portB_DDR: Initialize portB (Channels [15:8]) Data Direction

portC_DDR: Initialize portC (Channels [23:16]) Data Direction

Outputs: None

Description: Setup does the following:

- Configures pinNumber as an output to serve as the SELECT signal. Defaults to logic "1", de-selecting the DIO24-ARD on the SPI bus.
- Channel Data Direction is initialized. Channels that will be used as input-only must have their corresponding Data Direction bits set to "0". Output channels must have their corresponding Data Direction bits set to "1". Data Directions can be changed later by a calling setDDR().
- The Arduino Serial-Peripheral-Interface is activated in its default configuration.

Example:

```
/**
 * Setup a DIO24-ARD to use Pin Number 10 and the Data Directions as follows:
 * portA (channels 7:0) as Input-only
 * portB (channels 15:8) as Output
 * portC (channels 19:16) as Input-only
 * portC (channels  23:20) as Outputs
 */
DIO24ARD.setup( 10, 0x00, 0xFF, 0xFE );
```

Function: **writeDDR()**

Syntax: writeDDR (DDR_to_Write, DDR_value);

Inputs: DDR_to_Write : 0, 1, or 2 to set the Data Direction of portA, portB, or portC respectively

DDR_value : unsigned 8-Bit value. Bits set to "1" are outputs, "0" bits are input-only.

Outputs: None

Description: Writes the Data Direction for the designated port. Writes an unsigned 8-bit value where each bit corresponds to a specific channel. Bits set to "1" will be outputs, bits that are 0 will be input-only.

portA [7:0] corresponds to channels [7:0]
portB [7:0] corresponds to channels [15:8]
portC [7:0] corresponds to channels [23:16]

It is important that "0" is always written to any channels that will be used as Input-Only. The library routines manage this in the background by logically ANDing the data being written against the Data Direction registers. Any Data Direction register bit set to "0" will result in a write operation always being "0" regardless of the intended state.

Example:

```
/**
 * Setup Data Directions as follows:
 *   portA (channels 7:0) as Output
 *   portB (channels 15:8) as Input-only
 *   portC (channels 19:16) as Input-only
 *   portC (channels  23:20) as Outputs
 */
DIO24ARD.writeDDR( 0, 0xFF );   // Write portA DDR
DIO24ARD.writeDDR( 1, 0x00 );   // Write portB DDR
DIO24ARD.writeDDR( 2, 0xF0 );   // Write portC DDR
```

Function: **readDDR()**

Syntax: readDDR (DDR_to_Read);

Inputs: DDR_to_Read : 0, 1, or 2 for the Data Direction of portA, portB, or portC respectively

Outputs: Unsigned 8-bit value

Description: Returns 8-bit Data Direction of the designated port. Channel bits that are "0" are input-only. Channel bits that are "1" are outputs.

Example:

```
#include <SCIDYNE_DIO24ARD.h>          // Include and Instantiate the DIO24-ARD library
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*************************************************************
  *  Setup the DIO24-ARD board and library
  *  - Arduino digital I/O #10 designated as DIO24-ARD select on SPI bus
  *  - Initially configure all DIO24-ARD channels as input-only
  */
 DIO24ARD.setup(10, 0x00, 0x00, 0x00);  // pin#10, portA_DDR, portB_DDR, portC_DDR

 Serial.begin(115200);    // Initialize serial communications for Serial Monitor
}

void loop()
{
 /*************************************************************
  * Send the ports Data Directions to the Serial Monitor
  */
  Serial.println(F("Initial Data Directions"));

  Serial.print(F("PortA_DDR: "));          // Display portA Data Direction
  Serial.println(DIO24ARD.readDDR(0),HEX);

  Serial.print(F("PortB_DDR: "));          // Display portB Data Direction
  Serial.println(DIO24ARD.readDDR(1),HEX);

  Serial.print(F("PortC_DDR: "));          // Display portC Data Direction
  Serial.println(DIO24ARD.readDDR(2),HEX);

  Serial.println();

 /*************************************************************
  * Now change the Data Direction of portB to all outputs. The Data Direction of portA and PortC are unaffected and remain
  * configured as input-only.
  */
  DIO24ARD.writeDDR(1, 0xFF);

 /*************************************************************
  * Send the new ports Data Directions to the Serial Monitor
  */
  Serial.println(F("New Data Directions"));

  Serial.print(F("PortA_DDR: "));          // Display portA Data Direction
  Serial.println(DIO24ARD.readDDR(0),HEX);

  Serial.print(F("PortB_DDR: "));          // Display portB Data Direction
  Serial.println(DIO24ARD.readDDR(1),HEX);

  Serial.print(F("PortC_DDR: "));          // Display portC Data Direction
  Serial.println(DIO24ARD.readDDR(2),HEX);

  Serial.println();

  while(1);  // Loop here forever. Press board reset to repeat program
}
```

Function: **setBit()**

Syntax: setBit (bit_Number);

Inputs: bit_Number = The channel to set

Outputs: None

Description: Sets the channel output identified by bit_Number. Valid range is 0 to 23.

Channels [7:0] corresponds to portA [7:0]
Channels [15:8] corresponds to portB [7:0]
Channels [23:16] corresponds to portC [7:0]

Notes:

1. The DIO24-ARD hardware inverts the state written. Setting a bit to "1" results in a "0" on the J1 connector pin for that channel.

2. Channels configured for input-only cannot be set.

Example:

```
/*************************************************************
 * Instantiate the DIO24-ARD library
 */
#include <SCIDYNE_DIO24ARD.h>
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*************************************************************
  *  Setup the DIO24-ARD board and library
  *  - Arduino digital I/O #10 used as DIO24-ARD select on SPI bus
  *  - All DIO24-ARD channels configured as outputs
  */
 DIO24ARD.setup(10, 0xFF, 0xFF, 0xFF);  // pin#10, portA_DDR, portB_DDR, portC_DDR
}

void loop()
{
 /*************************************************************
  * Continuously toggle DIO24-ARD channel #4. All other
  * channels are unaffected
  */
 delay(5);                 // Wait 5mS
 DIO24ARD.setBit(4);       // This causes channel #4 to go low
 delay(5);                 // wait 5mS
 DIO24ARD.clrBit(4);       // This causes channel #4 to go high
}
```

Function: **clrBit()**

Syntax: clrBit (bit_Number);

Inputs: bit_Number = The channel to clear

Outputs: None

Description: Clears the channel output identified by bit_Number. Valid range is 0 to 23.

Channels [7:0] corresponds to portA [7:0]
Channels [15:8] corresponds to portB [7:0]
Channels [23:16] corresponds to portC [7:0]

Notes:

1. The DIO24-ARD hardware inverts the state written. Setting a bit to "0" results in a "1" on the J1 connector pin for that channel.

2. Channels configured for input-only cannot be cleared.

Example:

```
/*************************************************************
 * Instantiate the DIO24-ARD library
 */
#include <SCIDYNE_DIO24ARD.h>
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*************************************************************
  *  Setup the DIO24-ARD board and library
  *  - Arduino digital I/O #10 used as DIO24-ARD select on SPI bus
  *  - All DIO24-ARD channels configured as outputs
  */
 DIO24ARD.setup(10, 0xFF, 0xFF, 0xFF);  // pin#10, portA_DDR, portB_DDR, portC_DDR
}

void loop()
{
 /*************************************************************
  * Continuously toggle DIO24-ARD channel #4. All other
  * channels are unaffected
  */
 delay(5);                 // Wait 5mS
 DIO24ARD.setBit(4);       // This causes channel #4 to go low
 delay(5);                 // wait 5mS
 DIO24ARD.clrBit(4);       // This causes channel #4 to go high
}
```

Function: **bitSetClr()**

Syntax: bitSetClr (bit_Number, state);

Inputs: bit_Number = The channel to Set or Clear
state = "1" to set or "0" to clear

Outputs: None

Description: Set or Clear the channel output identified by bit_Number. Valid range is 0 to 23.
Use state = "1" to set, or state = "0" to clear

Channels [7:0] corresponds to portA [7:0]
Channels [15:8] corresponds to portB [7:0]
Channels [23:16] corresponds to portC [7:0]

Notes:

1. The DIO24-ARD hardware inverts the state written. Setting a bit to "1" results in a "0" on the J1 connector pin for that channel. Clearing a bit to "0" results in a "1" on the J1 connector pin for that channel.

2. Channels configured for input-only cannot be cleared.

Example:

```
/*************************************************************
 * Instantiate the DIO24-ARD library
 */
#include <SCIDYNE_DIO24ARD.h>
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*************************************************************
  *  Setup the DIO24-ARD board and library
  *  - Arduino digital I/O #10 will be designated as DIO24-ARD select on SPI bus
  *  - All DIO24-ARD channels configured as outputs
  */
 DIO24ARD.setup(10, 0xFF, 0xFF, 0xFF);  // pin#10, portA_DDR, portB_DDR, portC_DDR
}

void loop()
{
 /*************************************************************
  * Continuously toggle DIO24-ARD channel #4. All other channels are unaffected
  */
 delay(5);                    // Wait 5mS
 DIO24ARD.bitSetClr(4, 1);    // This causes channel #4 to go low
 delay(5);                    // Wait 5mS
 DIO24ARD.bitSetClr(4, 0);    // This causes channel #4 to go high
}
```

Function: **readBit()**

Syntax: X = readBit(bitNumber);

Inputs: bit_Number = The channel to be read

Outputs: Boolean “0” or “1”

Description: Reads the digital state of the channel identified by bitNumber. Valid range is 0 to 23. This function works equally on channels configured as inputs or outputs.

Channels [7:0] corresponds to portA [7:0]
Channels [15:8] corresponds to portB [7:0]
Channels [23:16] corresponds to portC [7:0]

Notes:

1. The DIO24-ARD hardware inverts the channel state. If the channel on connector J1 is “1” the readBit() function returns “0”. Similarly, if the channel on connector J1 is “0” the readBit() function returns “1”.

Example:

```
#include <SCIDYNE_DIO24ARD.h>
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*************************************************************
  *  Setup the DIO24-ARD board and library
  *  - Arduino digital I/O #10 designated as DIO24-ARD select on SPI bus
  *  - Configure DIO24-ARD channels 7:0 (portA) as inputs and all others as outputs.
  */
 DIO24ARD.setup(10, 0x00, 0xFF, 0xFF);  // pin#10, portA_DDR, portB_DDR, portC_DDR

 // Initialize serial communications for Serial Monitor
 Serial.begin(115200);
}

void loop()
{
 /*************************************************************
  * Send the state of channels 7:0 (portA) to the Serial Monitor
  * Note: Open the Serial Monitor in the Arduino IDE to view.
  */

 Serial.print(F("Channel-0: "));  Serial.println( DIO24ARD.readBit(0) , BIN);
 Serial.print(F("Channel-1: "));  Serial.println( DIO24ARD.readBit(1) , BIN);
 Serial.print(F("Channel-2: "));  Serial.println( DIO24ARD.readBit(2) , BIN);
 Serial.print(F("Channel-3: "));  Serial.println( DIO24ARD.readBit(3) , BIN);
 Serial.print(F("Channel-4: "));  Serial.println( DIO24ARD.readBit(4) , BIN);
 Serial.print(F("Channel-5: "));  Serial.println( DIO24ARD.readBit(5) , BIN);
 Serial.print(F("Channel-6: "));  Serial.println( DIO24ARD.readBit(6) , BIN);
 Serial.print(F("Channel-7: "));  Serial.println( DIO24ARD.readBit(7) , BIN);

 Serial.println(F("------------"));
 delay(500);  // Delay added to slow down screen scroll
}
```

Function: **portWrite()**

Syntax: portWrite(port_to_Write, port_Byte);

Inputs: port_to_Write = The port that will be written to; portA = 0, portB = 1, portC = 2

port_Byte = 8-bit value that will be written

Outputs: None

Description: Writes an 8-bit value (port_Byte) to the designated port (port_to_Write)

Notes:

1. Any channel of a port being used as an output must have the corresponding bits of the ports Data Direction register set to "1" to enable output operations. If not "0" will be written to those channels.

2. The DIO24-ARD hardware inverts the state written. Writing a bit as "1" results in a "0" on the J1 connector pin for that channel. Writing a bit as "0" results in a "1" on the J1 connector pin for that channel.

Example:

```
#include <SCIDYNE_DIO24ARD.h>
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*****************************************************************
  *  Setup the DIO24-ARD board and library software
  *  - Arduino digital I/O #10 used as DIO24-ARD select on SPI bus
  *  - The portA channels configured as inputs, portB and portC configured as outputs
  */
 DIO24ARD.setup(10, 0x00, 0xFF, 0xFF);  // pin#10, portA_DDR, portB_DDR, portC_DDR

 Serial.begin(115200);   // Initialize serial communications for Serial Monitor
}

void loop()
{
 uint8_t counter = 0x00;                          // An 8-bit counter initialized to 0x00
 uint8_t portA_data, portB_data, portC_data;      // Will hold the port data after a read

 while(1)  {
  // Continuously toggle the bits of portB by echoing the 8-bit counter value.
  DIO24ARD.portWrite(1, counter);          // Write portB with counter value
  counter++;                                       // Increment the 8-bit counter

  // Read the three ports
  portA_data = DIO24ARD.portRead(0);   // Read and store portA data
  portB_data = DIO24ARD.portRead(1);   // Read and store portB data
  portC_data = DIO24ARD.portRead(2);   // Read and store portC data

  // Send the port data to the serial monitor.  Note: Open the Serial Monitor in the Arduino IDE to view.
  Serial.print(F("PortA: "));       Serial.println(portA_data,HEX);         // Display portA data
  Serial.print(F("PortB: "));       Serial.println(portB_data,HEX);         // Display portB data
  Serial.print(F("PortC: "));       Serial.println(portC_data,HEX);         // Display portC data

  Serial.println();
 }
}
```

Function: **portRead()**

Syntax: Returned Value = portRead(port_to_Read);

Inputs: port_to_Read = The port that will be read; portA = 0, portB = 1, portC = 2

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

Description: Reads the 8-bit value of the designated port (port_to_Read)

Notes:

1. The DIO24-ARD hardware inverts the channel state. If the channel on connector J1 is "1" the readBit() function returns "0". Similarly, if the channel on connector J1 is "0" the readBit() function returns "1".

Example:

```
#include <SCIDYNE_DIO24ARD.h>
SCIDYNE_DIO24ARD DIO24ARD;

void setup()
{
 /*************************************************************
  *  Setup the DIO24-ARD board and library software
  *  - Arduino digital I/O #10 used as DIO24-ARD select on SPI bus
  *  - The portA channels configured as inputs, portB and portC configured as outputs
  */
 DIO24ARD.setup(10, 0x00, 0xFF, 0xFF);  // pin#10, portA_DDR, portB_DDR, portC_DDR

 Serial.begin(115200);   // Initialize serial communications for Serial Monitor
}

void loop()
{
 uint8_t counter = 0x00;                          // An 8-bit counter initialized to 0x00
 uint8_t portA_data, portB_data, portC_data;      // Will hold the port data after a read

 while(1)  {
  // Continuously toggle the bits of portB by echoing the 8-bit counter value.
  DIO24ARD.portWrite(1, counter);          // Write portB with counter value
  counter++;                               // Increment the 8-bit counter

  // Read the three ports
  portA_data = DIO24ARD.portRead(0);   // Read and store portA data
  portB_data = DIO24ARD.portRead(1);   // Read and store portB data
  portC_data = DIO24ARD.portRead(2);   // Read and store portC data

  // Send the port data to the serial monitor.  Note: Open the Serial Monitor in the Arduino IDE to view.
  Serial.print(F("PortA: "));       Serial.println(portA_data,HEX);         // Display portA data
  Serial.print(F("PortB: "));       Serial.println(portB_data,HEX);         // Display portB data
  Serial.print(F("PortC: "));       Serial.println(portC_data,HEX);         // Display portC data

  Serial.println();
 }
}
```

Specifications

Number of Channels: 24 individually programmable digital Input/Output channels, non-isolated.

Input Level:
Logic 0 = 0.8vdc maximum, -0.3vdc minimum
Logic 1 = 2.0vdc minimum, 5.6vdc maximum

Output Level:
Logic 0 = < 1.0vdc (15ma load)
Logic 1 = > 2.5vdc (500µa load)

Max Output Current: Per channel: Source: 1.0ma, passively by 4.7k Pull-Up resistor to +5V
Sink: 85ma, actively by open-collector transistor to 0V (GND)

Software: Uses standard Arduino Serial-Peripheral-Interface (SPI) library functions.

Communications: Serial-Peripheral-Interface (SPI) through Arduino ICSP connector. Requires one user designated Arduino digital output for enable jumper selectable. EXT input for optional external enable signal.
SPI Configuration: Mode: 0 (i.e.; SPI_MODE0) SCK: 8 MHz maximum (i.e.; SPI_CLOCK_DIV2)

Field Connections: 50 Position IDC Ribbon Cable. Pin out compatible with Industry Standard 8, 16, and 24 position I/O Module Racks.

Arduino Connections: Stack-through connectors allows multiple shields.
Power: 8 Pos. x 1 Row
Analog: 6 Pos x 1 Row
Digital: 8 Pos x 1 Row & 10 Pos. x 1 Row
ICSP: 3 Pos x 2 Row

Power Requirement: +5vdc ±5% @ 45ma typical, all ports written as 0xff, external loads excluded

Dimensions: 2.80"W x 2.97"L x 1.04"H overall. Modified Arduino R3 format

Environmental: Operating temperature: -40˚C to 85˚C
Non-condensing relative humidity: 5% to 95%

Compliance: RoHS / Lead-Free Compliant

Product Origin: Designed, engineered, and assembled in U.S.A. by SCIDYNE Corporation using domestic and foreign components.

Appendix A: Schematic Diagram