USER'S REFERENCE MANUAL

MULTI-I/O-ARD

Analog and Digital I/O
Shield for Arduino and Compatibles

Model No.	100-7714	
Doc. No.	M7714	Rev: 1.01 12/11/23

SCIDYNE®

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

www.scidyne.com

Previous revision: None

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 Duration and conditions of warranty may be superseded when the product is integrated into other SCIDYNE products.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

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 (Set) and FALSE (Clear). Names with an asterisk (*) postscript or overlined are inverted or active low. Unless otherwise noted TRUE is considered logic '1' (Positive Voltage, +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 MULTI-IO-ARD is an Arduino peripheral board designed to satisfy common analog and digital input/output requirements in a broad range of embedded applications. The hardware has been engineered to operate at either 3.3V or 5V making it compatible with common microcontroller boards such as Arduino Uno, Mega, Due, Giga, Adafruit Metro, SparkFun Red Board, and numerous others. In many instances, the MULTI-IO-ARD will be the only peripheral board needed.

The purpose of this manual is to provide basic insight of the hardware and fundamental software concepts needed to apply the MULTI-IO-ARD. The reader is encouraged to refer to the actual chip manufacturers data sheets for detailed understanding of the features and capabilities of the devices used.

Key Features

- Standard Arduino R3 hardware footprint
- Support libraries from Adafruit, SparkFun, and others
- Selectable 3.3V or 5V operating voltage
- Low Power requirement
- I²C interface, Up to 1.7Mbps
- 16 Digital Input / Output channels
  - MCP23017 Chip
  - All channels are bi-directional
  - Programable pull-up resistors
  - Change-of-State and pattern matching interrupt capability
- Eight Analog Inputs
  - Two ADS1115 Chips
  - 16-Bit Resolution
  - Input ranges from ±256 mV to ±6.144 Volts
  - Single-Ended and Differential modes
  - Comparator detects under/over voltage measurements
- Eight Analog Outputs
  - Two MCP4728 Chips
  - 12-Bit Resolution
  - EEPROM retains last output voltage at Power-Up
- Optional Stemma QT / Qwiic and nodeLynk connectors for driving external hardware

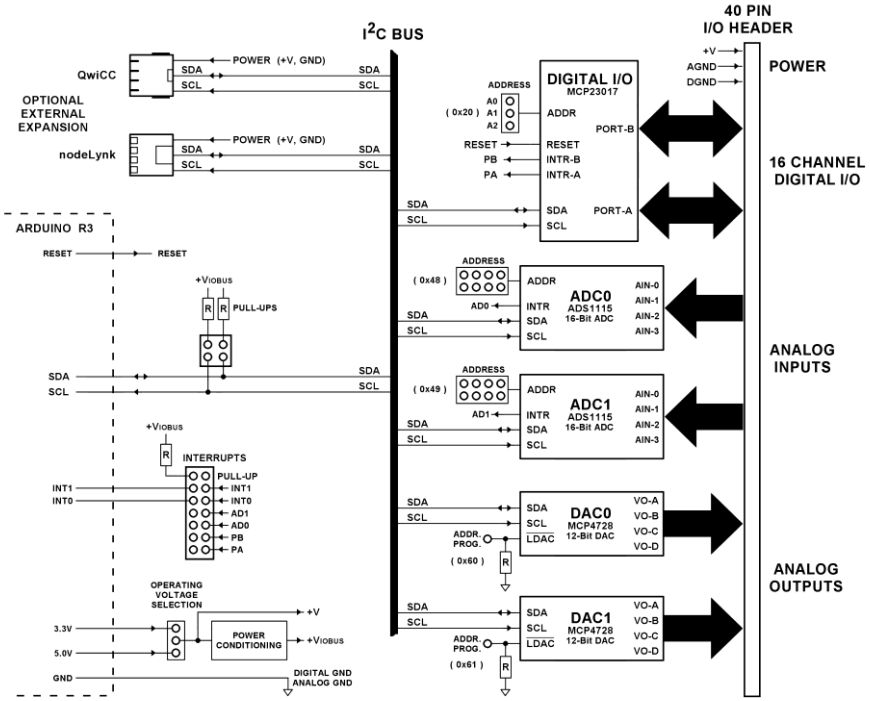


Figure 1 – MULTI-IO-ARD Simplified Block Diagram

Component Identification

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

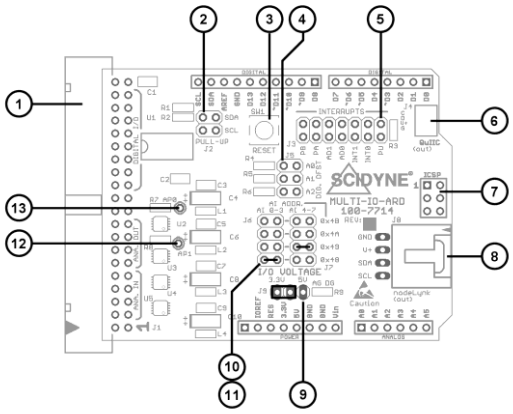


Figure 2 – MULTI-IO-ARD Component Identification

1 **I/O connector (J1)**
This 40-pin IDC header is used to connect the MULTI-IO-ARD to external devices. Please refer to Appendix-A for wiring information.

2 **I²C Pull-Ups (J2)**
These jumpers enable optional Pull-Up resistors on the SDA and SCL signals. Only one set of Pull-Up resistors should be used on any I²C bus segment and generally at the furthest point. The Pull-Ups are referenced to V+ as determined by J9

3 **RESET Push Button**
Momentarily pressing this button will reset the entire Arduino system.

4 **Digital I/O Address (J5)**
This jumper block determines the address of the Digital I/O chip on the I²C bus.

5 **Interrupt configuration jumpers (J3)**
This jumper block sets which interrupt sources will be used by the MULTI-IO-ARD to request interrupt service from the host.

6 **Optional Qwiic Connector (J4)**
This connector allows standard Stemma QT / Qwiic[1] devices to be connected through the MUTI-IO-ARD board.

7 **Optional ICSP Stackthrough connector**
An optional 2x3 header can be installed to pass-through ICSP signals from a lower board to one attached above. Only the connectors ground signal (Pin#6) is connected to the ground signal of MULTI_IO-ARD circuitry.

8 **Optional nodeLynk Connector (J8)**
This connector allows external device supporting the nodeLynk[1] interface to be driven from the MULTI-IO-ARD.

9 **I/O Voltage Selection (J9)**
The operating voltage for the MULTI-IO-ARD is set by this jumper. Default position is 3.3V.
Note: The voltage MUST match the operating voltage of the device the MULTI-IO-ARD is plugged in to. Failure to do so could damage either or both boards.

10,11 **Analog Input Address (J6, J7)**
These two jumper blocks determine the address of the Analog Input (ADC) chips. Jumper J6 Sets the address for AICH-0 to AICH-3; Jumper J7 Sets the address for AICH-4 to AICH-7.

12,13 **DAC Address Programming (AP0, AP1)**
The I²C addresses of the two DAC chips are changed using these hardware points and special programming software. AP0 is used to program DAC0, AP1 is used to program DAC1.

1) Stemma QT / Qwicc and nodeLynk are peripheral hardware buses based on I²C communications. Popularized by Adafruit, SparkFun, National Control Devices and other third parties.

Operating Voltage

The MULTI-IO-ARD circuitry can operate at either 3.3V or 5V. This ability allows its use with a wide variety of microcontroller boards and compatible hardware including those based on FPGAs. The operating voltage determines the logic levels of the signals between the MULTI-IO-ARD and host as well as the logic levels of any external circuitry connected through connectors J1, J4, and J8.

Place jumper J9 to either the 3.3V or 5V position to set the $+V_{IOBUS}$ operating voltage.

The operating I/O voltage of the MULTIO-IO-ARD must be properly configured to match that of the host microcontroller board. A mismatch could permanently damage the MULTI-IO-ARD, the microcontroller board, or both devices.

If relocating the MULTI-IO-ARD to a different system this prerequisite must be verified and the J9 I/O Voltage jumper changed if necessary.

The source of the 3.3V and 5V operating power is derived from the host. The MULTI-IO-ARD does not possess any on-board power supply regulation or current limiting circuitry.

I²C Addresses

A host uses I^2C to communicate to the MULTI-IO-ARD circuitry. To operate correctly, each device on the I^2C bus must have a unique address. The five circuits and their default addresses used by MULTI-IO-ARD are summarized in the following table. Device addresses can be changed but devices cannot be disabled or removed from the I^2C bus. Subsequent sections of this document describe the devices and their addresses in more detail. Jumper block J2 allows optional 4.7K Pull-Up resistors to be enabled on the SDA and SCL signals.

I^2C Addresses used by the MULTI-IO-ARD				
Circuit Function	Device	Factory Default Address	Possible Addresses	Comment
Digital I/O	PA0 – PA7, PB0 – PB7 (DIO0, MCP23017, U1)	0x20	0x20 – 0x27	Change by jumper block J5
Analog Inputs	AICH0 – AICH3 (ADC0, ADS1115, U4)	0x48	0x48 – 0x4B	Change by jumper block J6
	AICH4 – AICH7 (ADC1, ADS1115, U5)	0x49		Change by jumper block J7
Analog Outputs	AOCH0 – AOCH3 (DAC0, MCP4728, U2)	0x60	0x60 – 0x67	Change by re-programming
	AOCH4 – AOCH7 (DAC1, MCP4728, U3)	0x61		Change by re-programming

Digital Input / Output

The MULTI-IO-ARD module uses an industry standard MCP23017 I/O Expander chip to provide 16 non-isolated digital Input/Output channels across two 8-bit ports. This device is very versatile and offers flexible configurations, including software programable channel directions and interrupt-driven change-of-state and pattern matching functions. Each channel features TTL/CMOS compatible signal levels and ±25mA drive capability. In addition, software programmable weak pull-up resistors are available on any of the channels. This feature makes sensing open-collector, switches, and contact-closure type devices simple and straight-forward. All channels default to inputs during system reset. The digital I/O circuitry cannot be disabled.

Digital I/O circuitry highlights:

- 16 I/O channels organized as two 8-bit ports
- Individually programmable channel direction
- Programmable weak Pull-Ups resistors
- Change of State and pattern matching interrupt capability
- Programmable read polarity

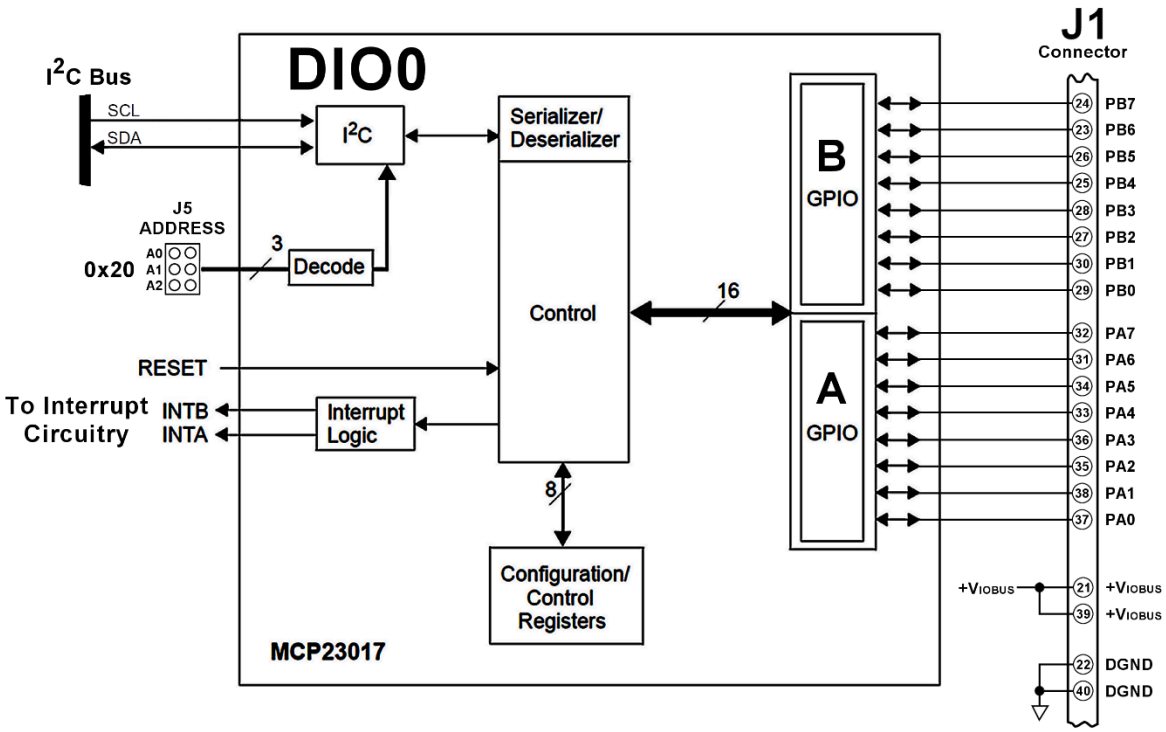


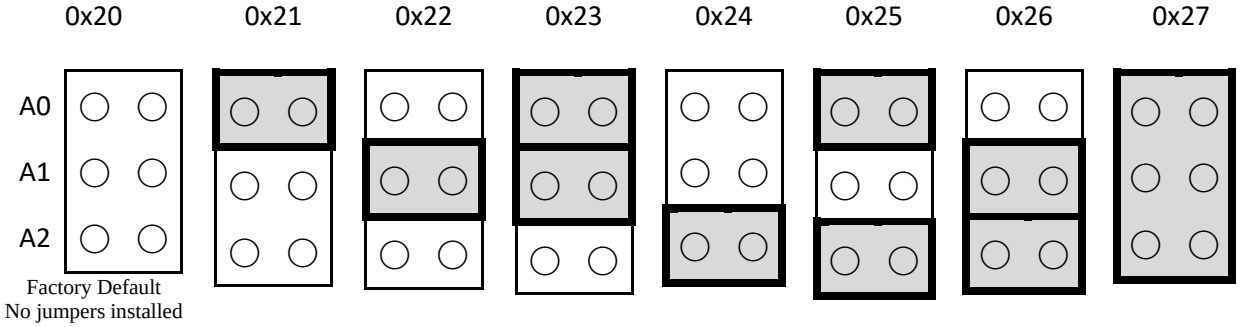
Figure 3 - Digital I/O Block Diagram

External components attach to the MULTI-IO-ARD through pins of connector J1. Please refer to Appendix-A to determine all signal locations. A companion terminal board (SCIDYNE PN 100-7625/40) is available to make field wiring easier.

Setting the Digital I/O I²C address

The default I²C address of the MCP23017 chip is 0x20. Jumper block J5 allows the address to be changed as shown below. Since the address is most often a one-time setup fixed wire jumpers are typically soldered in place. However, if the address will be changed frequently a 2x3 header can be installed and moveable shorting shunts used to conveniently modify the address as needed.

J5 Address Settings



To prevent conflicts, each device residing within the 0x20 – 0x27 address range must have a unique address on the I²C bus segment they share.

Digital I/O Software example

This example shows a basic use of the MULTIO-IO-ARD Digital I/O circuitry.

Please refer to the MCP23017 manufacturer's data sheet for complete hardware and software information related to this device.

```
/*************************
 * SCIDYNE Corporation, Oct 12, 2021,  Mark Durgin
 * Simple digital I/O demo program for MULTI-IO-ARD PN #100-7714
 * - Repeatedly writes pattern of alternating 0's and 1's to Port-A of the MCP23017 chip
 *   Use an oscilloscope to observe results
 * - Reads state of Port-B and prints to serial monitor. Short pins to ground to cause changes
 */
#include <Wire.h>   // Needed for Arduino I2C support

/*******************************
 * Program constants and defines
 */
#define MCP23017_ADDRESS   0x20      // Chip Address A0 = A1 = A3 = GND.

// Define the MCP23017 registers to be used. See MCP23017 data sheet for details
#define MCP23017_IODIRA      0x00     // MCP23017 Port-A Data Direction Register
#define MCP23017_GPIOA       0x12     // MCP23017 Port-A Data register
#define MCP23017_IODIRB      0x01     // MCP23017 Port-B Data Direction Register
#define MCP23017_GPIOB       0x13     // MCP23017 Port-B Data register
#define MCP23017_GPPUB       0x0D     // MCP23017 Port-B Pull-Up enable register

/*******************************
 * setup runs once then execution goes to loop
 */
void setup()
{
  Wire.begin();                                 // Setup I2C communications

  Serial.begin(9600);                           // Setup communications to IDE Monitor
    while (!Serial);

  // Setup MCP23017 Port-A as all outputs
  Wire.beginTransmission(MCP23017_ADDRESS);          // Open I2C communication with the MCP23017
  Wire.write(MCP23017_IODIRA);                       // Next byte intended for Port-A direction register
  Wire.write(0x00);                                  // Make all outputs, Bits set as 0 are outputs
  Wire.endTransmission();                            // Close I2C communications

 // Setup MCP23017 Port-B as all inputs
  Wire.beginTransmission(MCP23017_ADDRESS);          // Open I2C communication with the MCP23017
  Wire.write(MCP23017_IODIRB);                       // Next byte intended for Port-B direction register
  Wire.write(0xFF);                                  // Make all inputs, Bits set as 1 are inputs
  Wire.endTransmission();                            // Close I2C communications

  Wire.beginTransmission(MCP23017_ADDRESS);          // Open I2C communication with the MCP23017
  Wire.write(MCP23017_GPPUB);                        // Next byte intended for Port-B Pull-Up register
  Wire.write(0xFF);                                  // Bits set as 1 enable corresponding pull-up
  Wire.endTransmission();                            // Close I2C communications

}


/*******************************
 * loop runs continuously
 */
void loop()
{
  // Toggle bit pattern on Port-A
  Wire.beginTransmission(MCP23017_ADDRESS);          // Open I2C communication with the MCP23017
  Wire.write(MCP23017_GPIOA);                        // Next byte intended for Port-A data register
  Wire.write(0xAA);                                  // Make Port-A = 1 0 1 0 1 0 1 0
  Wire.endTransmission();                            // Close I2C communications

  delay(100);                                        // Wait 100ms

  Wire.beginTransmission(MCP23017_ADDRESS);          // Open I2C communication with the MCP23017
  Wire.write(MCP23017_GPIOA);                        // Next byte intended for Port-A data register
  Wire.write(0x55);                                  // Make Port-A = 0 1 0 1 0 1 0 1
  Wire.endTransmission();                            // Close I2C communications

  delay(100);                                        // Wait 100ms

  // Read the the state of Port-B and display on Arduino IDE Serial Monitor
  Wire.beginTransmission(MCP23017_ADDRESS);          // Open I2C communication with the MCP23017
  Wire.write(MCP23017_GPIOB);                        // Read will be from Port-B data register
  Wire.endTransmission();

  Wire.requestFrom(MCP23017_ADDRESS, 1);             // Request 1 byte from the MCP23017
  unsigned char x = Wire.read();                     // Receive the Port-B byte and store in x
  Serial.println(x);                                 // Print the character

  delay(100);                                        // Wait 100ms

}
```

Analog Inputs

Two ADS1115 Analog-to-Digital Converter (ADC) chips serve to implement the eight analog inputs. The analog input signals are routed to connections on the 40-position IDC header, J1. Please refer to Appendix-A to determine all signal locations.

All analog input signals are non-isolated and share the same ground potential as the digital circuitry and host. However, the physical ground signals associated with the analog circuitry have been meticulously routed on the MUTI-IO-ARD. Ideally, when attaching external analog input circuitry, keep Analog Ground (AGND) and Digital Ground (DGND) separated to obtain the best measurement performance.

Analog Input circuitry highlights:

- Eight Analog Inputs
- 16-Bit resolution
- Input ranges from ±256 mV to ±6.144 V
- Single-Ended and Differential modes
- Programmable Data Rate: 8 SPS to 860 SPS
- Fast Single-Cycle Settling
- Digital Comparator for Over/Under voltage measurements
- Internal Low-Drift voltage reference

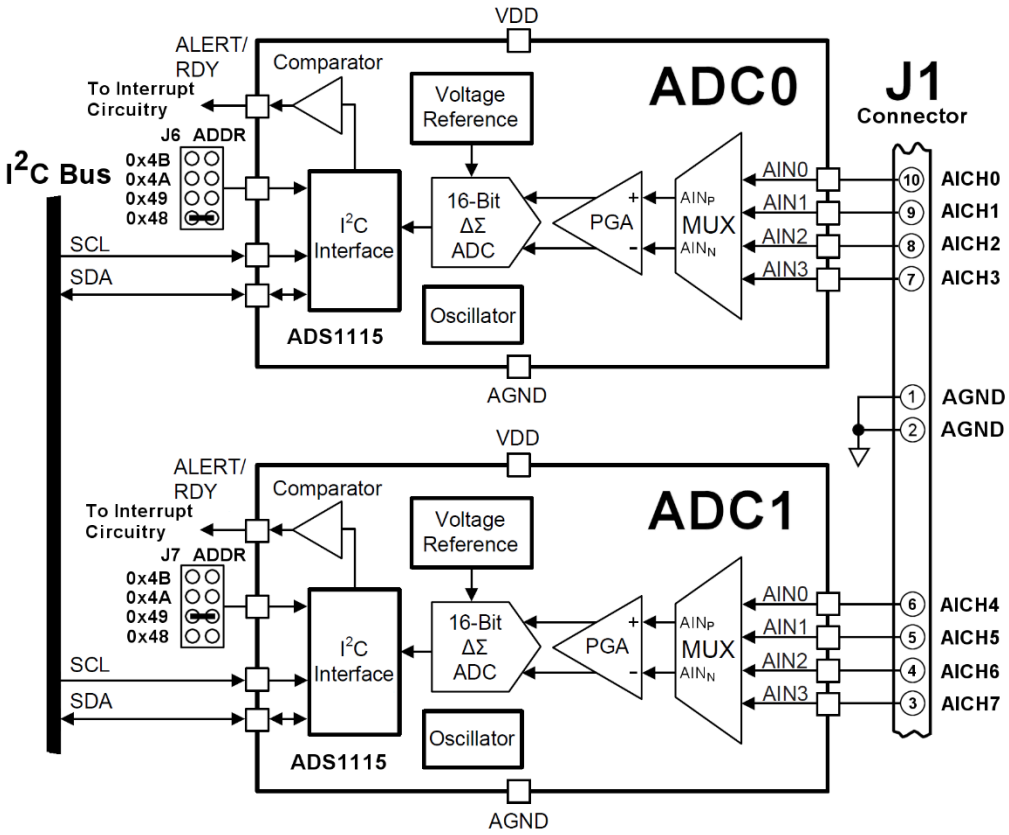


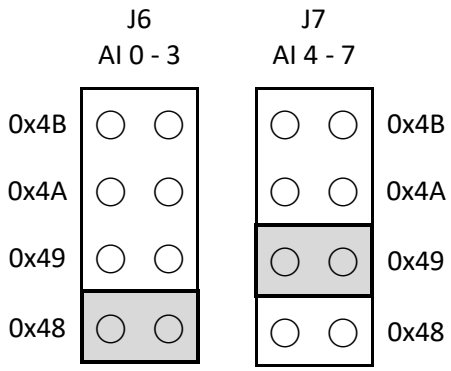
Figure 4 – Analog Inputs Block Diagram

Setting the Analog Input I²C addresses

Each ADC chip must have a unique address on the I²C bus. The default address of the first ADC chip (ADC0), serving Analog Input channels 0 to 3, is 0x48 and is set by jumper block J6. The default address for the second ADC chip (ADC1), serving Analog Input channels 4 to 7, is 0x49 and is set by jumper block J7. The Analog Input circuitry cannot be disabled.

Since the addresses are most often a one-time setup fixed wire jumpers are soldered in place. However, if the addresses will be changed frequently 2x4 headers can be installed and moveable shorting shunts used to conveniently set the addresses as needed.

Analog Input I²C address settings
Default positions shown



To prevent conflicts, each device residing within the 0x48 – 0x4B address range must have a unique address on the I²C bus segment they share.

Please refer to the ADS1115 manufacturer's data sheet for complete hardware and software information related to this device.

Data Format

The ADS1115 provide 16-bits of data in binary two's complement format. A positive Full-Scale (+FS) input produces an output code of 7FFFh and a negative Full-Scale (–FS) input produces an output code of 8000h. The output clips at these codes for signals that exceed full-scale. The following table summarizes the ideal output codes for different input signals.

Input Signal Versus Ideal Output Code	
Input Signal **VIN = (V_{AINP} – V_{AINN})**	Ideal Output Code (Excludes the effects of noise, INL, offset, and gain errors)
≥ +FS (2^{15} – 1) / 2^{15}	0x7FFFh
+FS/2^{15}	0x0001h
0	0x0000h
–FS/2^{15}	0xFFFFh
≤ –FS	0x8000h

Single-ended signal measurements, where V_{AinN} = 0 V and V_{AinP} = 0 V to +FS, only use the positive code range from 0000h to 7FFFh. However, because of device offset and other effects, the ADS1115 can still output negative codes when an analog input is close to 0 V.

Single-Ended and Differential Measurements

Prior to each conversion the ADS1115 is instructed to perform either a Single-Ended or Differential measurement. This is done using the Multiplexer bits within the configuration register of ADC0 or ADC1. Details can be found in the ADS1115 datasheet.

The table below summarizes the relationship between the two ADS1115 chips, measurement type, and their physical locations on the J1 connector.

Input multiplexer configuration									
Measurement Type	MUX[2:0]	ADC0				ADC1			
		AICH0 J1-10	AICH1 J1-9	AICH2 J1-8	AICH3 J1-7	AICH4 J1-6	AICH5 J1-5	AICH6 J1-4	AICH7 J1-3
Differential	0 0 0	AIN_P	AIN_N			AIN_P	AIN_N		
	0 0 1	AIN_P			AIN_N	AIN_P			AIN_N
	0 1 0		AIN_N		AIN_N		AIN_N		AIN_N
	0 1 1			AIN_P	AIN_N			AIN_P	AIN_N
Single-Ended(2)	1 0 0	AIN_P				AIN_P			
	1 0 1		AIN_P				AIN_P		
	1 1 0			AIN_P				AIN_P	
	1 1 1				AIN_P				AIN_P

1. AIN_P is the Positive voltage input and AIN_N is the Negative voltage input to the Programmable Gain Amplifier.
2. When configured for Singled-Ended measurements, AIN_N of the Programmable Gain Amplifier is connected to Analog Ground by the MULTI-IO-ARD hardware.

Analog Input Software Example

```
/* ************************
 * SCIDYNE Corporation, Oct 14 2021  Mark Durgin
 * Analog Input demo program for MULTI-IO-ARD PN #100-7714
 *
 * Display Single-Ended conversions of Analog Inputs Channels 0 - 7
 * Results displayed on Arduino IDE Serial Monitor
 * See ADS1115 datasheet for details of setup and operation
 */

/* ******************************
 * Important include files
 */
#include <Wire.h>   // Needed for Arduino I2C support

/* ******************************
 * Prototypes
 */
uint16_t readADC (unsigned char channel);   // Digitize an analog input channel, return 16-bit result

/* ******************************
 * Program constants and defines
 */
#define ADS1115_0_ADDRESS   (0x48)          // I2C Address of first ADS1115, AICH0 - AICH3
#define ADS1115_1_ADDRESS   (0x49)          // I2C Address of second ADS1115, AICH4 - AICH7

#define ADS1X15_REG_POINTER_CONVERT (0x00)  // ADS1115 internal Conversion Register
#define ADS1X15_REG_POINTER_CONFIG (0x01)   // ADS1115 internal Configuration Register

/* ******************************
 * setup runs once then execution goes to loop
 */
void setup()
{
  Wire.begin();                     // Setup I2C communications

  Serial.begin(9600);               // Setup communications to IDE Monitor
    while (!Serial);
}


/* ******************************
 * loop runs continuously
 * Digitaize each Analog Input channel and display results on
 * Arduino IDE Serial Monitor.
 */
void loop()
{
   uint16_t i;       // loop and channel variable
   uint16_t result;  // Holds 16-bit ADC conversion result

   for (i = 0; i < 8; i++)
   {
      result = readADC(i);
      Serial.print("Channel-");
      Serial.print(i);
      Serial.print(": ");
      Serial.println(result,HEX);

   }

   Serial.println("----------------");
   delay(1000);
}
```

```
/* ************************
 * Read an ADC Channel
 * Call with desired channel, Returns signed 16-bit result
 */
uint16_t readADC (unsigned char AIchannel)
{
  uint16_t var = 0x0000;          // 16-bit temp stoarge variable
  uint16_t config_var;            // Used to build configuration register bits
  uint8_t ads1115_address;        // Holds I2C address of ADS115 being accessed

  // Assure AIchannel is within valid range of 0-7
  AIchannel &= 0x07;

  // Determine which ADS1115 chip will be used
  ads1115_address = (AIchannel > 3) ? ADS1115_1_ADDRESS : ADS1115_0_ADDRESS;

  // Build the 16-bit ADS1115 Configuration Register, see ADS1115 data sheet for details
  config_var =
     (0x0003)   |       // Bit 0..1    Disable the comparator (default val)
     (0x0000)   |       // Bit 2       Non-latching (default val)
     (0x0000)   |       // Bit 3       Alert/Rdy active low   (default val)
     (0x0000)   |       // Bit 4       Traditional comparator (default val)
     (0x0080)   |       // Bit 5..7    Samples-per-second rate, 128 samples-per-second (default)
     (0x0100)   |       // Bit 8       Single-shot mode (default)
     (0x0400)   |       // Bit 9..11   Set input voltage range, default is +/- 2.048 Volts
     /* Set below */     // Bit 12..13  Mux Channel
     (0x4000)           // Bit-14      Single-Ended / Differential Mode
     /* Set below */  ;  // Bit 15      Write: Single-Shot start, Read: Conversion status

  // Setup for conversion on desired input channel where bits 12 and 13 set the mux channel 0-3
  config_var |= (AIchannel % 4) << 12;

  // Set bit 15 which 'start single-conversion' when written
  config_var |= 0x8000;

  // Write config register to the ADC which starts the conversion
  Wire.beginTransmission(ads1115_address);      // Open I2C communication with the ADS1115
  Wire.write(ADS1X15_REG_POINTER_CONFIG);       // Next 2 bytes intended for config register
  Wire.write(config_var >> 8);                  // Write config register High-Byte
  Wire.write(config_var & 0xFF);                // Write Config register Low-Byte
  Wire.endTransmission();                       // Close I2C communications

  // Wait for the conversion to complete
  do {
    Wire.requestFrom(ads1115_address, 2);       // Request 2 bytes (16-Bits) from the ADS1115
    var = Wire.read() << 8;                     // Read the configuration register High-Byte
    var |= Wire.read();                         // Read the configuration register Low-Byte
  } while (!(var & 0x8000));                    // Loop until bit 15 is "0", i.e.; conversion complete

  // Read the signed 16-bit conversion results
  // Concatenate the two bytes to form the 16-bit result
  Wire.beginTransmission(ads1115_address);    // Open I2C communication with the ADS1115
  Wire.write(ADS1X15_REG_POINTER_CONVERT);    // Next 2 bytes read will be from
conversion register
  Wire.endTransmission();                     // Close I2C communications

  Wire.requestFrom(ads1115_address, 2);       // Request 2 bytes (16-Bits) from the ADS1115
  var = Wire.read() << 8;                     // First byte is High-Byte, bits [15:8] 8..15
  var |= Wire.read();                         // Second byte is Low-Byte, bits [7:0]  0..7

  return(var);                          // Return the result

}
```

Analog Outputs

The Analog Outputs are produced by two MCP4728 Quad Digital-to-Analog (DAC) chips.

Analog Output circuitry highlights:

- Eight Analog Outputs
- 12-Bit resolution
- Software Selectable Voltage Reference
  - System VDD
  - Internal Low-Drift reference
- EEPROM retains last output voltage at Power-On
- Software programable power saving modes

Please refer to the MCP4728 manufacturer's data sheet for complete hardware and software information related to this device.

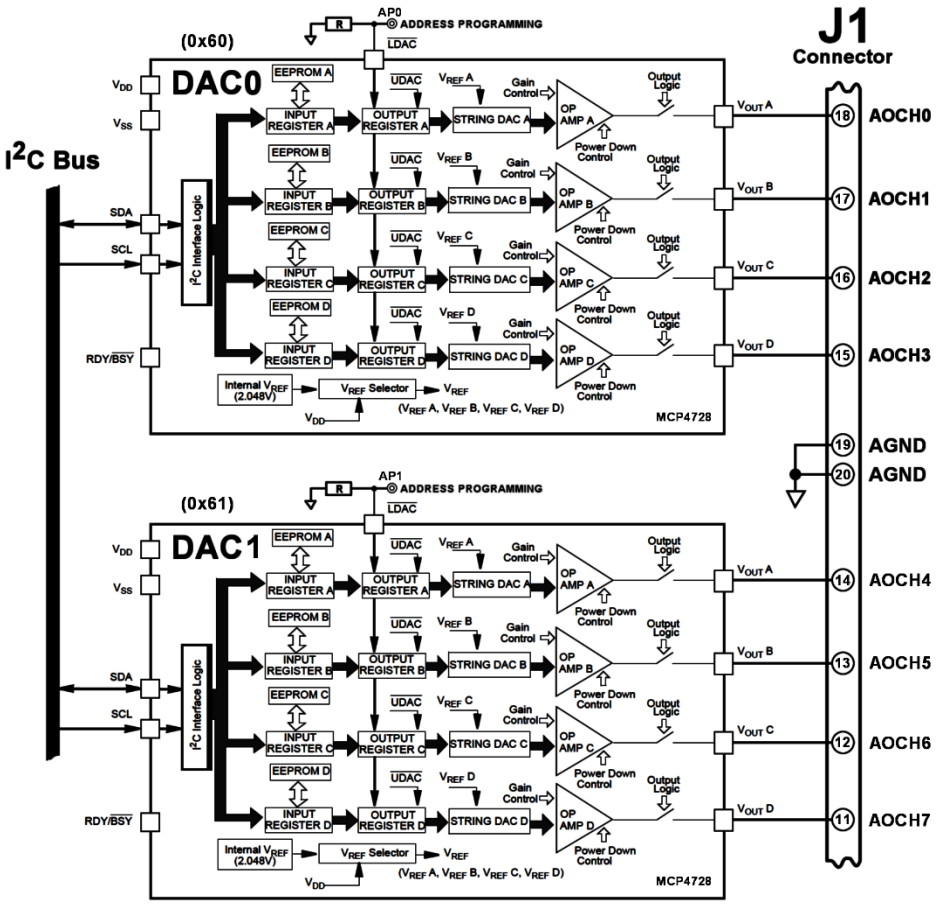


Figure 5 - Analog Outputs Block Diagram

Setting the Analog Output I²C addresses

The default I²C address of DAC0 is 0x60 and the default address of DAC1 is 0x61. The addresses are stored within the respective DAC chip and can only be changed by reprogramming their internal EEPROM. The Analog Output circuitry can not be disabled.

To prevent conflicts, each device residing within the 0x60 – 0x67 address range must have a unique address on the I²C bus segment they share.

Analog Output Software Example

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * SCIDYNE Multi-IO-ARD  10-26-21  Mark Durgin
 * Basic demo for using the eight Analog Outputs
 * Utilizing two MCP4728 4-Channel 12-bit I2C DACs
 *
 * For compatibility with 3.3V and 5V hardware the program sets
 * VREF to use internal 2.048 reference and a Gain of 1.
 * Therefore, any DAC Channel Vout = 2.048V * (VALUE / 4095)
 * where VALUE is the 12-bit number written to the input register
 * of a DAC channels.
 *
 * See MCP4728 datasheet for details
 */

/* * * * * * * * * * * * * * * * *
 * Important include files
 */
#include <Wire.h>

/* * * * * * * * * * * * * * * * *
 * Setup Runs only once. Execution then
 * goes to loop
 */
void setup(void) {

  // DAC0 definitions
 #define MCP4728_DAC0_ADDRESS 0x60
 #define AO_0_VALUE  0      // DAC0 Channel A, Analog Output#0 Produces 0.000 Volts
 #define AO_1_VALUE  585    // DAC0 Channel B, Analog Output#1 Produces 0.293 Volts
 #define AO_2_VALUE  1170   // DAC0 Channel C, Analog Output#2 Produces 0.585 Volts
 #define AO_3_VALUE  1755   // DAC0 Channel D, Analog Output#3 Produces 0.878 Volts

 // DAC1 definitions
 #define MCP4728_DAC1_ADDRESS 0x61
 #define AO_4_VALUE  2340   // DAC1 Channel A, Analog Output#4 Produces 1.170 Volts
 #define AO_5_VALUE  2925   // DAC1 Channel B, Analog Output#5 Produces 1.463 Volts
 #define AO_6_VALUE  3510   // DAC1 Channel C, Analog Output#6 Produces 1.755 Volts
 #define AO_7_VALUE  4095   // DAC1 Channel D, Analog Output#7 Produces 2.048 Volts

 // Local temporary variables created on stack
 uint8_t output_buffer[16];   // Holds DAC bytes to be written via I2C

 // Initialize I2C support
 Wire.begin();

 // Preload DAC0 values into buffer
 output_buffer[0] = AO_0_VALUE >> 8;        // DAC0 Channel A HB
 output_buffer[1] = AO_0_VALUE & 0xFF;    // DAC0 Channel A LB

 output_buffer[2] = AO_1_VALUE >> 8;        // DAC0 Channel B HB
 output_buffer[3] = AO_1_VALUE & 0xFF;    // DAC0 Channel B LB

 output_buffer[4] = AO_2_VALUE >> 8;        // DAC0 Channel C HB
 output_buffer[5] = AO_2_VALUE & 0xFF;    // DAC0 Channel C LB

 output_buffer[6] = AO_3_VALUE >> 8;        // DAC0 Channel D HB
 output_buffer[7] = AO_3_VALUE & 0xFF;    // DAC0 Channel D LB

 // Preload DAC1 values into buffer
 output_buffer[8] = AO_4_VALUE >> 8;        // DAC1 Channel A HB
 output_buffer[9] = AO_4_VALUE & 0xFF;    // DAC1 Channel A LB

 output_buffer[10] = AO_5_VALUE >> 8;       // DAC1 Channel B HB
 output_buffer[11] = AO_5_VALUE & 0xFF;   // DAC1 Channel B LB

 output_buffer[12] = AO_6_VALUE >> 8;       // DAC1 Channel C HB
 output_buffer[13] = AO_6_VALUE & 0xFF;   // DAC1 Channel C LB

 output_buffer[14] = AO_7_VALUE >> 8;       // DAC1 Channel D HB
 output_buffer[15] = AO_7_VALUE & 0xFF;   // DAC1 Channel D LB

  /* * * * * * * * * * * * * * * * * * * * * * * *
   * Set VREF of each DAC channel. In this example,
   * internal VREF (VREF = 2.048) is selected for all DAC channels
   */
  // Perform on DAC0 ...
  Wire.beginTransmission(MCP4728_DAC0_ADDRESS);  // Open I2C communication with the
MCP4728 DAC0
  Wire.write(0x8F);                    // C2 = 1, C1,C0 = 0,  VrefA - VrefD = 1 (Internal Vref)
  Wire.endTransmission();              // Close I2C communications
  // ... and also on DAC1
  Wire.beginTransmission(MCP4728_DAC1_ADDRESS);  // Open I2C communication with the
MCP4728 DAC1
  Wire.write(0x8F);                    // C2=1, C1,C0 = 0,  VrefA - VrefD = 1 (Internal Vref)
  Wire.endTransmission();              // Close I2C communications

  /* * * * * * * * * * * * * * * * * * * * *
   * Set Gain of each DAC channel. In this example,
   * Gain of 1 is selected for all DAC channels.
   */
  // Perform on DAC0 ...
  Wire.beginTransmission(MCP4728_DAC0_ADDRESS);  // Open I2C communication with the
MCP4728 DAC0
  Wire.write(0xC0);                    // C2,C1 = 1, C0 = 0,  GainA - GainD = 0 (Gain= 1X)
  Wire.endTransmission();              // Close I2C communications
  // ... and also on DAC1
  Wire.beginTransmission(MCP4728_DAC1_ADDRESS);  // Open I2C communication with the
MCP4728 DAC1
  Wire.write(0xC0);                    // C2,C1 = 1, C0 = 0,  GainA - GainD = 0 (Gain= 1X)
  Wire.endTransmission();              // Close I2C communications


  /* * * * * * * * * * *
   * Perform Fast Write to all DAC channels
   */
  // Perform on DAC0 Analog Outputs 0,1,2,3 ...
  Wire.beginTransmission(MCP4728_DAC0_ADDRESS);  // Open I2C communication with the
MCP4728 DAC0
  // Analog Output #0
  Wire.write(output_buffer[0]);        // DAC0 Channel A Value HB, Analog Output#0
  Wire.write(output_buffer[1]);        // DAC0 Channel A Value LB, Analog Output#0
  // Analog Output #1
  Wire.write(output_buffer[2]);        // DAC0 Channel B Value HB, Analog Output#1
  Wire.write(output_buffer[3]);        // DAC0 Channel B Value LB, Analog Output#1
  // Analog Output #2
  Wire.write(output_buffer[4]);        // DAC0 Channel C Value HB, Analog Output#2
  Wire.write(output_buffer[5]);        // DAC0 Channel C Value LB, Analog Output#2
  // Analog Output #3
  Wire.write(output_buffer[6]);        // DAC0 Channel D Value HB, Analog Output#3
  Wire.write(output_buffer[7]);        // DAC0 Channel D Value LB, Analog Output#3
  Wire.endTransmission();              // Close I2C communications with the MCP4728 DAC0

  // ... and on DAC1 Analog Outputs 4,5,6,7 ...
  Wire.beginTransmission(MCP4728_DAC1_ADDRESS);  // Open I2C communication with the
MCP4728 DAC1
  // Analog Output #4
  Wire.write(output_buffer[8]);        // DAC0 Channel A Value HB, Analog Output#4
  Wire.write(output_buffer[9]);        // DAC0 Channel A Value LB, Analog Output#4
  // Analog Output #5
  Wire.write(output_buffer[10]);       // DAC0 Channel B Value HB, Analog Output#5
  Wire.write(output_buffer[11]);       // DAC0 Channel B Value LB, Analog Output#5
  // Analog Output #6
  Wire.write(output_buffer[12]);       // DAC0 Channel C Value HB, Analog Output#6
  Wire.write(output_buffer[13]);       // DAC0 Channel C Value LB, Analog Output#6
  // Analog Output #7
  Wire.write(output_buffer[14]);       // DAC0 Channel D Value HB, Analog Output#7
  Wire.write(output_buffer[15]);       // DAC0 Channel D Value LB, Analog Output#7
  Wire.endTransmission();              // Close I2C communications with the MCP4728 DAC1
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Execution comes here after setup runs only once.
 * loop continuously does nothing
 */
void loop() { delay(1000); }
```

Interrupts

The MULTI-IO-ARD can generate an interrupt to signal the completion of Analog-to-Digital conversion, an analog input has reached a pre-defined comparator threshold, when a digital input has changed state, or when a pre-defined pattern appears on the digital inputs.

When properly configured for "open-collector" type outputs multiple interrupts sources can be used together to drive the same host interrupt.

Host Interrupt Selection

Jumper block J3 configures which host interrupt will be associated with the board and which interrupt sources will be used.

Interrupt Configuration Jumpers, J3	
Jumper	Description
PB	Digital I/O circuitry Port B Interrupt output
PA	Digital I/O circuitry Port A Interrupt output
AD1	Analog-to-Digital ADC1 Interrupt output
AD0	Analog-to-Digital ADC0 Interrupt output
INT1(1)	Host Interrupt input #1, often shared with Arduino Digital I/O #3
INT0(1)	Host Interrupt input #2, often shared with Arduino Digital I/O #2
PU	Enable Pull-Up resistor, 4.7K to $+V_{IOBUS}$

1) Interrupt input may be shared with other system resources.

Try selecting an interrupt which is not currently being used by other system resources. If interrupts must be shared, make sure all the software applications and hardware involved support interrupt sharing. To prevent excessive current draw and the possibility of erroneous operation, use only one pull-up resistor.

Appendix - A J1, Input / Output Connections

Analog Ground	1▼	2	Analog Ground
AICH7	3	4	AICH6
AICH5	5	6	AICH4
AICH3	7	8	AICH2
AICH1	9	10	AICH0
AOCH7	11	12	AOCH6
AOCH5	13	14	AOCH4
AOCH3	15	16	AOCH2
AOCH1	17	18	AOCH0
Analog Ground	19	20	Analog Ground
+VIOBUS [1]	21	22	Digital Ground
(bit 14) PB6	23	24	PB7 (bit 15)
(bit 12) PB4	25	26	PB5 (bit 13)
(bit 10) PB2	27	28	PB3 (bit 11)
(bit 8) PB0	29	30	PB1 (bit 9)
(bit 6) PA6	31	32	PA7 (bit 7)
(bit 4) PA4	33	34	PA5 (bit 5)
(bit 2) PA2	35	36	PA3 (bit 3)
(bit 0) PA0	37	38	PA1 (bit 1)
+VIOBUS [1]	39	40	Digital Ground

Notes:

1) 3.3V or 5V depending on setting of jumper J9. Supplied by Host. Non-Isolated, Unfused.

Appendix - B Specifications

Specifications subject to change without notice

Analog Inputs:

General:	Two ADS1115 ADC chips providing up to eight Single-Ended or four Differential analog inputs
A/D resolution:	16-bit (1 in 65536, No Missing Codes)
Input ranges:	Software programmable: ± 6.144V, ± 4.096V, ± 2.048V, ± 1.024V, ± 0.512, ± 0.256V
Input impedance:	710K minimum
Nonlinearity:	±1LSB
Sampling Rate:	8, 16, 32, 64, 128, 250, 475, 860 SPS

Analog Outputs:

General:	Two MCP4728 DAC chips provide eight analog output channels
D/A resolution:	12-bit (1 in 4096 of full scale)
Output ranges:	Using Internal V_{REF} (2.048V): • 0.000V to 2.048V with Gain Setting = 1 • 0.000V to 4.096V with Gain Setting = 2 Note: 4.096V only available when $+V_{IOBUS}$ = 5.0V Using External V_{REF} (V_{DD}): • 0.000V to V_{DD}
Output current:	±25mA max. per output. Total may be limited by hosts inability to supply enough current.
Settling time:	6µs max. to within ±½LSB of final value
Nonlinearity:	Less than ±2LSB

Digital I/O:

General:	One MCP23017 chip provides 16 bi-directional I/O channels across two 8-Bit ports
Output current:	±25mA max. per output. Total may be limited by hosts inability to supply enough current.
Pull-Up Resistor:	100K, individually software enabled on each I/O channel

I^2C Interface:

Software:	Uses standard Arduino I^2C library functions. Fully supports third-party software libraries like those from Adafruit and SparkFun.
Addressing:	Digital I/O: Default: 0x20. Jumper 0x20 – 0x27 Analog Inputs: Default: ADC0 = 0x48, ADC1 = 0x49. Jumper 0x48 – 0x4B Analog Outputs: Default: DAC0 = 0x60, DAC1 = 0x61. Reprogrammable 0x60 – 0x67
Speed:	Standard (100kbps), Fast (400kbps), High Speed (1.7Mbps)
Pull-Ups:	Optional jumper enabled 4.7K Pull-Ups on SDA and SCL signals
Interrupt (Optional):	One Arduino interrupt, jumper selectable IRQ 1 or 2. Used by Analog-to-Digital converters and/or Digital I/O. Optional 4.7K Pull-Up Resistor

Connections:

I/O:	40 Position IDC Ribbon Cable
External Expansion:	4 Position Qwicc (optional), 4 Position nodeLynk (optional)
Arduino:	Stack-through connectors allow 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 (optional)
Power:	Jumper selectable +3.3V or 5.0V. Power derived from Arduino host.
Dimensions:	Standard Arduino UNO R3 footprint and dimensions. Approx.. 2.10” W x 3.00”L overall.

Environmental:

Operation:	-25°C to 65°C (Standard) Non-condensing relative humidity: 5% to 95%
Compliance:	RoHS, Lead-Free
Product Origin:	Designed, Engineered, and Assembled in U.S.A. by SCIDYNE® Corporation using domestic and foreign components.

User Notes