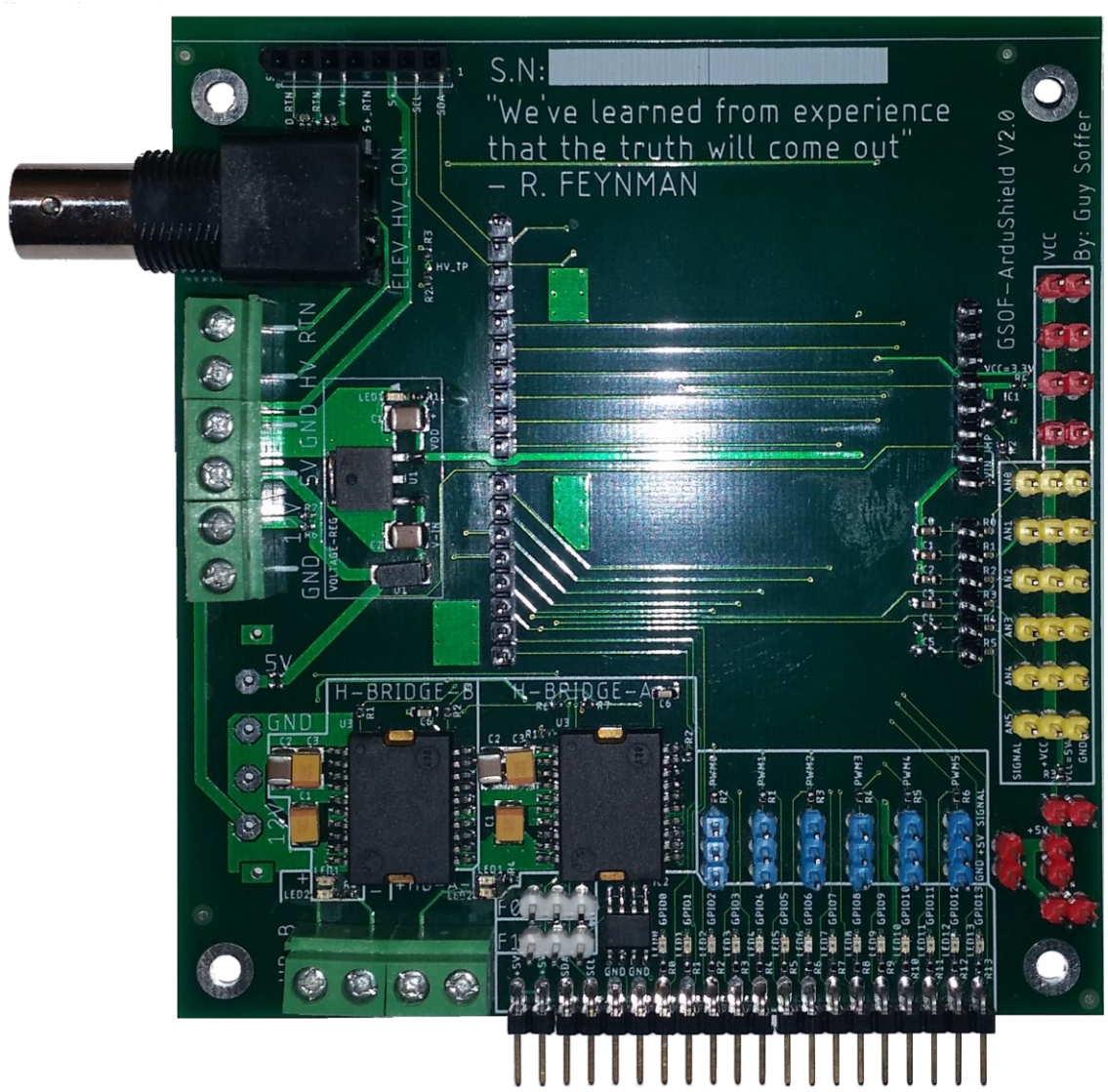
User manual for ArduBridge-Shield



Revision 0.2

Acknowledgements

I would like to thank the following beta testers for their valuable contributions to developing of the ArduBridge program:
James Perry, Laura Leclerc, Kenza Samlali, Tzur Soffer, and Sam Little.
Their feedback and suggestions were instrumental in identifying and resolving any issues in the program. I would especially like to acknowledge James Perry for using the Ardubridge in his academic publications and teaching new students how to use it.

List of publications

1. Real-Time Optogenetics System for Controlling Gene Expression Using a Model-Based Design.
2. Expanding the limits towards ‘one-pot’ DNA assembly and transformation on a rapid-prototype microfluidic device.
3. A fucosyltransferase inhibition assay using image analysis and digital microfluidics.
4. One Cell, One Drop, One Click: Hybrid Microfluidics for Mammalian Single Cell Isolation.

Contents

Table of figures

Introduction

The ArduBridge (GSOF_ArduBridge) is a software package designed to bridge between PC functionality (host/master computer) and attached embedded electronics (client/slave).
The ArduShield (GSOF_ArduShield) is an expansion board for the Arduino Uno R3 that adds additional peripherals and connectors to attach devices such as RC servos, DC motors, analog sensors, and high-power actuators.
The ArduBridge software sends Input/Output (IO) commands to the attached Arduino over a standard USB interface. Together with a Python user script, any PC can control virtually any electronic board. The PC runs the main algorithm (script), while the ArduBridge interfaces with real-world electronic devices. This setup is advantageous for developing real-time control loops, integrating software and hardware, and developing complex algorithms that are hard to build and debug without modern and interactive tools. However, the hardware can be used without the ArduBridge software by programming the Arduino directly in C/C++.

Top-level overview:

The chapter will briefly overview the hardware and software capabilities of the ArduShield and ShieldBridge.

Shield hardware

The ArduShield includes a dual H-bridge capable of driving:

- DC motors.
- Thermo Electric Coolers (TEC).
- Other high-power devices up to 12V / 2A.

In addition, the onboard 5V regulator eliminates the need to supply a precise voltage to drive the external peripherals and to prevent overloading the USB host.

The ArduShield also includes connectors for easy connection of servos, analog sensors, General Input Output port (GPIO), and devices over the Serial-Peripheral-Interface (SPI) or Inter-Integrated-Circuit I2C bus.

Although the ArduShield can be used in various projects, it was initially designed to encapsulate the electronic circuits used in Digital-Microfluidics (DMF) setup. A unique feature of the shield is the capability to stack up additional extension boards (e.g., the Digital-Microfluidics (DMF) electrode driver) without the need for extra cables and workbench space (Figure 7). Another unique capability is its ability to measure the impedance between any DMF electrode and the Indium-Tin-Oxide (ITO) plate. This is extremely useful for detecting faults in droplets and estimating droplet size. Figure 1 shows the top-level architecture of the ArduShield.
Each chapter will show the electrical circuit of each block and include an explanation regarding its electrical functionality and its respective Python classes.

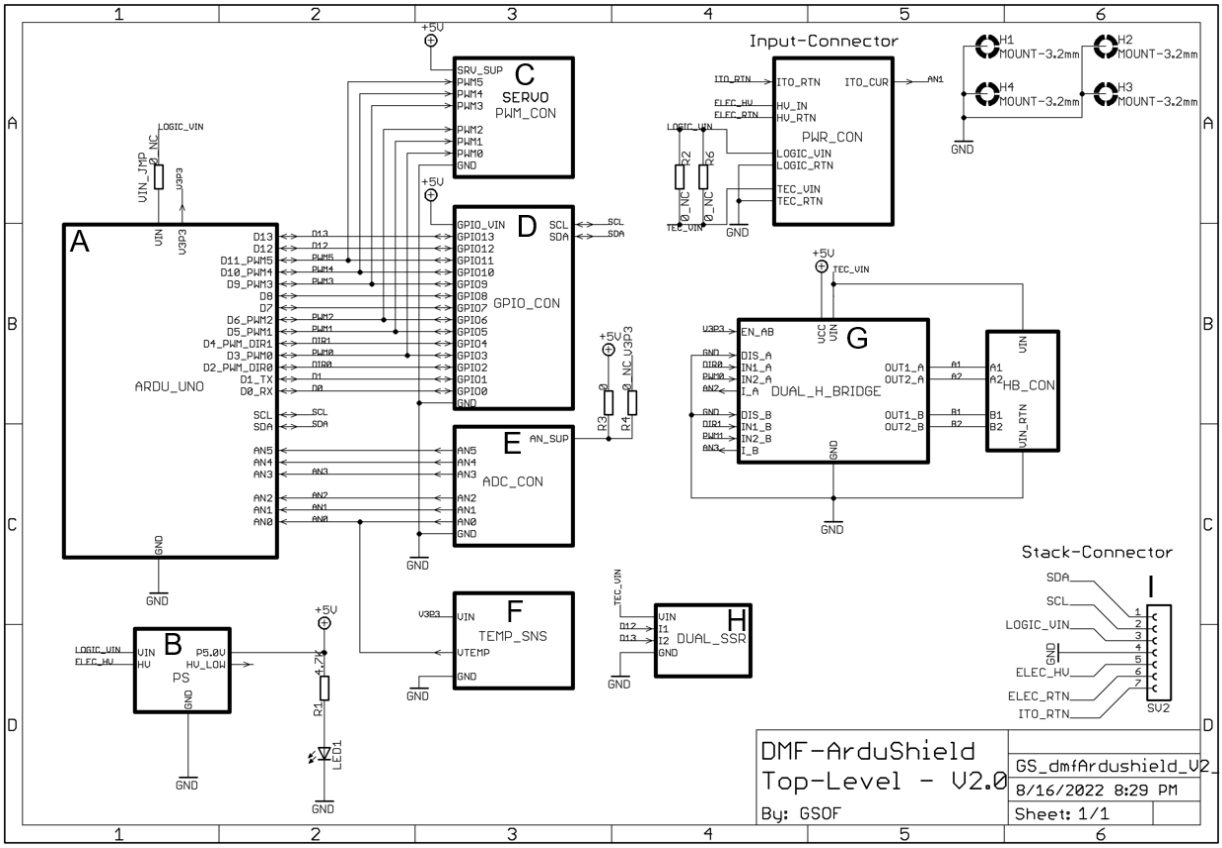


Figure 1: Top-level overview of the ArduShield electrical design

The ArduShield includes (**A**) socket for the Arduino Uno R3, (**B**) onboard 5V regulator to drive external peripherals, (**G**) dual H-bridge capable of driving 12V / 2A, (**H**) two Solid-State-Relays (SSR), (**F**) onboard TC1046 temperature sensor (optional). In addition, the board includes connectors with (**C**) standard RC servos pinout, (**D**) GPIO signals with SPI and I2C bus, (**E**) analog input with 5V supply, and (**I**) expansion connector with Inter-Integrated-Circuit I2C bus. Another unique capability of the shield is its ability to measure the impedance between any DMF electrode and the Indium-Tin-Oxide (ITO) plate (**B**).

Python classes

The ArduBridge software includes several classes. A dedicated Python class supports each hardware feature. The main script has two global objects to access the hardware: ***ardu*** and ***ards*** (**Figure 2**). Figure **3** describes the methods of each class next to its IO connector. The arduBridge_Shield class includes the special features of the shield. The arduBridge can run without the shield; in this case, the ***ards*** object can be removed or ignored.

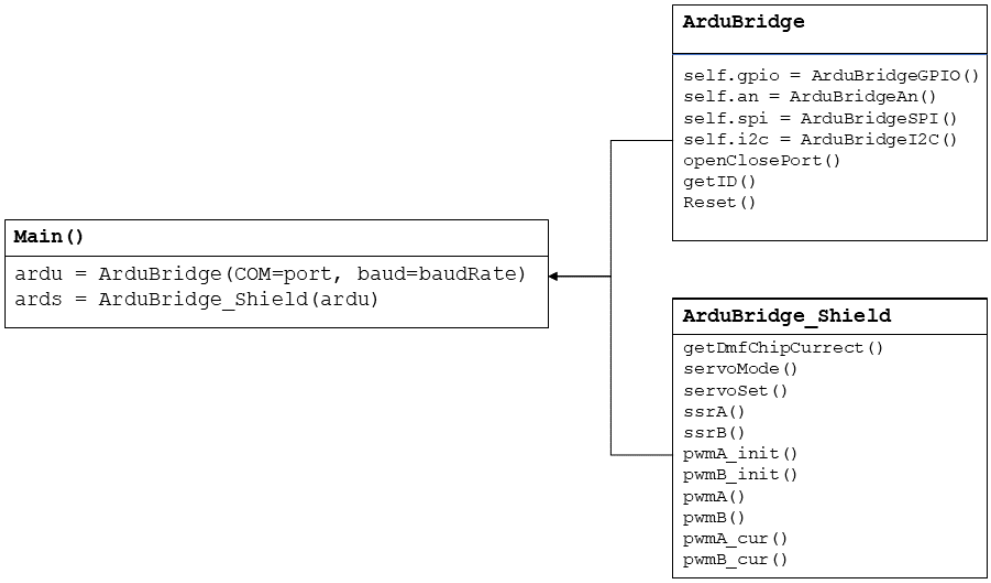


Figure 2 - Global ArduBridge system object

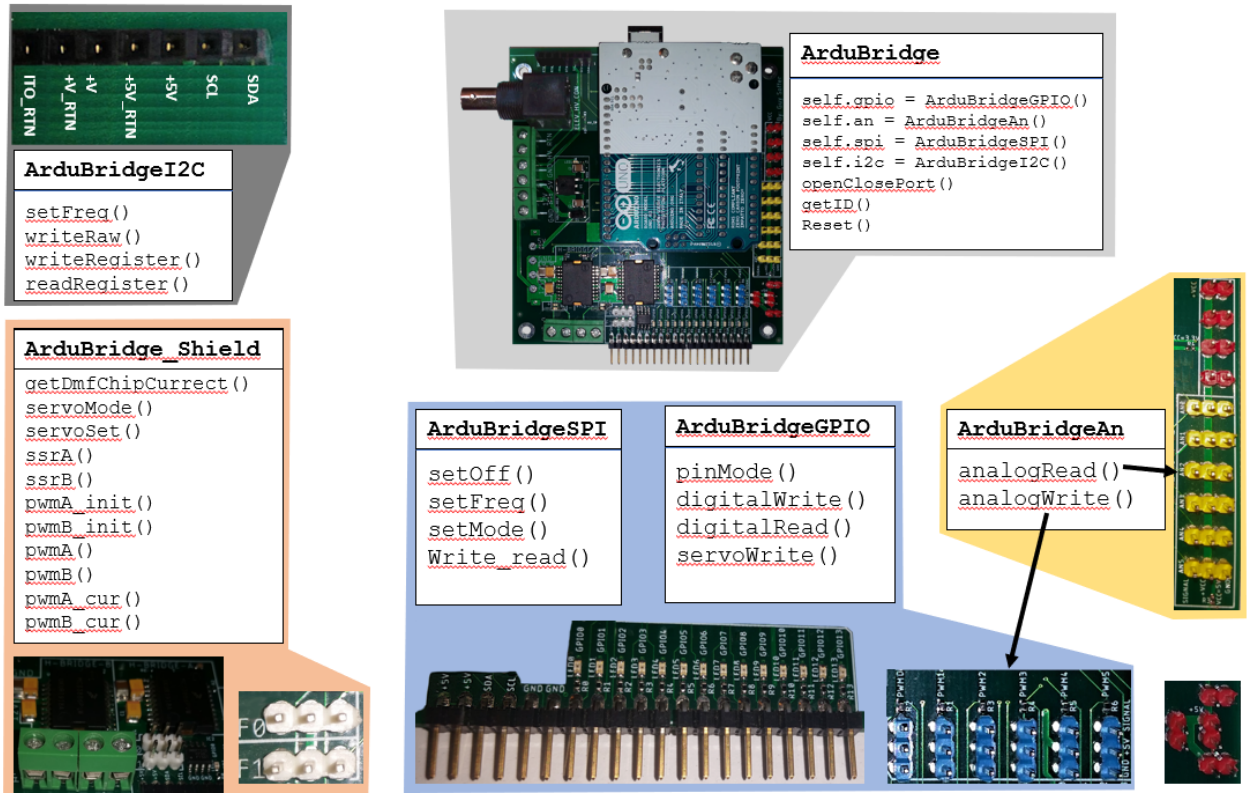


Figure 3 - The ArduBridge classes and their IO connectors

Installation and hardware setup

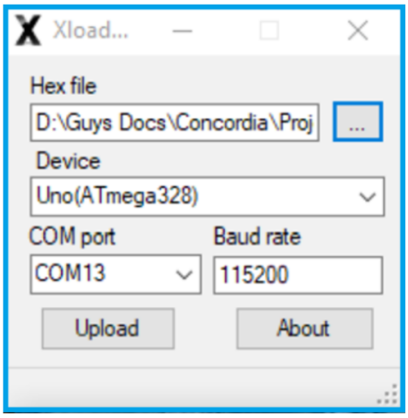
The ArduBridge software has two parts, the firmware that runs on the Arduino and the Python module that runs on the PC. Each part should be installed independently. The Firmware installation procedure is usually called programming or uploading.

Firmware upload

To program the Arduino with the compiled firmware (FW), use the xloader application in the ArduBridge folder and follow the steps below:

1. Open the xloader application (Figure 4A).
2. Select the COM port that the Arduino Uno R3 is connected to.
3. Select the Hex file "Bridge_Ctrl_Vxx.ino.hex" (or the most updated FW you have) (Figure 4B).
4. Press the "Upload" button and wait for the programming operation to finish after a few seconds.

A



B

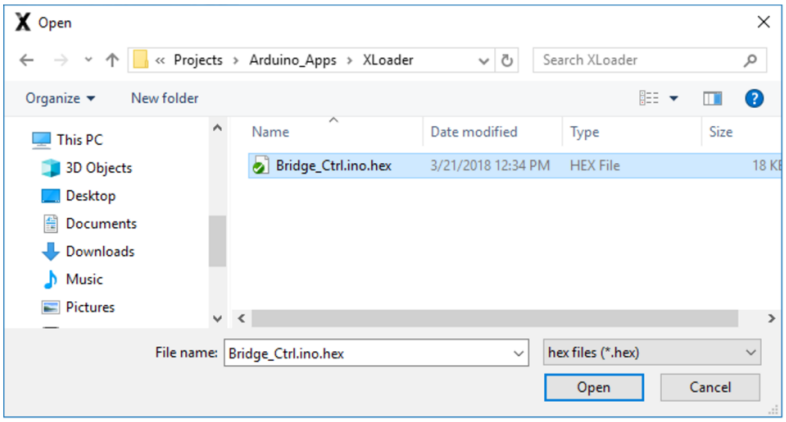


Figure 4 - Xloader application

(**A**) Select the ***Uno (ATmega328)*** Device with baudrate set to ***115200***. The COM port value is detected via the device manager tool (in Windows based machines). To select the Hex file, press the '…' button to open an explorer dialog box (**B**) and select the latest firmware file.

Python modules installation

Before installing the ArduShield module, make sure Python 3.7 (or above) is installed together with the latest pyserial module. Installation of the ArduBridge can be done in two ways:

1. By double clicking the setup.bat batch file (Figure 5A).
2. By running the setup.py script in a command prompt ("CMD") window (Figure 5B).

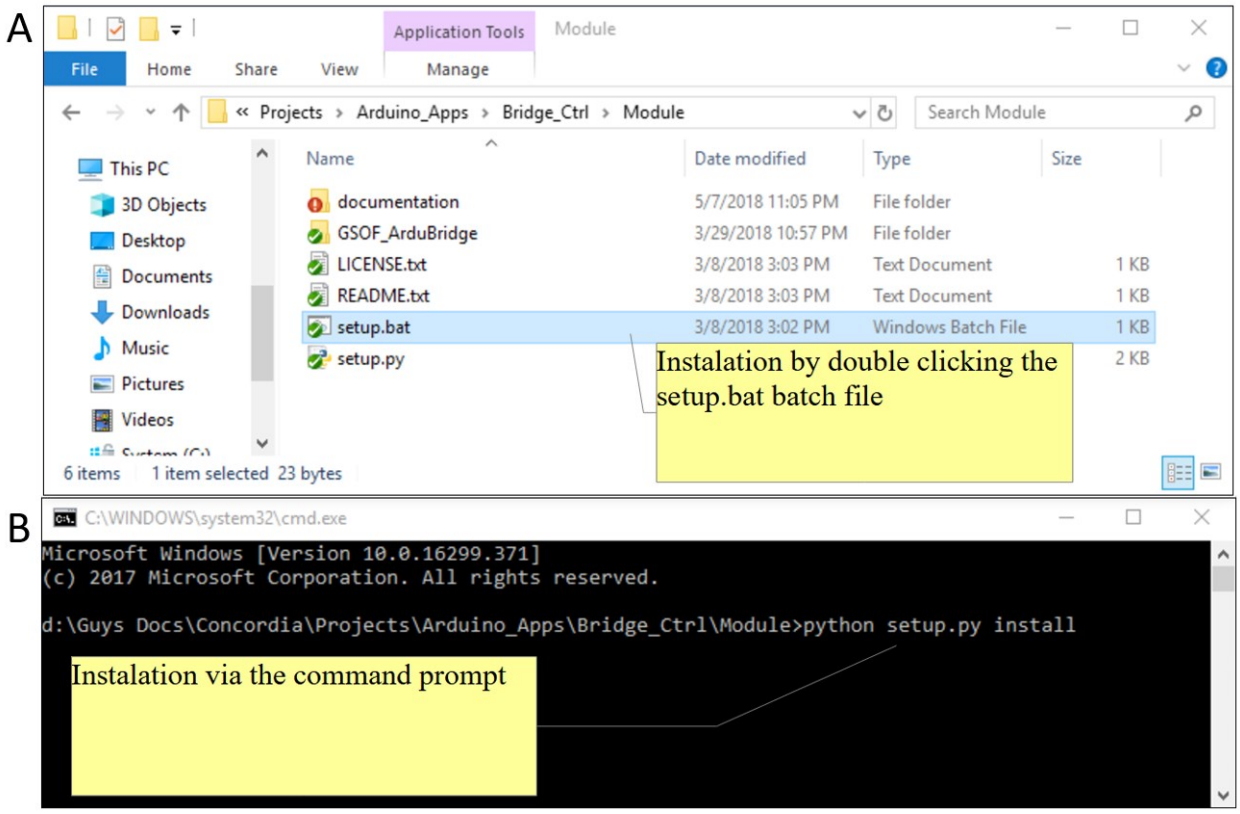


Figure 5 - Two options to install the ArduBridge Python module

To install the Python GSOF_ArduBridge module perform the following: (**A**) When using the windows file explorer double click the ***setup.bat*** file or (**B**) open the command prompt, navigate to the proper location, and enter `python setup.py install <ENTER>`.

Power supply configurations

The ArduShield can operate from either single, dual, or triple power supplies. The Arduino board is driven from its ***USB*** power. The ***VIN*** voltage should be between 7.5V to 12V, and it is regulated to 5.0 V by an onboard voltage regulator. The regulated 5.0 V is routed to the PWM, AN, GPIO, and expansion connectors. The ***+12V*** input is routed directly to the dual H-bridge circuit and should be between 5V to 12 V. In case ***VIN*** and ***+12V*** are the same, R6 and R7 can be jumpered to allow a single power supply configuration (Figure 6A). R6 and R7 are disconnected by default. The ***HV*** input is routed directly to the expansion connector (SV2), and its voltage depends on the expansion card specifications.

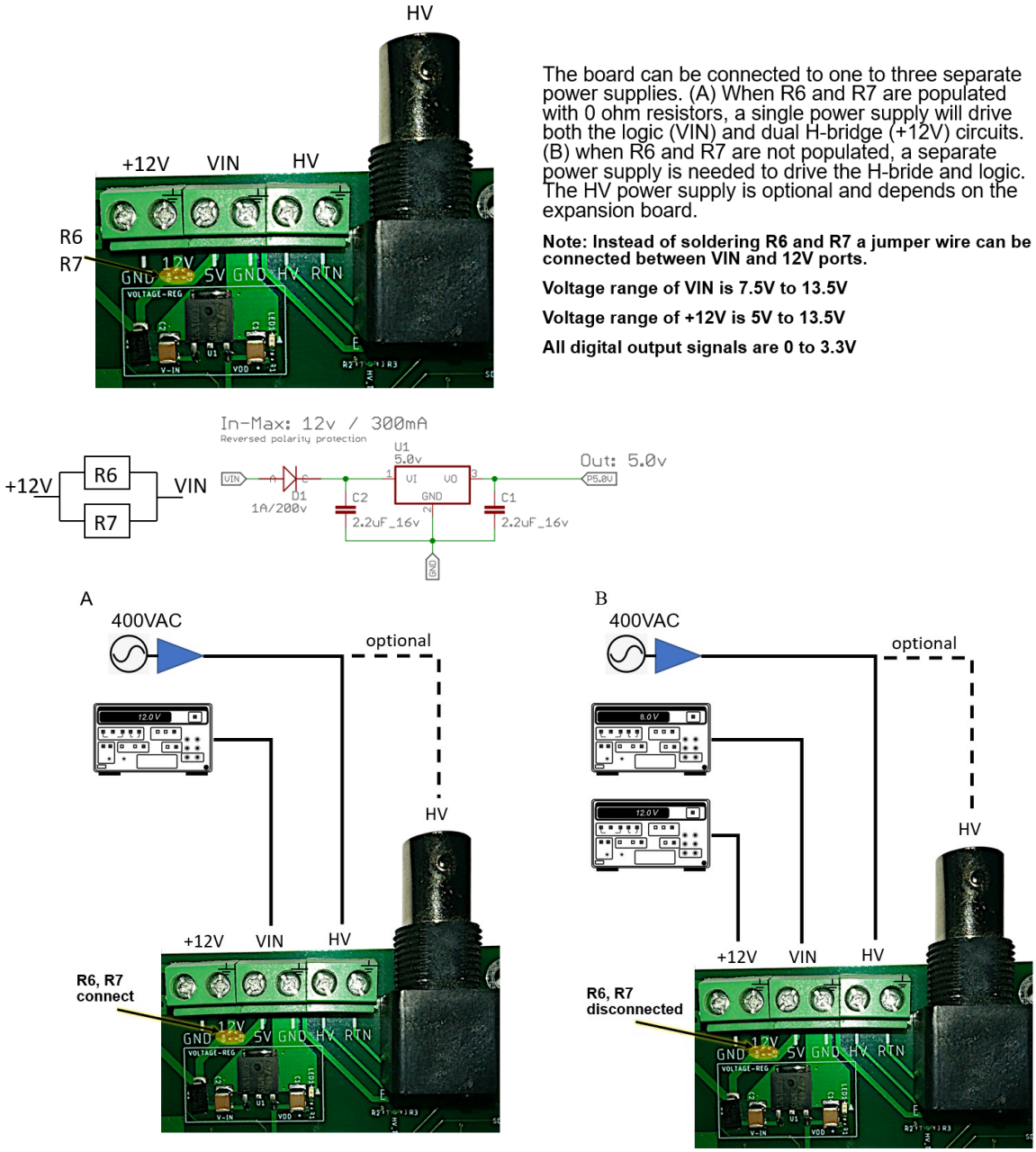


Figure 6 - Example of ArduShield power supply configurations

Expansion connector and stackup support

The ArduShield can be connected to any expansion board that uses I2C communication. The expansion connector can deliver two power supply voltages and grounds (+5V / +5_RTN and +V / +V_RTN). The ITO_RTN pin is dedicated for impedance measurement for DMF setup. The SDA and SCL are the standard I^2C bus signals. Figure 7A shows a stackup configuration in which the ArduShield is connected to two extension boards using only the expansion connector SV2 (Figure 7B). This eliminates cables and reduces the overall size of the setup.

Figure 7 - Example of ArduShield and expansion boards in stackup configuration

(**A**) Stackup of three boards, bottom two (red) are DMF drivers and on top (green) is the ArduShield.
(**B**) The pin arrangement and signal names of the expansion connector.

NOTE:	The SDA and SCL signals are also on the GPIO header. The SDA and SCL signals are connected to AN4 and AN5 (in the Arduino Uno). Remove any filtering capacitors and disconnect any loads from AN4,5.

Main configuration script

This chapter shows how to build and initialize ArduBridge and ArduShield objects.

Initializing the ArduBridge

Configuring the ArduBridge for use is simple and mostly automatic. The only parameter that should be manually edited is the COM or DEV name that the Operating System (OS) assigned to the Arduino. A simple template that builds the global ardu and ards objects and starts communication is included for convenience. It can serve as a starting point for any project. To complete the project, the user should add relevant imports, update the 'port' variable and add his code at the bottom (Figure 8).
The ArduShield class has the 'an_ref' variable for accurate ADC to voltage conversion. Its value depends on the Arduino and is usually configured to +3.3V. Currently, it's only essential when reading the temperature from the embedded temperature sensor (TC1047).

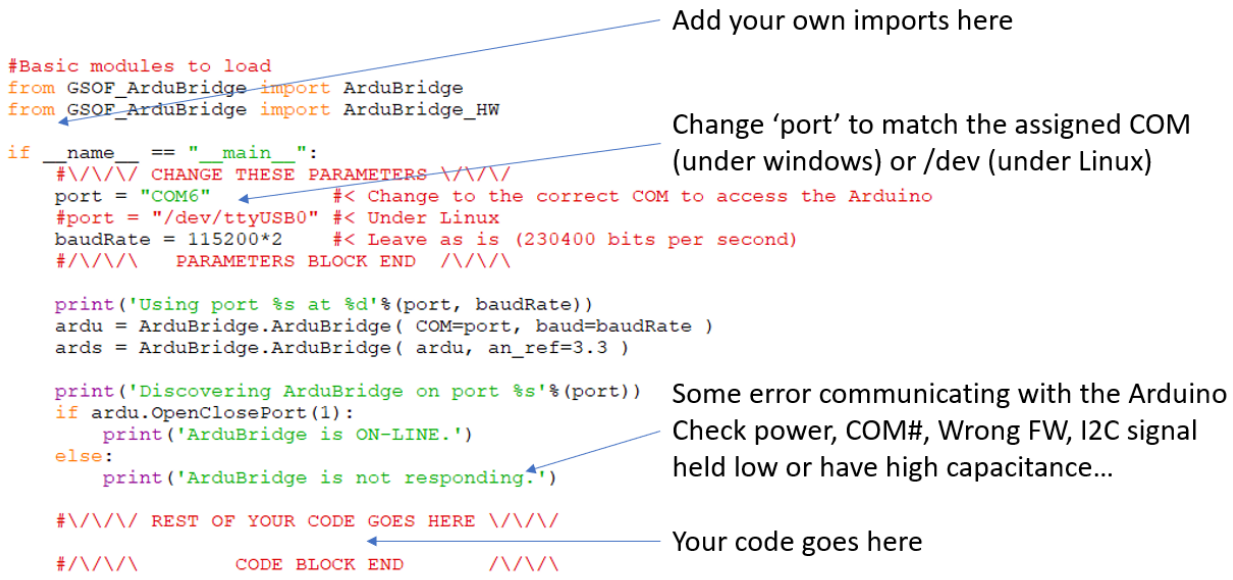


Figure 8 - Example of a basic ArduBridge configuration script

Analog To Digital (ADC) circuit

Each analog channel has an optional pull-up resistor to support a thermistor connection. In addition, a first-order Resistor-Capacitor (RC) Low-Pass-Filter (LPF) with a cutoff frequency of 12 Hz (R=1K Ohm, C = 1uF). The +V pin of all channels can be set to 3.3V or 5.0V by assembling R3 or R4. The analog connectors match the Arduino A0 to A5, respectively.

Class

The ArduBridgeAn class provides methods for interacting with the analog inputs and outputs of the Arduino via a serial connection.

Methods

`ardu.an.readAnalog(pin) - Returns a decimal value between 0 to 1023.`
`ardu.an.writeAnalog(pin) - Refer to the Pulse Width Modulation (PWM) output circuit chapter.`

Examples

`B = ardu.an.readAnalog(pin=3)` - Store the sampled voltage at pin<3> to variable B.

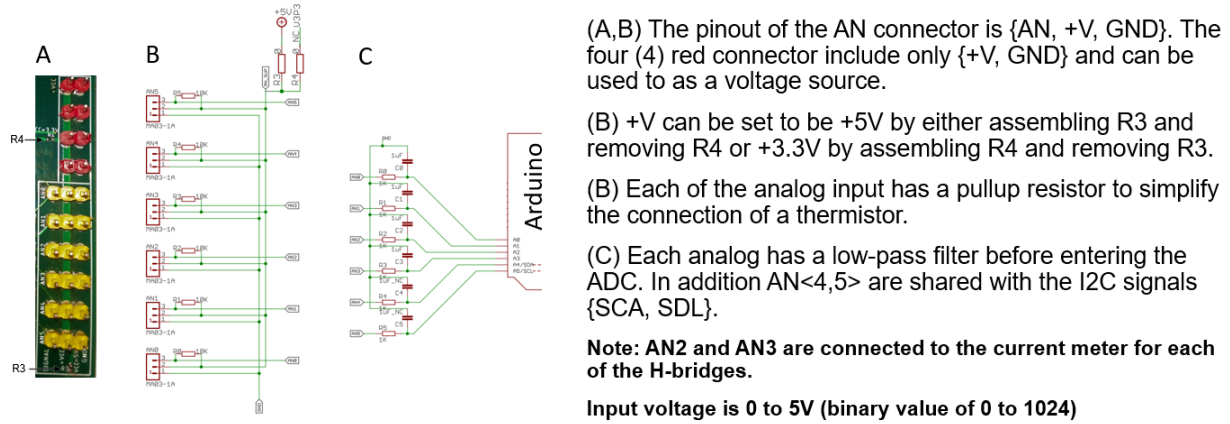


Figure 9 - The ArduShield ADC input circuit and connectors

The ANx<3< connectors are routed to the Arduino analog inputs via low-pass-filter. An additional pullup resistor can also be added on each channel. In addition, R3 and R4 can be set to select 3.3V or 5.0V supply voltage on pin ANx<2>.

NOTE:	Only pins A0 to A5 of the Arduino Uno can perform the Analog to Digital Conversion (ADC). Any pull-up resistors (R0 to R5) may be removed or modified to different values according to need. Different reference voltages can be supplied to the ADC via any +V pins if R3 and R4 are removed. This voltage is shared by all channels. AN4 and AN5 signals are connected to SCL and SDA (in the Arduino Uno).

GPIO, I²C, and SPI circuit

All digital signals can be found on the SV1 and SV2 connectors (ass a single row). All signals have an output voltage of 0 or 3.3V. The GPIO signals are used to drive electronic circuits while the I²C, and SPI signals are used to communicate the Integrated Circuit (IC) devices that support either of the two protocols.

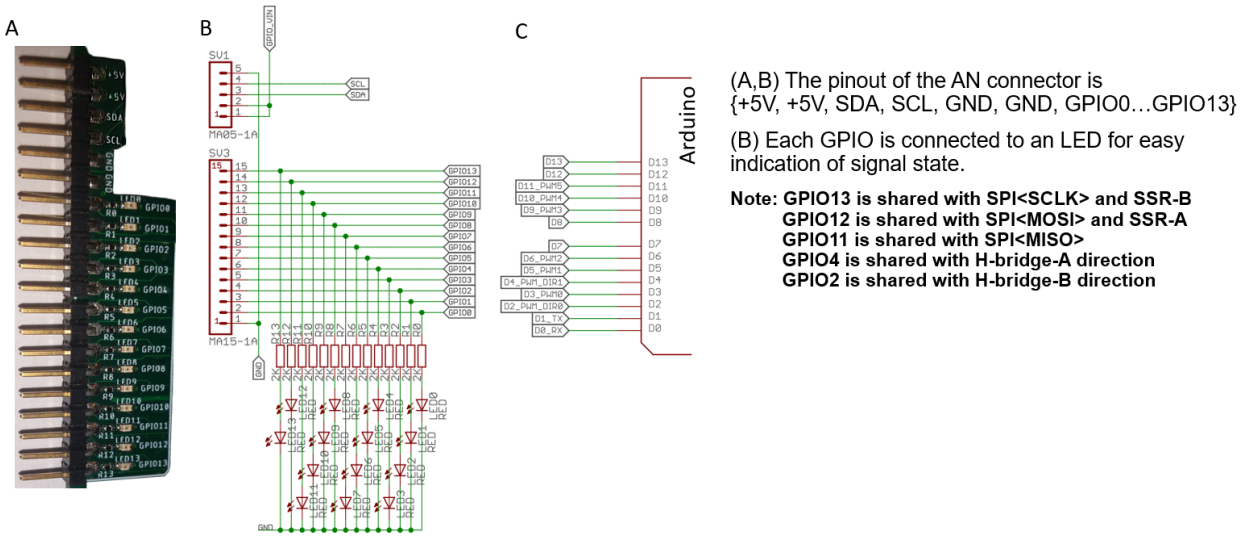


Figure 10 - The ArduShield GPIO circuit and connectors

Each GPIO is connected to an LED to clearly indicate the signal state.

General Port Input Output (GPIO) class:

The class provides methods for interacting with the digital inputs and outputs of an Arduino via a serial connection. It uses the BridgeSerial object to communicate over serial with the GSOF_ArduinoBridge firmware.

Constants:

```
ardu.gpio.INPUT
ardu.gpio.OUTPUT
```

Methods:

```
ardu.gpio.pinMode(pin, dir)
ardu.gpio.digitalWrite(pin, v)
ardu.gpio.digitalRead(pin)
ardu.gpio.pinPulse(pin, onTime)
```

Examples

`ardu.gpio.pinMode(pin=10, dir=ardu.gpio.INPUT)` - Set pin 10 to input

`ardu.gpio.pinMode(pin=9, dir=ardu.gpio.OUTPUT)` - Set pin 9 to output

`ardu.gpio.digitalRead(pin=10)` - Read digital pin 10

`ardu.gpio.digitalWrite(pin=9,val=1)` - Set pin 9 to high voltage

`ardu.gpio.pinPulse(pin=9,onTime=0.15)` - Set pin 9 to high for 150 ms and then to low

Inter-Integrated Circuit (I²C / IIC) class

I2C uses only two bidirectional open-drain signals: Serial Data (SDA) and Serial Clock (SCL), both are pulled up with resistors (2.2K Ohm) to the bus voltage (+3.3 V) by default (Figure 11). Usually, the bus address space is a 7-bit, with a rarely used 10-bit extension. Typical bus speeds are 100, 400, 1000, 3400, and 5000 kbit/s, but arbitrary clock frequencies are also allowed. Although the I2C standard supports clock stretching, the Arduino doesn't support it.

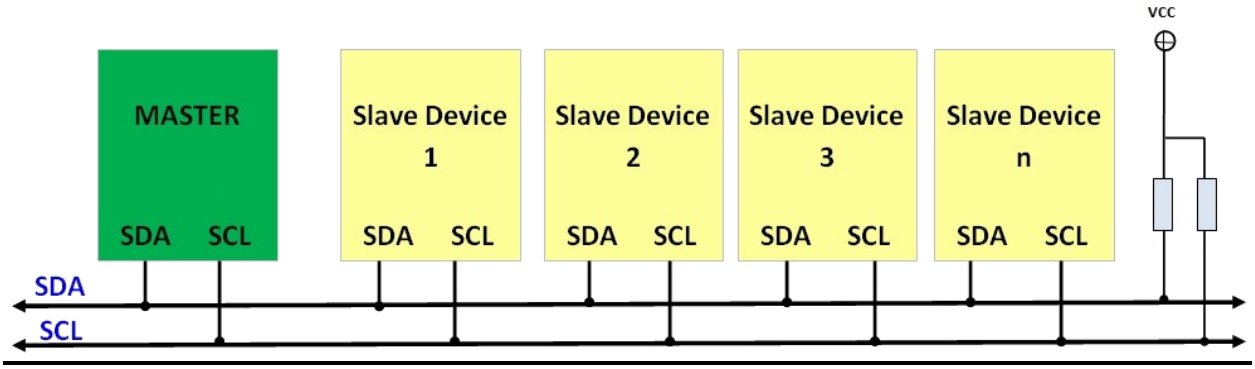


Figure 11 - The I2C bus signals and interconnection

Methods

`ardu.i2c.setFreq(freq)` - Set the bus frequency (hz).
`ardu.i2c.writeRaw(dev,vByte)` - Read vBytes to a device without internal addressing.
`ardu.i2c.readRaw(dev,N)` - Read N bytes from a device without internal addressing.
`ardu.i2c.writeRegister(dev,reg,vByte)` - Write vByte to specific device register.
`ardu.i2c.readRegister(dev,reg,N,delay=0)` - Read N bytes from specific device register.

The read methods return a list with two elements. The first element is a pass/fail boolean value. The second element is a list of the received bytes.

Examples

`ardu.i2c.setFreq(freq=100000)` - Set the bus frequency to 100K Hz
`ardu.i2c.writeRaw(dev=0x52,vByte=[5,7,6])` - Write bytes 5,7,6 to device 0x52
`ardu.i2c.readRaw(dev=0x50,N=10)` - Read 10 bytes from device 0x50
`ardu.i2c.writeRegister(dev=8,reg=2,vByte=[1])` - Write 1 to register 2 in device 8
`ardu.i2c.readRegister(dev=6,reg=10,N=5)` - Read 5 bytes from register 10 in device 6

Serial Peripheral Interface (SPI) class

The SPI is a full duplex, master-slave bus. All devices share the same clock (CLK), Master-Out-Slave-In (MOSI), and Master-In-Slave-Out (MISO) signals. In addition, every device has a dedicated Slave-Select (SS) signal for direct addressing. The Arduino's (master) clock output is connected to the CLK signal. Its Serial-Data-Out (SDO) pin is connected to the MOSI signal, and its Serial-Data-In (SDI) is connected to the MISO signal. Slave devices connect their clock pin to the CLK signal, their SDO pin to the MISO signal, and their SDI pin to the MOSI signal. In addition, the SS pin of the device is connected to the dedicated General-Pin-Output (GPO) on the Arduino (Figure 12).

To start a transaction, first, configure the Arduino to the correct mode (CLK and sampling polarity) using the method `ardu.spi.setMode(mode)`. Next, assert the proper SS signal using the method `ardu.gpio.digitalWrite(pin, val)`. The proper mode and SS polarity are defined in the device specifications. Then, send and receive the data using the method `ardu.spi.write_read(vByte)`. Lastly, release the SS signal.

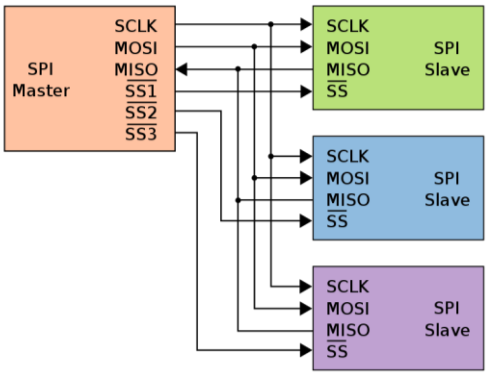


Figure 12 - The SPI bus signals and connection to slave devices

Methods

`ardu.spi.setOff()` - Turn off the SPI peripheral.

`ardu.spi.setMode(mode)` - Set the SPI mode {0,1,2,3}.

Where mode:
mode = 0 - Clock is normally low ,and data is sampled on the transition from low to high.
mode = 1 - Clock is normally low ,and data is sampled on the transition from high to low.
mode = 2 - Clock is normally high ,and data is sampled on the transition from high to low.
mode = 3 - Clock is normally high ,and data is sampled on the transition from low to high.

`write_read(vByte)` - Send and receive a list of bytes, where `vByte` is the list of bytes to send. The method returns a list with two elements. The first element is a pass/fail boolean value. The second element is a list of the received bytes.

Examples

`ardu.spi.setMode(mode=2)` - Set the SPI to mode 2.

`B = write_read(vByte=[0x12, 0x22, 0x56])` - Send byte 0x12, 0x22 and 0x56 and store the result in variable B (B[0] - pass/fail, B[1] - received bytes).

NOTE:	The number of bytes received is identical to the amount sent. It is impossible to receive data without sending data over the SPI.

Pulse Width Modulation (PWM) output circuit

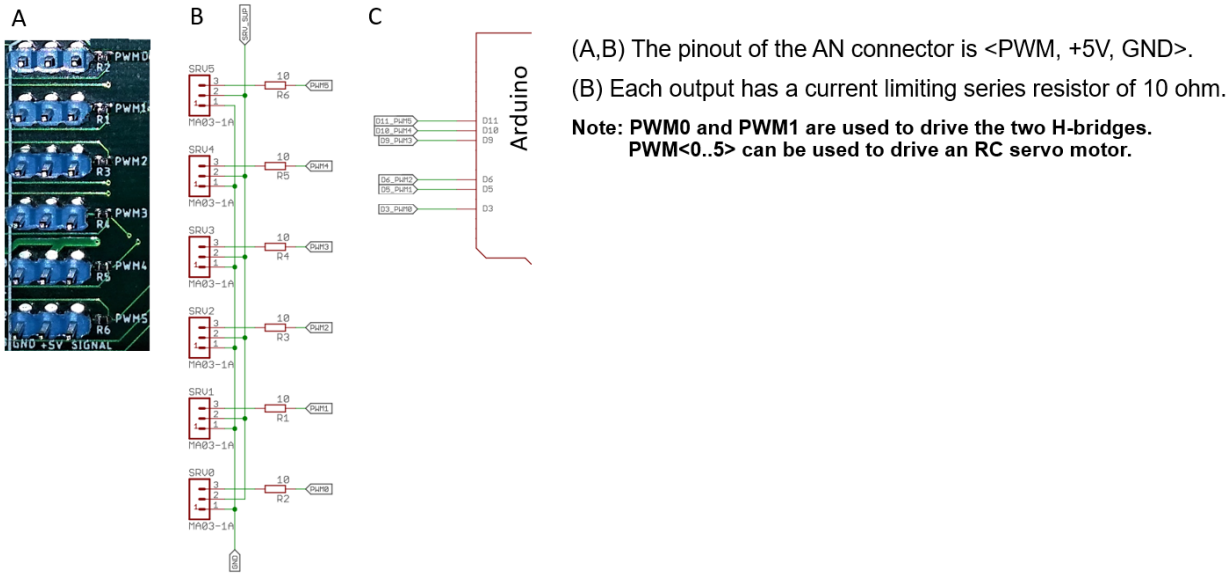
PWM is an efficient way to generate a multi-level signal using an oscillating digital signal. This is useful to control the amount of energy delivered to a load such as LED, motor, or heater. The control over the average amount of energy delivered is done by controlling the duration the digital signal is kept at high and low voltages. The ratio between the two periods is proportional to the average energy delivered. This technique is known as Pulse-Width-Modulation (PWM). The switching frequency between the signal's states is ~500 times per second (Hz) and the timing resolution is ~15 microseconds. The ArduShield can generate up to six (6) PWM signals with values ranging from 0 to 255. The PWM connectors are also used to control RC servos or ESC (refer to ***Servo control***).

Methods

`ardu.an.writeAnalog(ch, val)`

Example

`ardu.an.writeAnalog(ch=0, 255)` - Set PWM0 output to maximum (constant high)
`ardu.an.writeAnalog(ch=4, 128)` - Set PWM4 output to minimum (constant low)
`ardu.an.writeAnalog(ch=5, 0)` - Set PWM5 output to 50% duty-cycle



(A,B) The pinout of the AN connector is <PWM, +5V, GND>.

(B) Each output has a current limiting series resistor of 10 ohm.

Note: PWM0 and PWM1 are used to drive the two H-bridges. PWM<0..5> can be used to drive an RC servo motor.

Figure 13 - The ArduShield PWM / servo output circuit and connectors

NOTE:	-The analog class contains the methods to generate PWM output. -The PWM connector pinout is plug-and-play compatible with RC servos. -PWM0 to PWM5 can be used to drive an RC servo. -PWM0 and PWM1 are shared with the H-Bridge–A (HA) and H-Bridge-B (HB) circuitry, respectively. -Enabling servo control disables analogWrite() functionality on PWM[3,4] (GPIO[9,10])

Servo control

Any GPIO pin can be set to generate a valid RC servo control signal with a pulse width of 1 to 2 ms at 20hz. Nevertheless, the ArduShield PWM connectors have the standard pinout to connect an RC servo.
Both *ardu* and *ards* objects include methods for servo control. The main difference is that the *ardu* methods address GPIO pins while the *ards* methods address PWM channels. In addition, they are slightly more polished and user-friendly.
In both cases, the pin mode must be set to SERVO before issuing a set position command. The servo position is an integer value between 0 to 255. The scale factor to convert between degrees to binary value depends on the specific servo model, which is up to the user to find.
The *ards* object includes a *servoScurve()* method to perform smooth motion between two positions using the 'S' curve profile. The user has the freedom to specify the maximum acceleration (acc) and update rate (DT). The function blocks the program's execution during the servo movement.

Servo functions

`ards.servoMode(ch, on)` - If *on* is True, set pwm *ch* to servo control else, set *ch* to GPIO
`ards.servoSet(ch, val)` - Set the servo's position at ch to val
`ards.servoScurve(ch, P0, P1, acc=200, DT=0.05)` - Smooth 'S' curve transition from position P0 to position P1 (blocking function).
`ardu.gpio.pinMode(pin, ardu.gpio.SERVO)` – set pin to servo control
`ardu.gpio.servoWrite(pin, val)` – Set the servo's position at pin to val

Examples

To control a servo on PWM channel 0 (pin 3) and set it position to 128, use:

```
ards.servoMode(ch=0, on=True) #< Set PWM[0] to servo control
ards.servoSet(ch=0, val=128) #< Set the position of the servo at PWM[0] to 128
```

or

```
ardu.pinMode(pin=3, mode=ardu.gpio.SERVO) #< Set GPIO[3] to servo control
ardu.servoWrite(pin=3, val=128) #< Set the position of the servo at GPIO[3] to 128
```

To control a servo on PWM channel 5 (pin 11) and smoothly move it from position 0 to 64 use:

```
ards.servoMode(ch=5, on=True) #< Set PWM[0] to servo control
ards.servoScurve(ch=5, P0=0, P1=64) #< Move the servo at PWM[0] from 0 to 64
```

NOTE:	The PWM connector pinout is plug-and-play compatible with RC-servos. PWM0 to PWM5 can be used to drive an RC servo. PWM0 and PWM1 are shared with the H-Bridge–A (HA) and H-Bridge-B (HB) circuitry, respectively.

Dual Solid State Relay (SSR) circuit

The SSRs are great for turning ON / OFF high-power devices such as LEDs, heaters, DC motors, fans, heaters, LASER, electromagnets, relays…
Each SSR is connected to the +12V signal and can conduct up to 4.5 Amperes.

Methods

`ards.ssrA(on)` - Set the state of SSR-A to ON/OFF.
`ards.ssrB(on)` - Set the state of SSR-B to ON/OFF.

Where
val - is an integer value. Positive values mean ON, and negative or zero values mean OFF.

Examples

`ards.ssrA(on=1)` - Close SSR-A and short circuit F0<1> to GND.
`ards.ssrB(on=0)` - Open SSR-B and keep F1<1> floating.

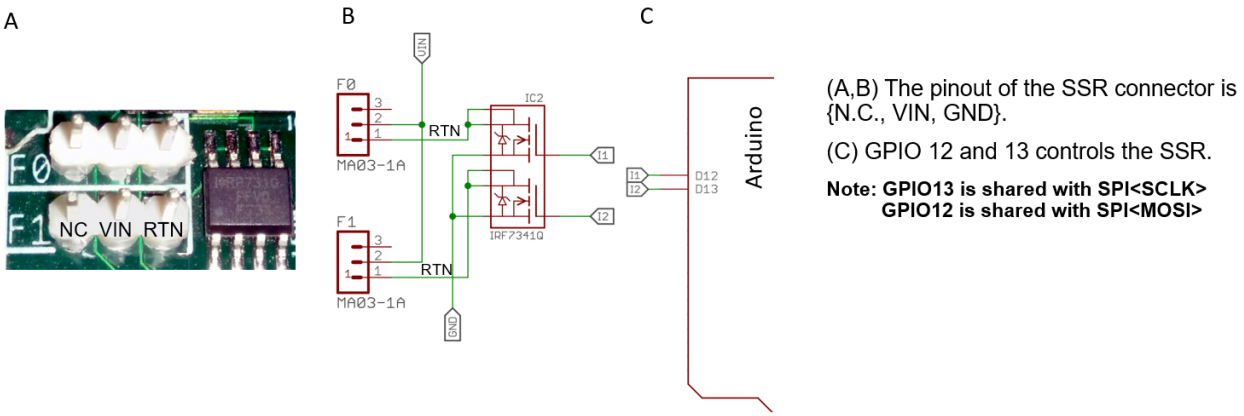


Figure 14 - The ArduShield SSR circuit and connectors

NOTE:	There is no need to set the pin mode before calling ssrA or ssrB methods. The load connected to the SSR should not consume more than 4.5 Amps

Dual H-Bridge circuit

The ArduShiled includes a dual H-bridge circuit with a current sensor for each channel. Each PWM channel can accept 9-bit sign values (-255 to +255). The current sensor returns a 10-bit binary value (0 to 1023), where 512 represents zero current. The circuit uses the Arduino 8-bit PWM<1,2> and GPO<2,4> to set the H-Bridge polarity. The circuit can deliver +/-1A without heat-sink and up to +/-3A with.

Methods:

`ards.pwmA_init()` - Must be called before calling the pwmA() method.
`ards.pwmB_init()` - Must be called before calling the pwmB() method.
`ards.pwmA(val)` - Generates PWM signal on HB-A output.
`ards.pwmB(val)` - Generates PWM signal on HB-B output.
`ards.pwmA_cur()` - Returns the current on HB-A (0 to 1023)
`ards.pwmB_cur()` - Returns the current on HB-B (0 to 1023)

Where:
val - is a sign integer value between *-255 to 255*

Example

`ards.pwmA(150)` - Generates PWM signal on HB-A output (current flow from OUT1 to OUT2).
`ards.pwmB(-50)` - Generates PWM signal on HB-B output (current flow from OUT2 to OUT1).
`B = ards.pwmA_cur()` - Store the current reading on HB-A to variable B.

NOTE:	In case separate power supplies are used to drive the Arduino logic and H-Bridge. Make sure to connect the Arduino USB or power supply before the H-Bridge power supply. The H-Bridge is designed to enter FAULT-MODE if the bus voltage (VIN) exists before applying the logic voltage (VCC). The H-Bridge output signal is floating (not referenced to GND). Therefore always connect the load between the HB+ and HB- pins. If continuous currents above +/-1 A are expected, connect a heatsink on the respective H-Bridge IC to protect it from overheating.

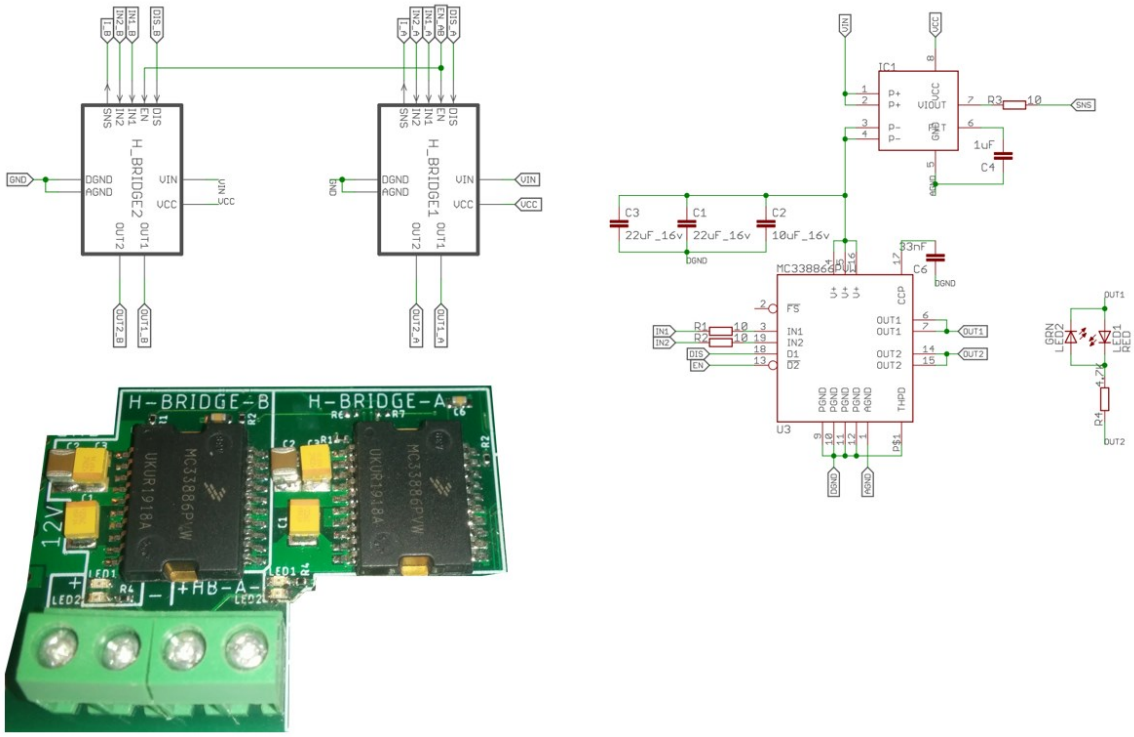


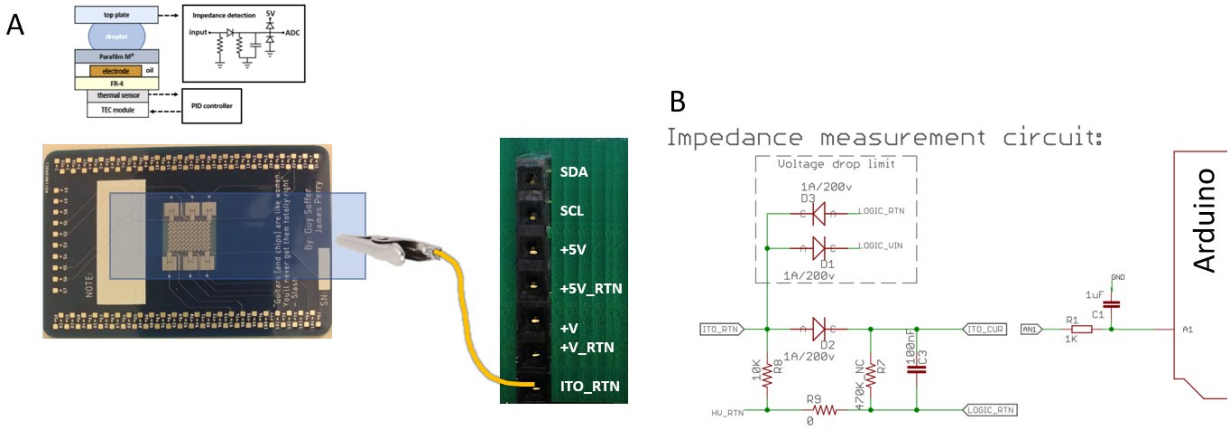
Figure 15 - The ArduShield Dual H-Bridge circuit and connectors

Digital Microfluidics (DMF) impedance measurement circuit

Digital microfluidics (DMF) is a platform to dispense, move, and mix water droplets in the microliter volume scale. The manipulation of droplets is done with a set of insulated electrodes. A unique feature of the ArduShield is its ability to measure the relative impedance between any of the electrodes to the ITO plate. This is done by measuring the current flow through the ITO to the electrical ground. The measurement circuit converts the current to voltage and holds its peak value before sampling it. The impedance is calculated by dividing the electrode's peak voltage by the measured peak current (R = V / I). The electrode's driving voltage should be known to the user before the measurement. The current is calculated by I = 5.0*Vbin/1024/10000.

`ards.getDmfChipCurrent(ch=3, units='bin')` - Returns the measured current in ADC units.

`ards.getDmfChipCurrent(ch=3, units='A')` - Returns the measured current in Amper units.



(A,B) The DMF connector pinout is <SDA, SCL, +5V, GND, +V, +V_RTN, ITO_RTN>. To ground the ITO connect it to the ITO_RTN pin. The return current will is used to measure the impedance between the electrode and ITO. The impedance is related to the size of droplet

Figure 16 - The ArduShield DMF impedance measurement circuit and connectors

Typical setup for electro-mechanical projects

The ArduBridge is a general-use board designed to expand the functionality of the Arduino Uno R3 with additional peripherals and easy-to-use connectors to attach standard devices such as RC servo motors, analog sensors, and high power actuators. The ArduBridge firmware makes it possible to execute Input Output (IO) commands from a PC that runs the main algorithm. This setup is advantageous for developing real-time control loops, integrating software and hardware, and developing complex algorithms that are hard to build and debug without modern and interactive tools.

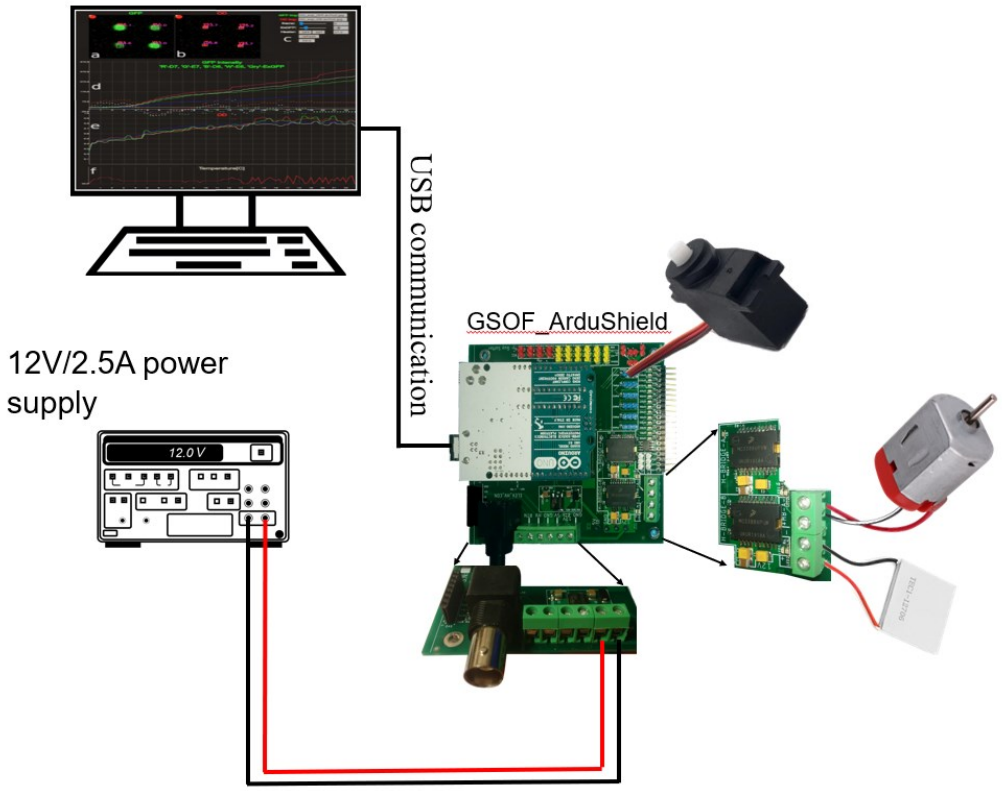


Figure 17 - Example of typical setup using the ArduShield

Typical setup with DMF electrode driver board

The typical setup shown in figure-xx and the DMF setup shown in figure-xx can be combined and further expanded with additional sensors and cooling fans (not shown).

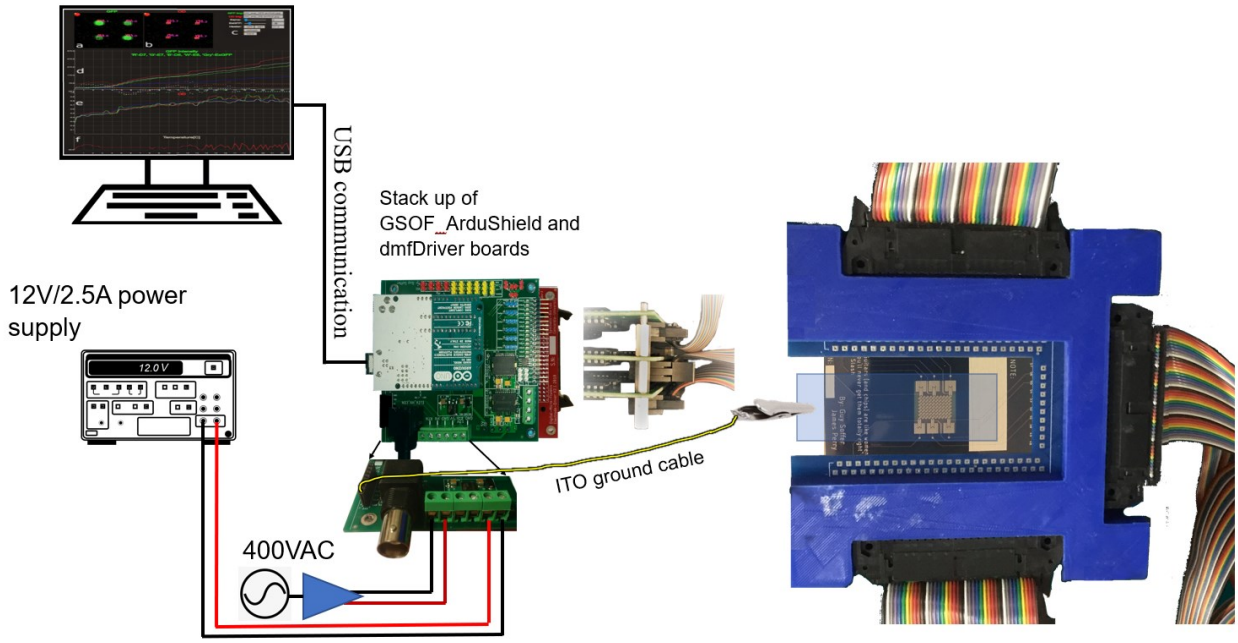


Figure 18 - Example of typical DMF setup using the ArduShield

NOTE:	The ITO ground (ITO_RTN) is connected to the logic ground via the impedance measurement circuit.