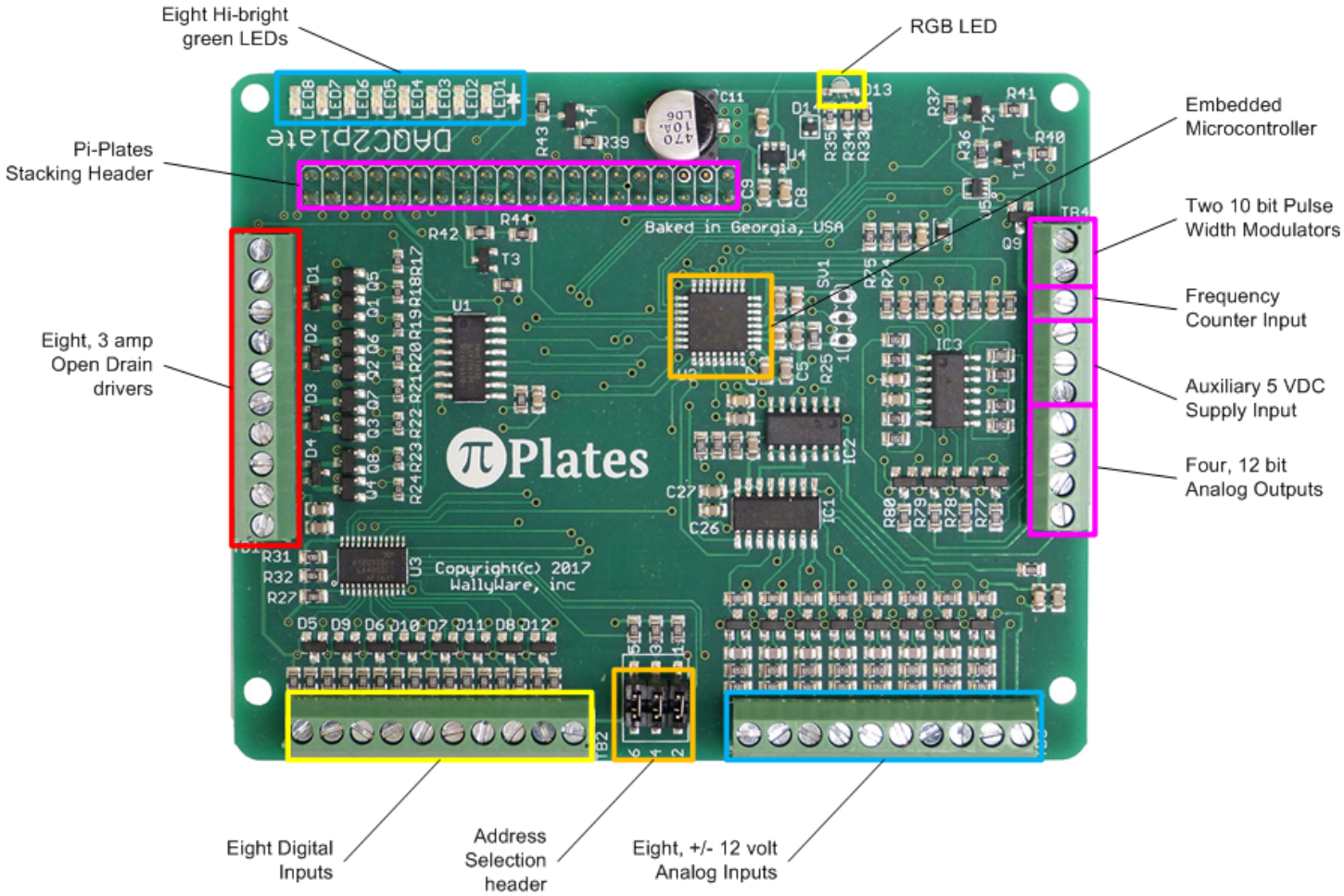
DAQC2plate Reference Guide

Version 1.04



Introduction

This document describes all of the features and functions of the Pi-Plates DAQC2plate. New to this design is a feature that allows the DAQC2plate to operate in four different functional modes. These are:

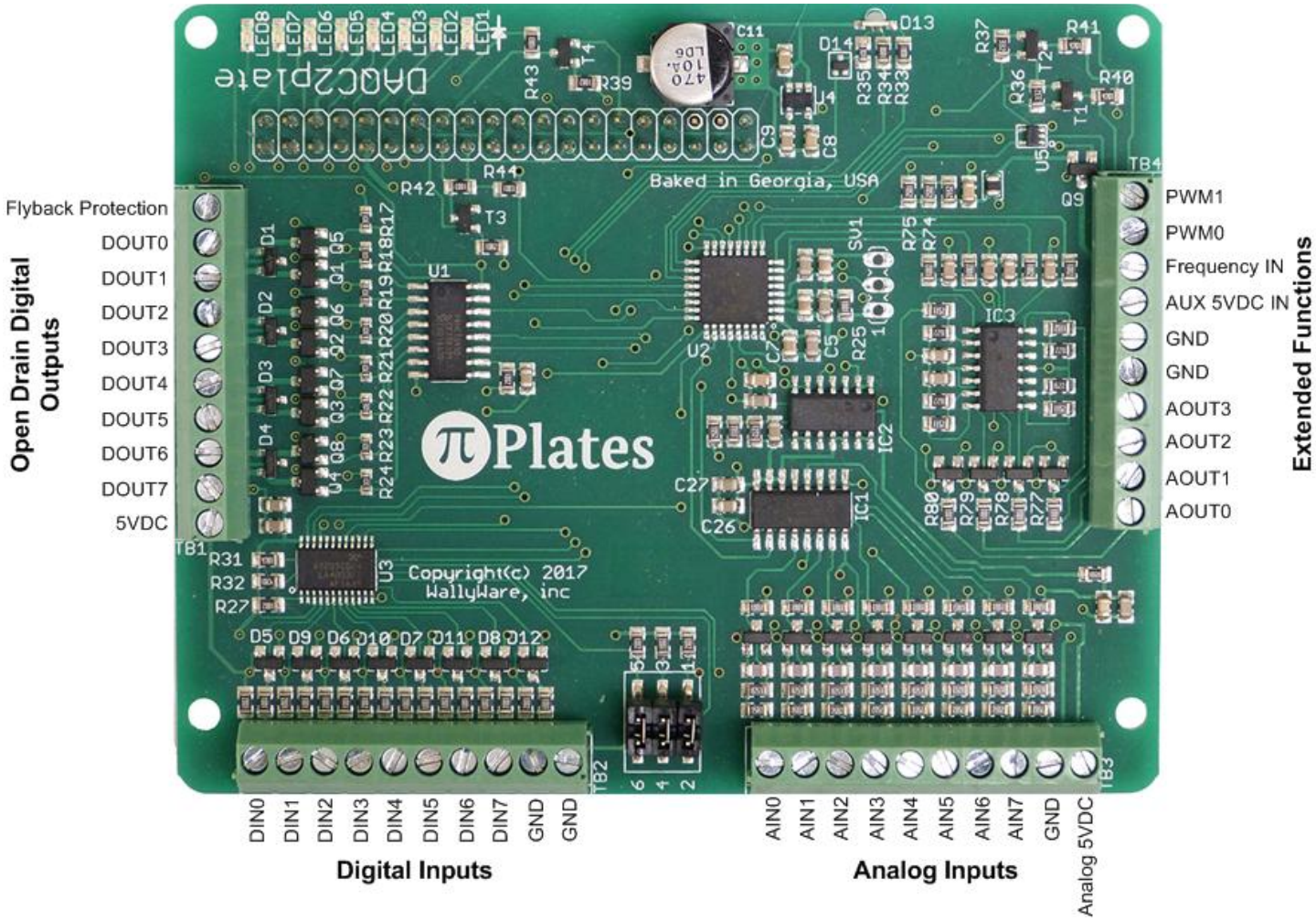
- Legacy Mode - the DAQC2plate will operate and use the same instruction set as the original DAQCplate
- Function Generator - in this mode, the DAQC2plate outputs two adjustable analog signals on DAC outputs 1 and 2
- Stepper Motor Controller - in this mode the DAQC2plate can independently drive two, unipolar stepper motors using the DOUT connectors
- Oscilloscope - in this mode the the DAQC2plate can be used as a two channel oscilloscope with a sample rate of 1 million samples / second. Analog inputs 0 and 4 are used in this mode.

Although the examples below have been written for Python 2.7, there are also modules available for Python 3. Finally, all of the code examples below assume that the DAQC2plate module was imported with the following statement:

import piplates.DAQC2plate as DAQC2

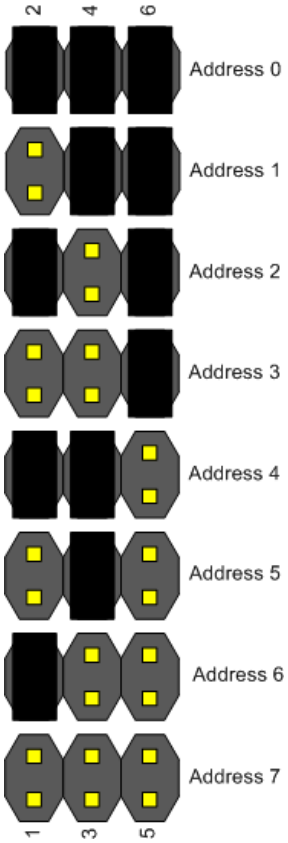
Board Layout

The terminal blocks and their functions are shown below. For convenience, the name of each terminal is printed on the bottom of the board. If you are familiar with the original DAQCplate, note that Digital Inputs Header and the Analog Inputs Header have been swapped. This was done to provide better signal to noise performance on the analog inputs.



Address Selection Header

Up to eight DAQC2plates can be used in a single stack of Pi-Plates. To do this, each board has to be set to a unique address. When shipped, the DAQC2plate is set to address zero. The address is set by positioning jumpers on the small, six pin header in the lower area of board as shown in the image above. Use the diagram below to set the address:



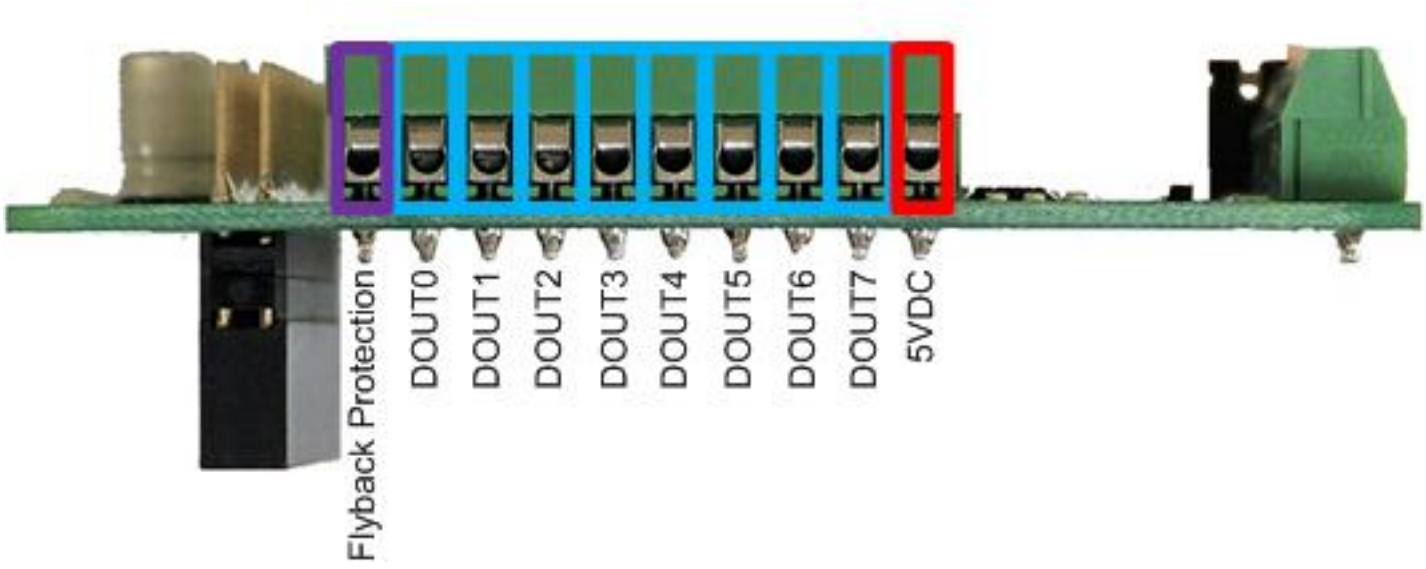
Note that Pi-Plates only read their address when they are powered up. So, you will have to cycle power to your stack after you change the board address.

Digital Outputs (DOUT)

The digital output connector provides eight open drain outputs, a 5VDC output for driving loads, and a flyback protection terminal for inductive loads. Use these outputs to drive LED strings, DC motors, relays, solenoids, buzzers, unipolar stepper motors, resistive heating elements, ultrasonic rangefinders, and incandescent automotive light bulbs. Each digital output also has a green LED connected to it. Whenever a digital output is turned on the corresponding green LED will come on. You do not need to connect anything to the digital outputs to control these LEDs. Conversely, these LEDs will not affect anything that you attach to the digital outputs. To obtain a simple digital signal with a DOUT pin, attach a 4.7K resistor between 5VDC and the output pin.

Connector

Refer to the diagram at the top of this page as well as the one below to locate the Digital Output terminals:



Specifications

- Each of the 8 Digital Outputs is an open drain transistor
- Maximum sink current is 3A
- Maximum load voltage is 30VDC
- On voltage is typically less than 50mV with a load current of 1A
- When using a relay or solenoid, the high side power supply should be connected to the "Flyback Protection" terminal
- If necessary, the on-board 5VDC is available on pin 10. If you choose to use it, then make sure your power supply can provide enough current for your Raspberry Pi (~700mA), your Pi-Plates (~100mA for each DAQC2plate with all LEDs on) and any components you connect to a DOUT terminal.
- To generate a simple digital signal output, connect a 4.7K resistor between 5VDC and the DOUT pin you wish to use. Note that the output will be inverted i.e. setting the pin will make the output zero volts while clearing it will make it 5VDC.

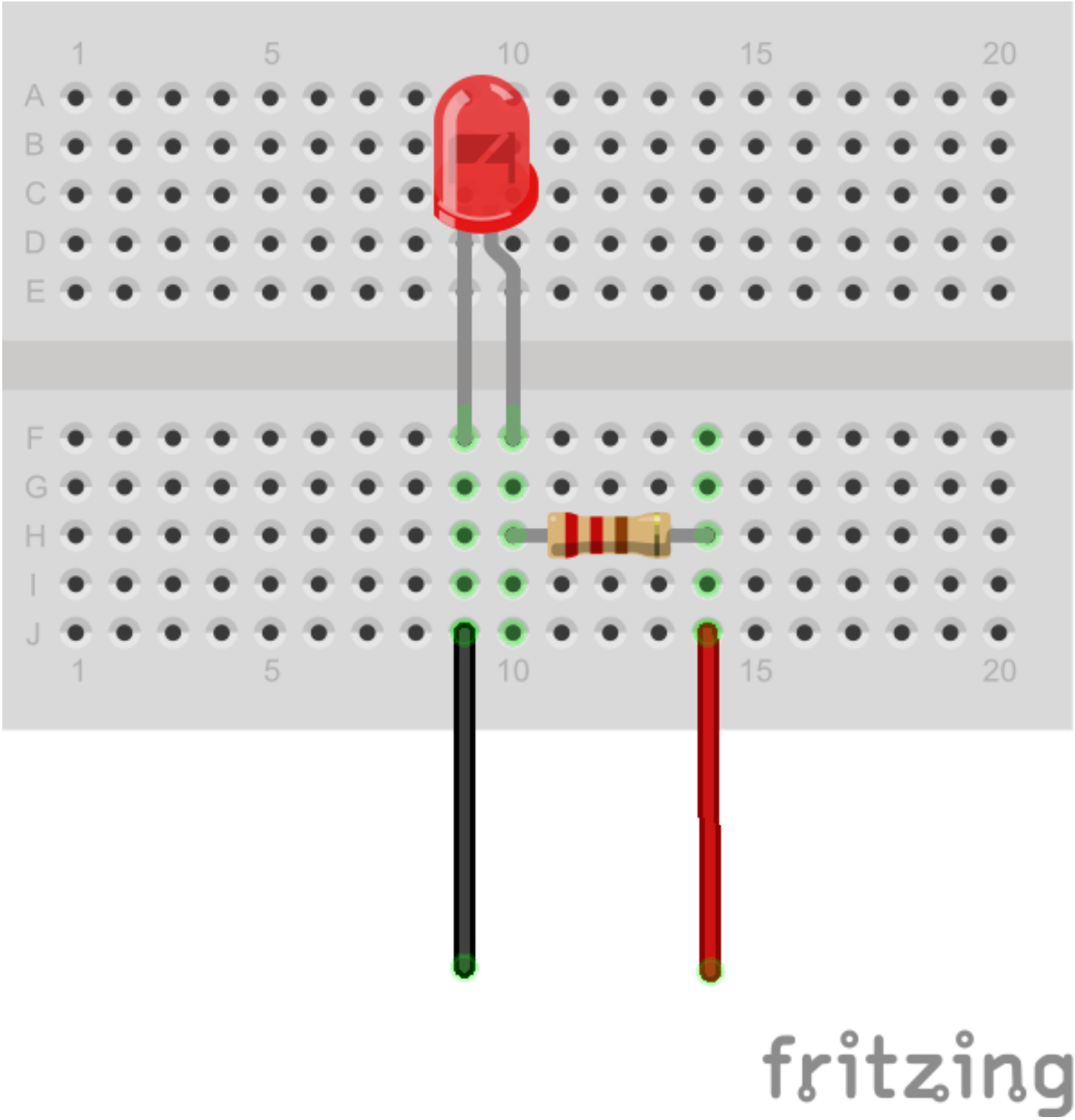
Functions

addr must be in the range of 0 through 7
bit must be in the range of 0 through 7

- setDOUTbit(addr, bit) - set single bit. Note that this turn ON the transistor and any device attached to it.
- clrDOUTbit(addr, bit) - clear single bit. Note that this turn OFF the transistor and any device attached to it.
- toggleDOUTbit(addr, bit) - toggle a single bit
- setDOUTall(addr,byte) - control all eight bits at once. The byte value must be in the range of 0 through 255.

Examples

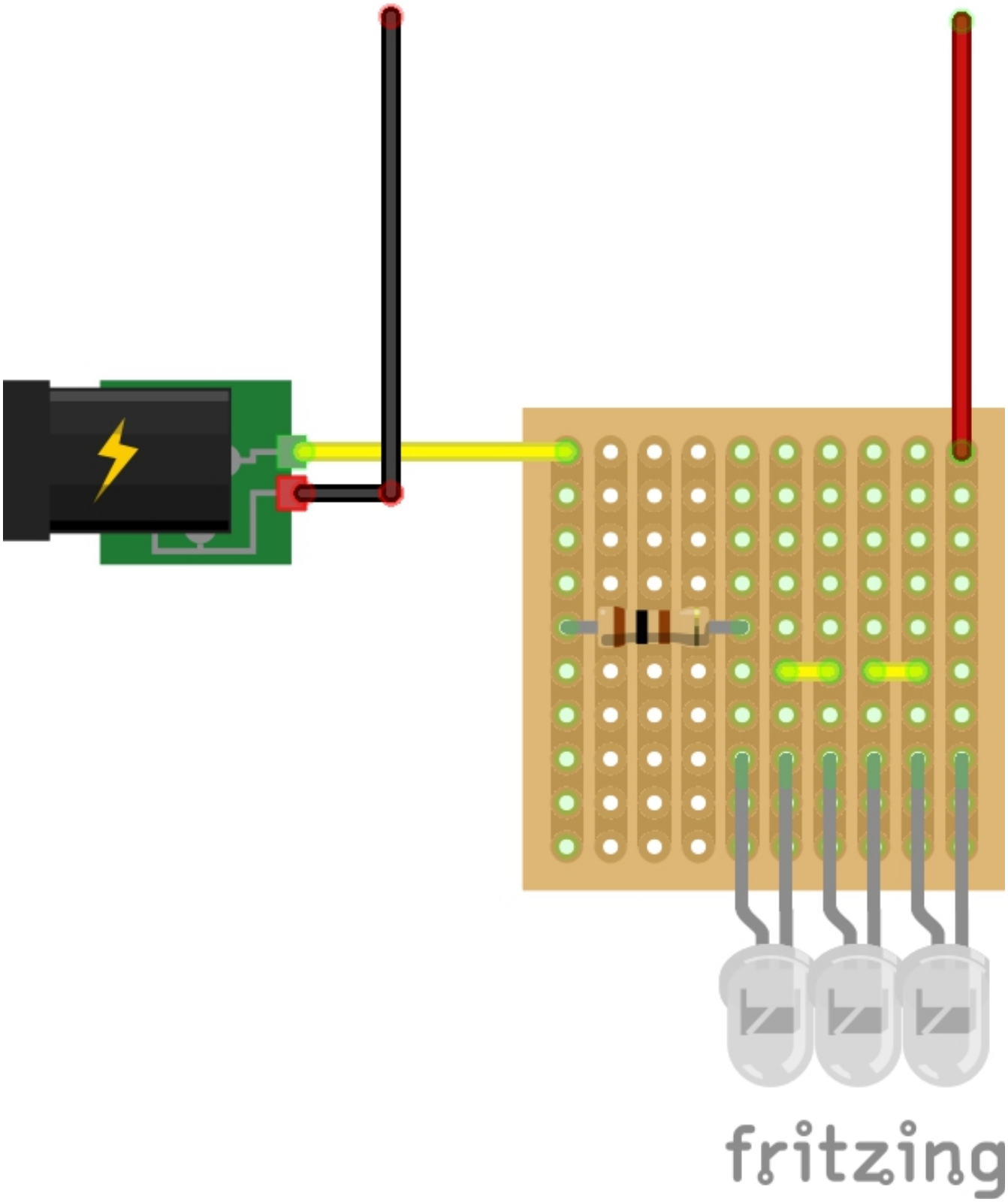
Simple External LED



Use a simple breadboard to build up this LED circuit. Connect the anode of the LED (the long wire) to one end of a 220 ohm resistor. Route a red wire from the other side of the resistor to the 5VDC terminal on the DAQC2plate digital output terminal block. Route a black wire from the cathode of the LED to Output 0 of the

terminal block. Assuming the DAQC2plate board is at address 0, type DAQC2.setDOUTbit(0,0) to turn the LED on. Type DAQC2.clrDOUTbit(0,0) to turn off the LED.

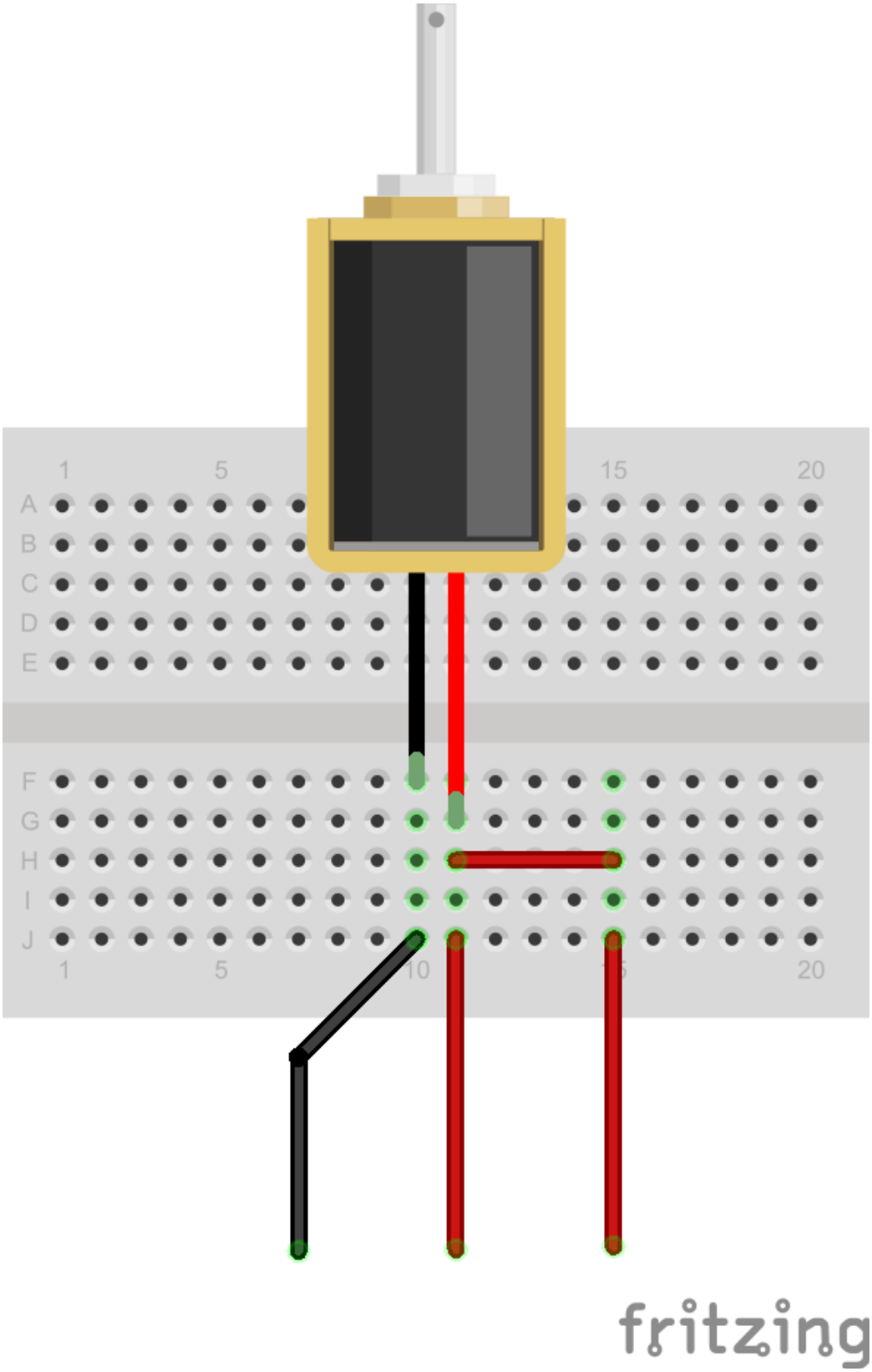
A Short String of White LEDs



Control some under-counter lighting with this circuit. Connect three, high-bright white LEDs in series as shown and then attach a 12VDC power supply with a center-positive connector. Then put a 100 ohm resistor between the LEDs and the power supply. Attach the cathode of the rightmost LED to DOUT0 of the terminal block (pin

1) and the ground of the power supply to pin 9 or 10 on the DIN terminal block. Again, assuming the DAQC2plate board is at address 0, type DAQC2.setDOUTbit(0,0) to turn the LED string on. Type DAQC2.clrDOUTbit(0,0) to turn off the LED string.

Driving an Inductive Load



1. Use a simple breadboard to build up this 5V solenoid circuit.
2. Connect the red wire from the solenoid to a pair of red wires that route from the breadboard.

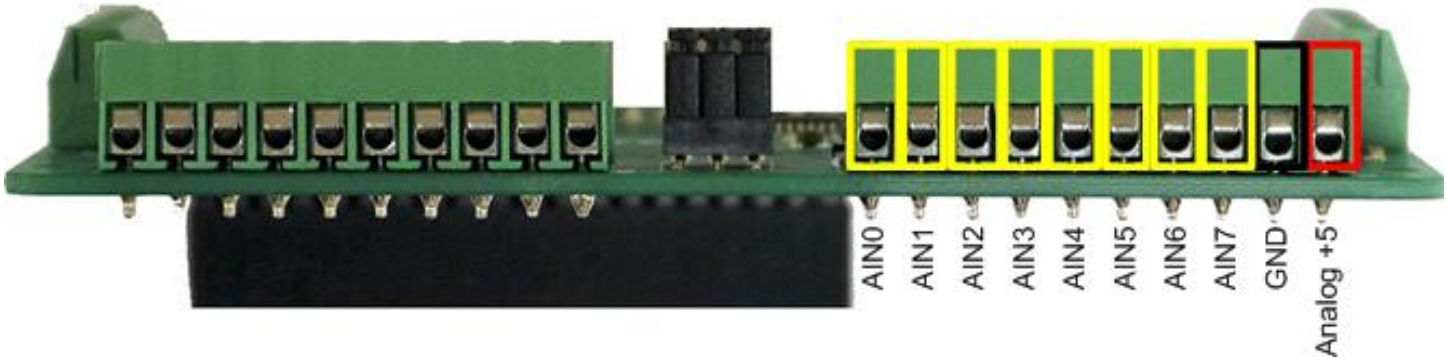
3. Attach one of these to the 5VDC terminal on the DAQC2plate digital output terminal block.
4. Connect the other red wire to the Flyback Protection terminal. Route a black wire to DOUT0 of the terminal block.
5. Assuming the DAQC2plate board is at address 0, type DAQC2.setDOUTbit(0,0) to turn the solenoid on.
6. Type DAQC2.clrDOUTbit(0,0) to turn off the solenoid.

Analog Inputs (ADC)

Use the analog input to measure sensors that produce a variable output voltage. These eight terminals can be used for measuring voltage, temperature, humidity, light brightness, potentiometers, strain gauges, and much more.

Connector

Refer to the diagram at the top of this page as well as the one below to locate the Analog Input terminals:



Specifications

- Eight inputs
- 16 bit resolution (16 bits is obtained by averaging 128 fourteen bit values)
- ~366µV per bit
- Maximum input voltage range: ±12 volts
- >0.1% precision
- SNR of 75dB with a 10VDC input voltage
- All inputs have ESD protection
- All inputs have over and undervoltage protection
- Bandwidth of each input is limited to 50Khz to minimize high frequency noise.
- All inputs are factory calibrated
- Note that a floating input will read 1.333 volts.
- Dedicated input (channel 8) for measuring power supply voltage (accurate to ±1.5%)

Functions

addr must be in the range of 0 through 7
channel must be in the range of 0 through 8

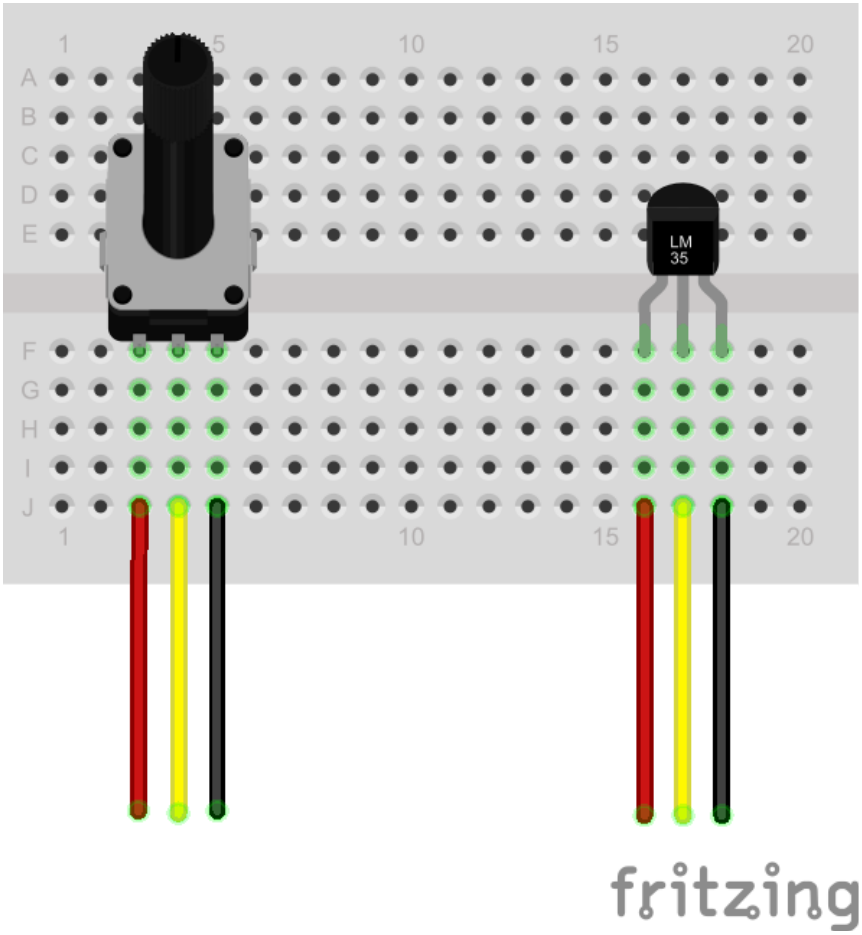
- getADC(addr, channel) - return voltage from single channel. Reading channel 8 will return the 5VDC power supply voltage.

- getADCall(addr) - return an 8 element list of floating point values with voltages from all eight input channels

Examples

Using a Potentiometer to Generate a Variable Voltage

1. Attach a 10K potentiometer to a protoboard as shown on the left side of the diagram below.
2. Route the red wire to the 5VDC terminals (terminal 10) on the Analog Input Block
3. Route the black wire to the ground terminal on the Analog Input Block (terminal 9)
4. Route the yellow wire to Analog Input 0 on the Analog Input Block (terminal 1)
5. Go into the Python interactive environment and import the DAQC2plate module by typing *import piplates.DAQC2plate as DAQC2*
6. Assuming a DAQC2plate board address of 0, type *DAQC2.getADC(0,0)* from the command prompt and look at the returned value
7. Rotate the "pot" and retype *DAQC2.getADC(0,0).* Turning the "pot" completely clockwise will return a value of around zero volts. Turning the "pot" completely counter clockwise will return a value of 5 volts.



Measure Temperature Using an LM35 Sensor

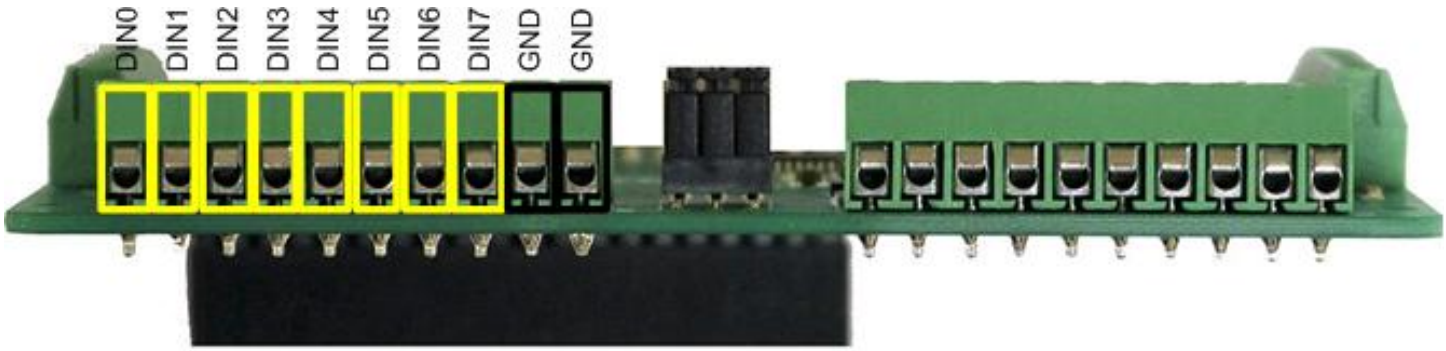
1. Attach an LM35 temperature sensor to a protoboard as shown on the right side of the diagram above.
2. Route the red wire to the 5VDC pin (terminal 9) on the Analog Input Block
3. Route the black wire to the ground terminal on the Analog Input Block (terminal 10)
4. Route the yellow wire to Analog Input 0 on the Analog Input Block (terminal 1)
5. Go into the Python interactive environment and import the DAQC2plate module by typing *import piplates.DAQC2plate as DAQC2*
6. Assuming a DAQC2plate board address of 0, type *DAQC2.getADC(0,0)* from the command prompt and look at the returned value. The LM35 will return a voltage that is proportional to the temperature in degrees Celsius: 10mV = 1 degree C. If you are performing this experiment inside then the DAQC2plate measurement should return a value of about 250mV for a temperature of 25C.
7. For more information about the LM35, go here.

Digital Inputs (DIN)

Use the digital inputs to detect simple on-off devices such as buttons, rotary encoders, and the output of another microcontroller such as an Arduino board.

Connector

Refer to the diagram at the top of this page as well as the one below to locate the Digital Input terminals:



Specifications

- All inputs have ESD protection
- All inputs have over and undervoltage protection
- All inputs have built-in pull up resistors for simple attachment of switches and outputs from optical sensors
- Compatible with 3.3 and 5V logic
- All inputs capable of triggering an interrupt to the Raspberry Pi

Functions

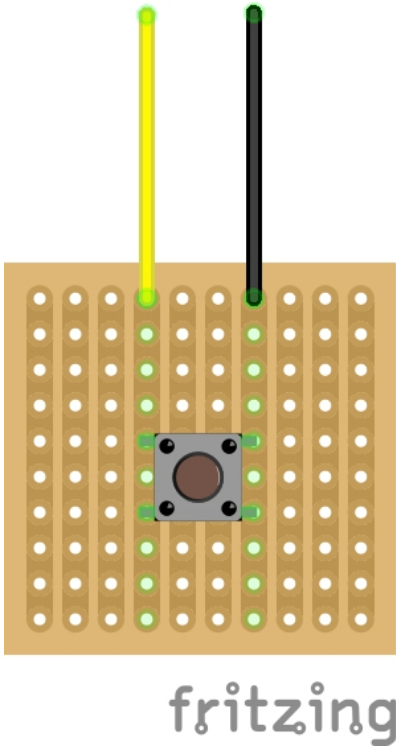
addr must be in the range of 0 through 7
bit must be in the range of 0 through 7

- getDINbit(addr,bit) - return single bit value
- getDINall(addr) - return all eight bits
- enableDINint(addr, bit, edge) - enable interrupts for an input change on the specified bit. The "edge" value can be 'r' for rising, 'f' for falling, of 'b' for both.
- disableDINint(addr,bit) - disable interrupts on the specified bit

Example: Monitor an External Button

With a built in pull up resistors, monitoring buttons and switches with the Digital Inputs of the DAQC2plate is simple:

1. Attach a normally open push button to a protoboard as shown below
2. Route the yellow wire to Digital Input 0 on the Digital Input Block (terminal 1)
3. Route the black wire to a ground terminal on the Digital Input Block (terminal 9 or 10)
4. Go into the Python interactive environment and import the DAQC2plate module by typing *import piplates.DAQC2plate as DAQC2*
5. Assuming a DAQC2plate board address of 0, type *DAQC2.getDINbit(0,0)* from the command prompt and look at the returned value. The DAQC2plate should return a value of 1 since the button is up and the input voltage is 5VDC.
6. While pushing the button, execute *DAQC2.getDINbit(0,0)* from the command prompt again. This time the DAQC2plate should have returned a value of 0 since pressing the button grounds the input.



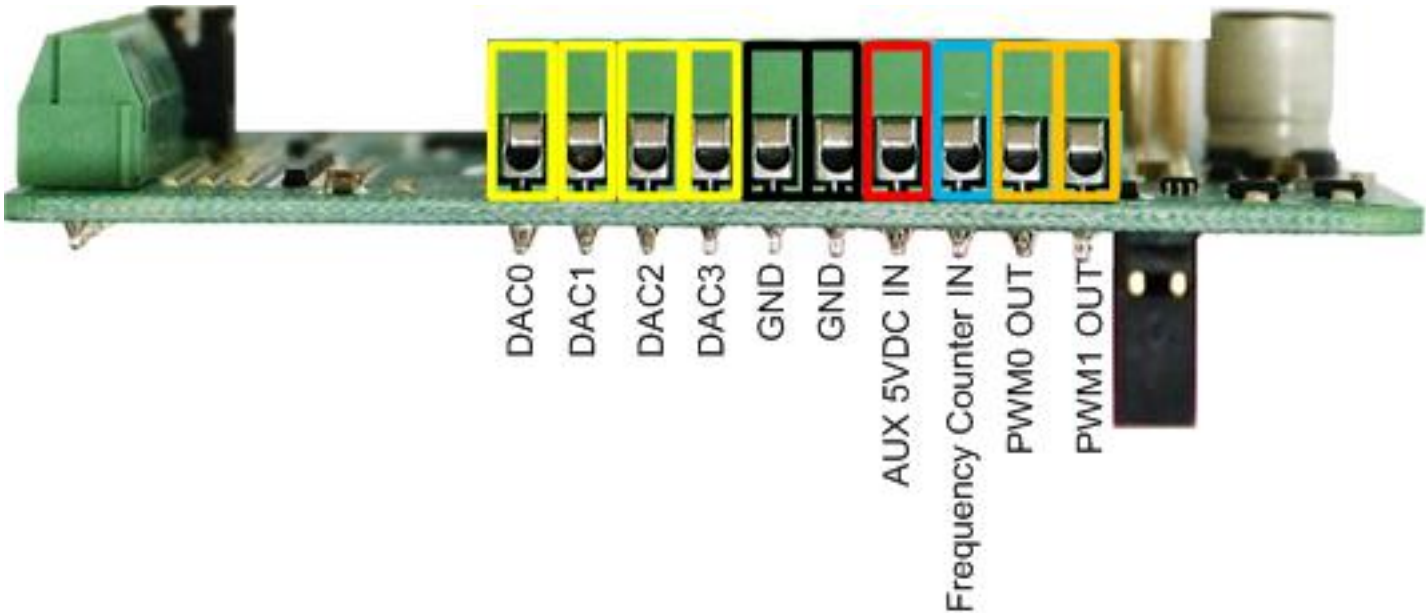
Extended Functions

The Extended Function Block provides four additional additional features to the DAQC2plate. These include:

1. Four calibrated 12 bit digital to analog converter outputs. These can be used for a number of functions and, when used in the Function Generator Mode, are fast enough to generate common waveforms such as sine, square, and triangle.
2. An auxiliary 5VDC power input. If your project requires more current than your RPi supply can provide or if you want a guaranteed clean source of power, attach your supply to this pair of connectors.
3. A frequency counter - this protected input will measure the frequency of any digital signal.
4. Two PWM outputs - these 10 bit resolution PWMs output a 5V digital waveform and can be set from 0 to 100%. Very useful for DC motor and fan speed control.

Connector

Refer to the diagram at the top of this page as well as the one below to locate the terminals for each of the Extended Functions:



Analog Outputs (DAC)

These outputs are on pins 1-4 of the Extended Function Block.

Specifications

- ESD protected
- Output swing of 0 4.095 volts
- 12 bit resolution
- Calibrated at factory for 0.2% accuracy using 3 point regression
- Resolution of 1mV
- Output noise of 110 µV rms
- Slew rate of 1 V/µs
- Nonlinearity of ±2 LSB

Functions

addr must be in the range of 0 through 7
channel must be in the range of 0 through 3

- setDAC(addr,channel,value) - set DAC output voltage from 0 to 4.095 volts.
- getDAC(addr,channel) - return current DAC output voltage. Returned value will be between 0 and 4.095 volts.

Auxiliary Power Supply

The DAQC2plate has a pair of terminals on the Extended Functions block that allow the use of a high quality power supply. Reasons you may want to use this feature include:

- You plan to stack a lot of Pi-Plates
- You need more power for your digital output loads
- You plan to operate the DAQC2plate in Motor Controller mode and want to use 5VDC stepper motors

There are a few rules when using this input:

1. It can't be used with a power supply plugged into the RPI. Well, it -can- but it won't buy you anything and we can't guarantee it will work with future versions of the Raspberry Pi.
2. Do not exceed four amps of current - the reverse bias protection circuitry will not like it.
3. Be sure to use a high quality supply with internal fusing. A UL or ETL mark is a good indicator of this.
4. If you use more than one DAQC2plate or DAQCplate in your stack, only one can be used for providing Auxiliary Power. By design, accidentally hooking up multiple supplies on multiple DAQCplate boards should not cause any damage but, only one of the supplies will be providing current.

Usage

1. Power down your stack and unplug the power supply currently attached to your Raspberry Pi.
2. Referring to the image below, loosen the screws on terminals 6 and 7 on the Extended Function Block
3. Noting the polarity, push the positive and negative wires of your power supply into the *AUX 5VDC IN* and *Ground* terminals and tighten the screws. Don't worry if you accidentally connect it backwards - the reverse bias protection circuit will prevent any damage.
4. Plug in your power supply and start coding!

PWM

These two outputs are located on pins 9 and 10 of the Extended Functions block.

Specifications

- All outputs have ESD Protection
- 10 bit resolution
- Oscillator frequency of 15.9Khz
- Output swings from 0 to 5VDC - note that this output is unregulated

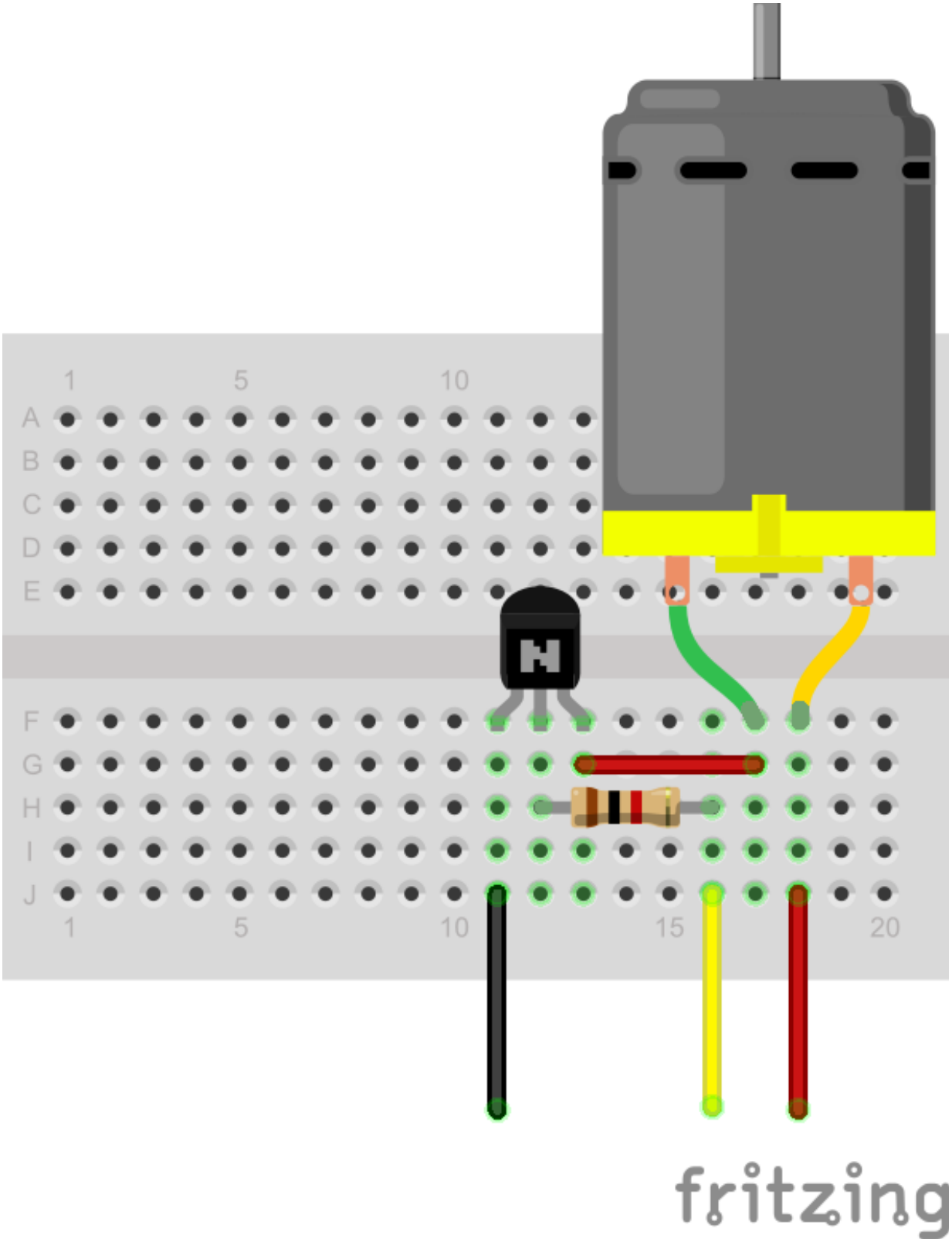
Functions

addr must be in the range of 0 through 7
channel must be a value of 0 or 1
value is a floating point number that can range from 0.0 to 100.0

- setPWM(addr,channel,value) - set PWM signal from 0 to 100%
- getPWM(addr,channel) - return current PWM setting. Returned value will be a number between 0 and 1023.

Example - Speed Control of a DC Motor

This example assumes that you have a small DC motor that operates at 5VDC max and pulls less than 1 Amp:



1. Attach the motor, a 1K resistor, a PN2222 NPN transistor, and the wiring to the protoboard as shown in the diagram above.
2. Route the red wire to the 5VDC terminal (pin 10) on the Digital Output Block
3. Route the black wire to a ground terminal on the Extended Functions block (terminal 4 or 5)

4. Route the yellow wire to PWM Output 0 on the Extended Functions block (terminal 10)
5. Go into the Python interactive environment and import the DAQC2plate module by typing *import piplates.DAQC2plate as DAQC2*
6. Assuming a DAQC2plate address of 0, type *DAQC2.setPWM(0,0,50.0)* to set the motor to approximately half speed.

Frequency Counter

The DAQC2plate has an input dedicated to measuring the frequency of any ground referenced signal with a amplitude in the range 3.3 to 5 volts. This can be used for simple bench measurements or as a tachometer for a DC motor.

Specifications

- Input Range: ground referenced signal can range from 3.3 to 5VDC.
- Measurable Frequency Range: ~ 1.00 to 150Khz
- Accuracy: ±2%
- Input Hysteresis: 1.3V

Functions

getFREQ(addr) - returns the frequency of the attached signal as an integer value.

RGB LED

Each DAQC2plate has a general purpose RGB LED. At power up, this LED is set to white to indicate a successful hardware initialization. However, you can use the LED functions below to change the color as you see fit for your application.

Functions

addr must be in the range of 0 through 7
color values is a string that can be set to the following:

1. 'off'
2. 'red'
3. 'green'
4. 'yellow'
5. 'blue'
6. 'magenta'
7. 'cyan'
8. 'white'

- setLED(addr,color) - turn on one of the LEDs in the bicolor LED package
- getLED(addr) - returns a string value with the current LED color

Interrupts

For detecting asynchronous events such as changes on the Digital Input terminals, the DAQC2plate supports the use of interrupts. An excellent tutorial on how to use interrupts in Python can be found here. Note that all Pi-Plates use the GPIO22 pin on the Raspberry Pi for generating an interrupt. If you don't choose to use interrupts, GPIO22 is available for your applications.

Functions

General

addr must be in the range of 0 through 7

- intEnable(addr) - enable interrupts from the DAQC2plate. GPIO22 will be pulled low if an enabled event occurs.
- intDisable(addr) - disables and clears all interrupts on the DAQC2plate
- getINTflags(addr) - returns 16 bit flag value and clears all INT flags

The interrupt flag register mapping looks like:

N/A	N/A	N/A	N/A	MOTOR 2 STOPPED	MOTOR 1 STOPPED	Oscope	A2D	DIN7	DIN6	DIN5	DIN4	DIN3	DIN2	DIN1	DIN0

Interrupt Flag Register

The bit definitions are:

- DIN0: set when a transition occurs on DIN0 that set by the *enableDINint* function described below.
- DIN1: set when a transition occurs on DIN1 that set by the *enableDINint* function described below.
- DIN2: set when a transition occurs on DIN2 that set by the *enableDINint* function described below.
- DIN3: set when a transition occurs on DIN3 that set by the *enableDINint* function described below.
- DIN4: set when a transition occurs on DIN4 that set by the *enableDINint* function described below.
- DIN5: set when a transition occurs on DIN5 that set by the *enableDINint* function described below.
- DIN6: set when a transition occurs on DIN6 that set by the *enableDINint* function described below.
- DIN7: set when a transition occurs on DIN7 that set by the *enableDINint* function described below.
- A2D: not used at this time
- Oscope: used in oscilloscope mode to signal the end of a sweep
- MOTOR 1 STOPPED: when enabled in the Motor Controller Mode, signals when stepper motor 1 has completed a MOVE operation
- MOTOR 2 STOPPED: when enabled in the Motor Controller Mode, signals when stepper motor 2 has completed a MOVE operation
- N/A: Not Assigned
- N/A: Not Assigned
- N/A: Not Assigned
- N/A: Not Assigned

Digital Input

The following functions control interrupt generation on the Digital inputs:
addr must be in the range of 0 through 7
bit must be a value between 0 and 7

- enableDINint(addr, bit, edge) - enable interrupts for an input change on the specified bit. The "edge" is a string value that is 'r' for rising, 'f' for falling, of 'b' for both.
- disableDINint(addr,bit) - disable interrupts on the specified bit

Special Modes

The DAQC2plate can be operated in 4 specific modes:

1. Legacy (all of the functions described in the sections above)
2. Motor Controller
3. Function Generator
4. Oscilloscope

Installing the Applications

Applications with graphical user interfaces have been developed for each of these modes. To install them, open a terminal window inside your home directory and perform the following steps:

1. If you haven't already done so: *sudo pip install Pi-Plates*
2. If you haven't already done so: *sudo pip3 install Pi-Plates*
3. Install the Python-QT routines: *sudo apt-get install python3-pyqt4*
4. Download the apps and the documentation: *sudo wget https://pi-plates.com/downloads/DAQC2apps.tar.gz*
5. Expand the archive: *tar -xzvf DAQC2apps.tar.gz*
6. After the files have been unpacked, delete the archive with: *rm DAQC2apps.tar.gz*

We know that's a lot of steps and we soon hope to have a script that will automate it with a pip command. But, when when you're done, you should have a nice set of shortcuts on the left side of your screen:

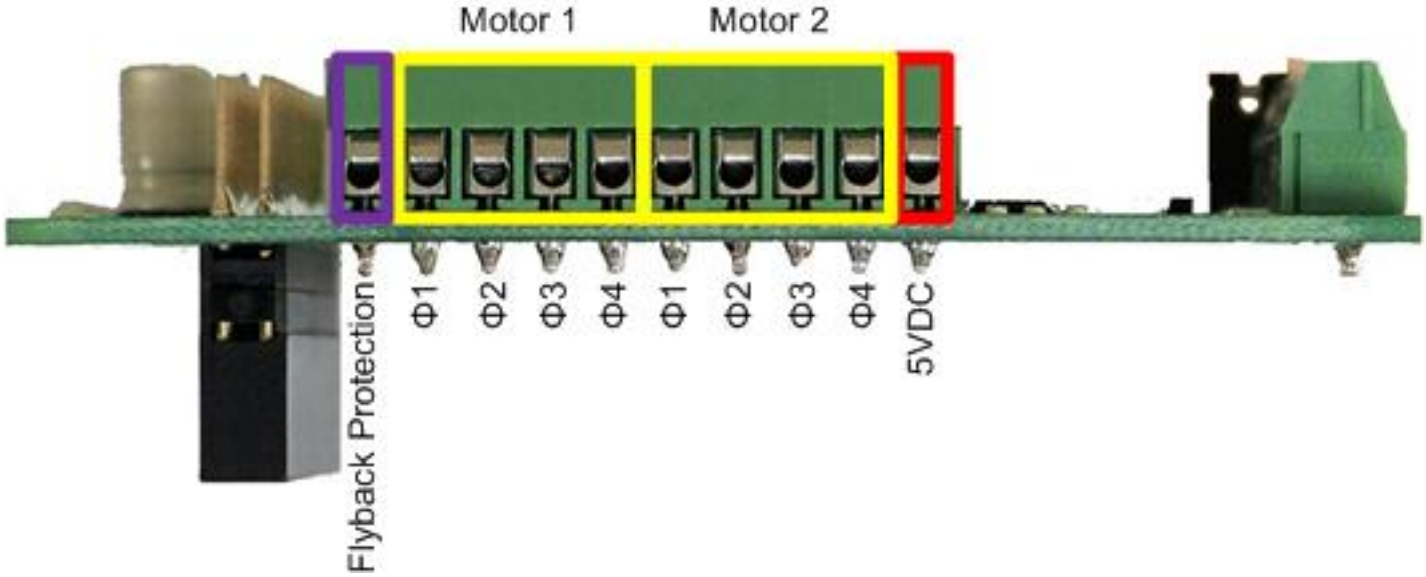
Once launched, most of the applications have a help button that brings up a manual explaining all of the controls and functions of the program.
Below, we will explain the functions behind each mode and how they are used. Keep in mind that the functions provided by these modes are not meant to replace bench top test equipment. Also note that using legacy

commands when these modes are enabled could provide unexpected results. In particular, don't combine the following:

- DOUT functions while in Motor Controller mode
- DAC functions while in Function Generator mode
- Using ADC functions while in Oscilloscope mode

Motor Controller

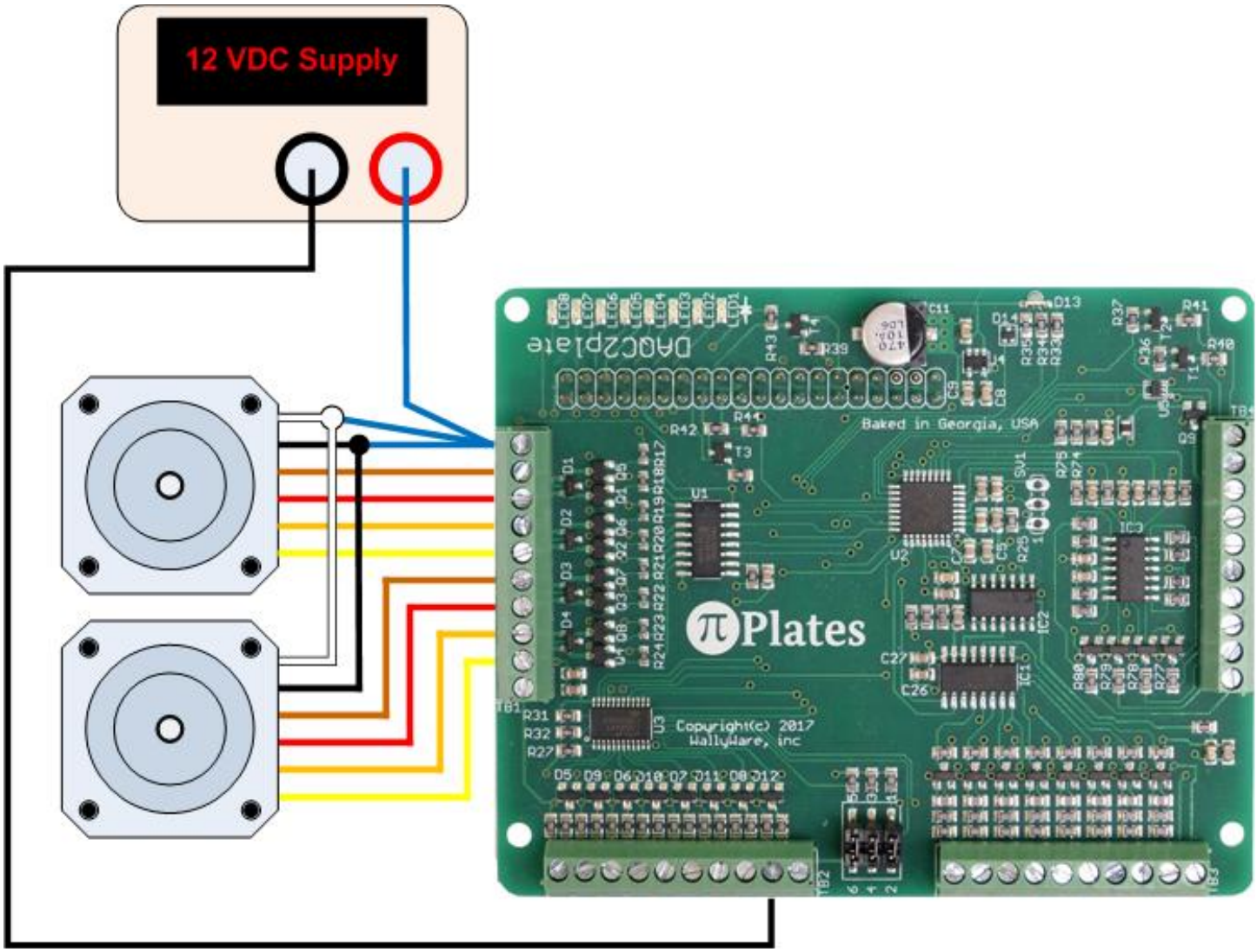
With a current handling capability of 3 amps per channel and a maximum voltage of 30 volts, the eight DOUT drivers make an ideal, dual unipolar stepper motor controller. When used in this mode, the pinout mapping changes to:



It is beyond the scope of this document to explain the theory operation of unipolar stepper motors so we suggest you learn more about them here.

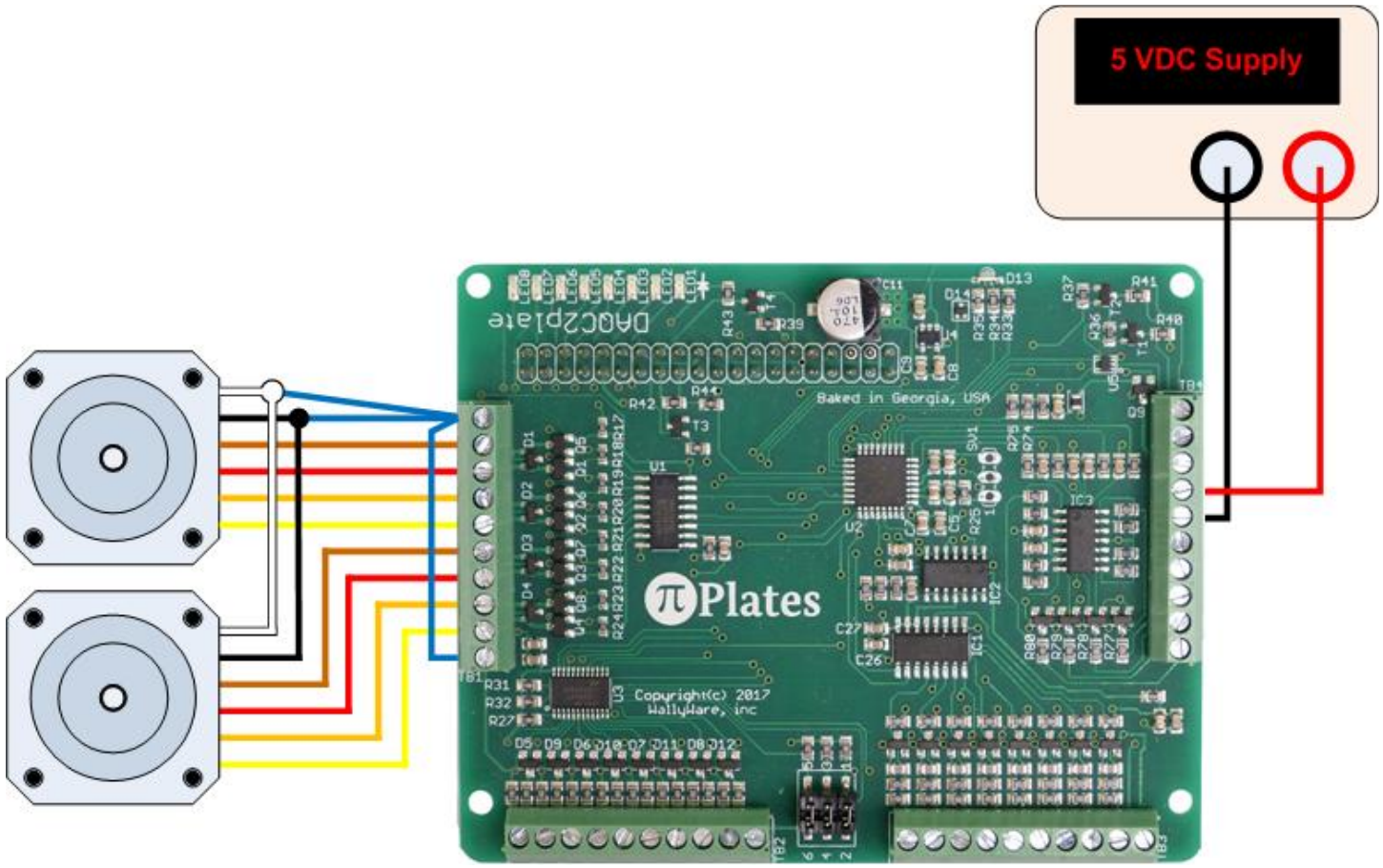
Connections

The picture below shows how to connect two 12VDC stepper motors to the DOUT header. In this case, an external supply is used and is attached to the ground connection on terminal block TB2 - pin9 and to the flyback protection pin. The power connections for each of the motors also attach to the flyback terminal.



The picture below shows how to connect two 5VDC stepper motors to the DOUT header and to use the auxiliary power supply input to drive them. In this case, and external 5VDC power supply is attached to terminals 6 and 7 of the Extended Functions connector. When connected in this manner, a power supply is not required for the Raspberry Pi. The power terminals for the motors attach to the flyback protection input, which

is also connected to 5VDC output on pin 10 via a separate wire.



Note that each of the above images show two motors with six wires each. This is not uncommon since motors of this type can be used in both unipolar and bipolar modes. Also note that the colors may not necessarily match those on your motor. Consult your motor manual for center tap wires that will attach to the power source and the wires for each of the windings.

Commands

The following arguments apply to the motor control functions:

- addr: the address of the DAQC2plate board being addressed. This can be 0-7 and is set via the address selection header described above.
- motor: selects the motor to command - can be 1 or 2.
- steps: the number of desired steps when a MOVE command is issued - must be in the range of -16383 and 16383
- dir: the direction of rotation when the JOG command is issued. This is a string argument and the valid values are 'CW' and 'CCW' for clockwise and counter clockwise.
- rate: the step rate of the motor in steps per second. Range of values is from 1 to 500
- stepsize: the motor controller supports whole steps and half steps. If your stepper motor moves 1.8° for every step, then a half step will be 0.9°. This is a shingle character argument and the valid values are 'w' for whole steps and 'h' for half.

The functions are:

- motorENABLE(addr) - enable motor mode
- motorDISABLE(addr) - disable motor mode and turn off all power going to motor windings
- motorDIR(addr,motor,dir) - set direction of selected motor
- motorRATE(addr,motor,rate,stepsize) - set step size and rate of selected motor

- motorMOVE(addr,motor,steps) - move selected motor the specified number of steps using the step size and rate specified by the motorRATE function
- motorJOG(addr,motor) - start rotating the motor in the direction set by motorDIR using the step size and rate specified by the motorRATE function
- motorSTOP(addr,motor) - stop the specified motor if it's currently in motion but leave coils energized
- motorOFF(addr,motor) - stop the specified motor if it's currently in motion and de-energize the coils. Also turns off the coils of a STOPped motor.

Function Generator

In the Function Generator mode, two of the 12 bit DAC outputs are updated 200,000 times a second to produce common waveforms such as:

- Sine
- Square
- Triangle
- Sawtooth
- Inverted Sawtooth
- Noise
- Sinc

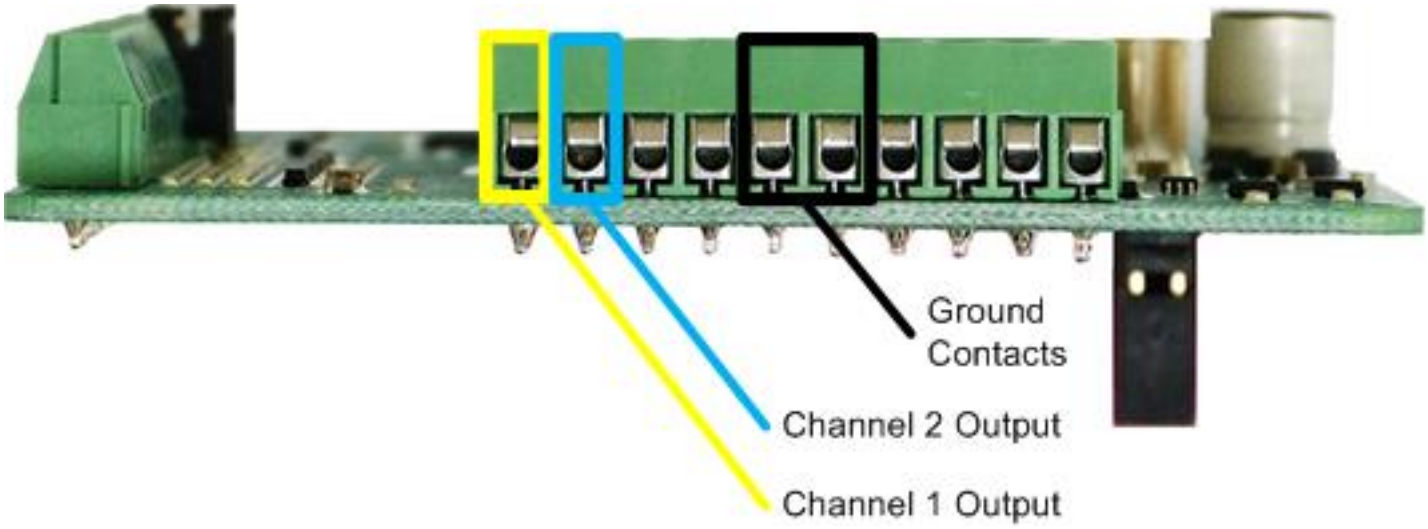
All signals are ground referenced with a maximum amplitude of 4.095 volts ±1.5%

Specifications

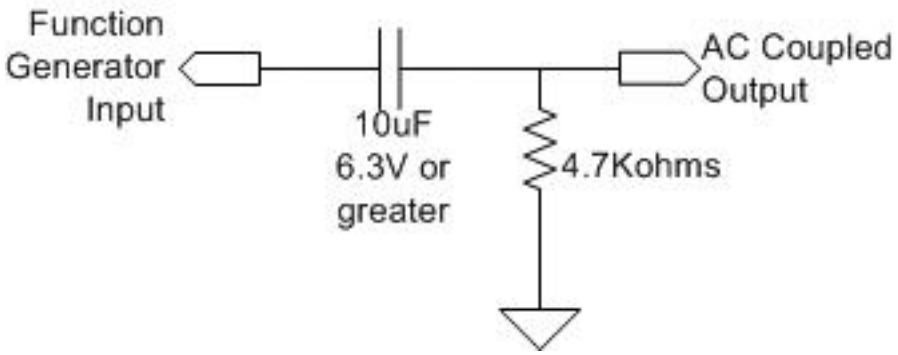
- Range: 10 to 10,000Hz
- Frequency Accuracy: ±1% typical
- Amplitude Accuracy: ±1.5%
- Resolution: 12 bits
- Number of Channels: 2
- Measured Jitter: -2µs to +12µs for square wave. ±4µs for all others
- Low Pass Filter 3dB Cutoff: 20Khz

Connections

Use the following connections to access the Function Generator outputs on the Extended Functions Block:



Use the circuit below if your application requires AC coupled signals.



Commands

The following arguments are used in the Function Generator commands:

- addr: the address of the DAQC2plate board being addressed. This can be 0-7 and is set via the address selection header described above
- chan: the output channel being referenced by the command. This can take on the value of 1 or 2.
- freq: the desired output frequency of the selected channel. This should be in the range of 10 to 10,000. Note that due to onboard filtering and the method of generation, some of the more complex waveforms will start showing distortion at frequencies greater than 5,000 Hz. The sine maintains its shape up to 10Khz before sampling artifacts become apparent.
- type: this argument sets the waveshape. Valid values are:
    - 1: sine - lookup table generated
    - 2: triangle - lookup table generated triangle wave
    - 3: square - computed
    - 4: sawtooth - lookup table generated
    - 5: inverted sawtooth - lookup table generated
    - 6: noise - computed 24 bit pseudo random output updated at 200Khz
    - 7: sinc - the classic sin(x)/x function generated with lookup table
- level: the two function generator outputs can be attenuated using the following level values:
    - 4: Full amplitude
    - 3: 1/2 amplitude
    - 2: 1/4 amplitude
    - 1: 1/8 amplitude

The list of commands is shown below:

- fgON(addr,chan) - enable function generator on selected channel. Note that this function also places the DAQCplate in Function Generator mode.
- fgOFF(addr,chan) - disable function generator on selected channel. Note that both channels have to be OFF before the DAQCplate will exit the Function Generator mode.
- fgFREQ(addr,chan,freq) - sets the desired frequency of the selected channel
- fgTYPE(addr,chan,type) - sets the waveshape of the selected channel. See the list of applicable values above.
- fgLEVEL(addr,chan,level) - sets the amplitude of the selected channel. See the list of applicable values above.

Oscilloscope

The Analog to Digital Converter (ADC) on the DAQC2plate has a sample rate of 1 million samples per second. Combine that with an input voltage range of ±12V and you've got the basic elements of a one or two channel

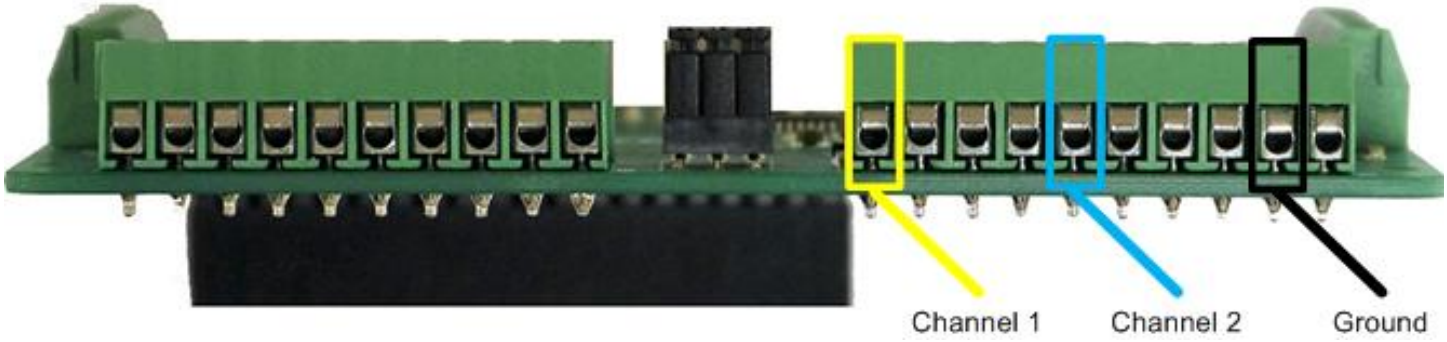
oscilloscope. There are a number of commands required to set up, trigger, capture, and download the sampled traces and so the reader is advised to install the application package and study the Python 3 code for the two channel digital oscilloscope.

Specifications

- Input voltage range: ±12V
- Input bandwidth: 370Khz
- Input Impedance: 1Mohm
- Maximum sample rate - 1 channel: 1 Msample/sec
- Maximum sample rate - 2 channel: 500 Ksample/sec
- Trace buffer size: 1024 samples
- Resolution: 12 bits
- Trigger Jitter: ±2μsec max

Connections

Use Analog Input terminal 1 for Oscilloscope Channel 1 and terminal 5 for Channel 2. Ground both signals using terminal 9:



Commands

The arguments used in the Oscilloscope commands are:

- addr: the address of the DAQC2plate board being addressed. This can be 0-7 and is set via the address selection header described above
- C1: input channel 1 enable. This can take on the value of 0 or 1.
- C2: input channel 2 enable. This can take on the value of 0 or 1.
- rate: per channel sample rate of A/D converter. The valid values in samples/second are:
    - 0: 100
    - 1: 200
    - 2: 500
    - 3: 1000
    - 4: 2000
    - 5: 5000
    - 6: 10,000
    - 7: 20,000
    - 8: 50,000
    - 9: 100,000
    - 10: 200,000
    - 11: 500,000

    - 12: 1,000,000 - only valid when a single channel is enabled
- type: trigger type - string with following values
    - 'auto'
    - 'normal'
- edge: trigger edge - string with the following values
    - 'rising'
    - 'falling'
- level: trigger level - numeric 12 bit value with a valid range of 0 to 4095. This corresponds to -12V to + 12V. Use the following equation to convert from volts to a numeric level:
    - 2048 • (1 + Vtrigger / 12)

The oscilloscope mode functions are:

- startOSC(addr) - enable oscilloscope mode on selected DAQC2plate
- stopOSC(addr) - disable oscilloscope mode on selected DAQC2plate
- setOSCchannel(addr, C1, C2) - select which channels to sample
- setOSCsweep(addr,rate) - select sample rate used during sweep
- getOSCtraces(addr) - download the sampled data from the DAQCplate
- setOSCtrigger(addr,channel,type,edge,level) - setup the trigger for the sweep.
- trigOSCnow(addr) - do immediate trigger of oscilloscope. This is typically used when no trigger occurs for a predetermined amount of time.
- runOSC(addr) - start a sweep sequence
- DAQC2.getOSCtraces(addr) - fetch the captured oscilloscope data from the DAQC2plate. This will 12-bit integer values located in two lists: trace1 and trace2.

Assuming a DAQC2plate set to address 0, the basic sequence of steps to generate a single trace on channel 1 looks like:

```
import piplates.DAQC2plate as DAQC2

DAQC2.startOSC(0)           #enable oscope
DAQC2.setOSCchannel(0,1,0)  #use channel 1

## Set up trigger:
##    Use channel 1
##    Normal trigger mode (don't collect data until trigger conditions are met)
##    Trigger on rising edge of waveform
##    Trigger at 0.0 volts
DAQC2.setOSCtrigger(0,1,'normal','rising',2048)

## setup sample rate for 10,000 samples per second
DAQC2.setOSCsweep(0,6)
DAQC2.intEnable(0)          #enable interrupts
DAQC2.runOSC(0)             #start oscope

## Wait for sweep to complete by monitoring Ocsope interrupt flag
dataReady=0
while(dataReady==0):
    if(DAQC2.GPIO.input(22)==0):
        dataReady=1
        DAQC2.getINTflags(0) #clear interrupt flags
```

```
DAQC2.getOSCtraces(0)

### print out first 1000 converted values and not the conversion from A2D integer
data to measured voltage
for i in range(1000):
    print((DAQC2.trace1[i]-2048)*12.0/2048)

DAQC2.stopOSC(0)             #turn off oscilloscope mode
```

Those are the fundamentals. For a real oscilloscope you need to visualize the data. For that, we recommend downloading our DAQC2plate application package. You can find the instructions for this at the beginning of the Special Modes section on this page. Once done, open the folder in your home directory called *Applications*. There you will see all of the python application programs including *QTscope.py*. The QT prefix indicates that it uses the QT QUI package.