DAQCplate Users Guide

Contents

Overview

This page provides the major features of the Pi-Plates DAQCplate Data Acquisition and Control board. This page can be considered to be a “living” document which means that it will see continual updates and corrections. All of the code examples below are written for Python 2.7. However, one of our customers, Jeremy Deters, has coded an experimental C# library along with a simple dashboard app for Windows 10 IOT. The two archives are available here and here. In addition, Nahuel Lofeudo has written a Java library for the DAQCplate and the RELAYplate. These can be downloaded from GitHub here.

http://pi-plates.com/downloads/PiPlates%200.9.3.zip

http://pi-plates.com/downloads/DAQCdashboard.zip

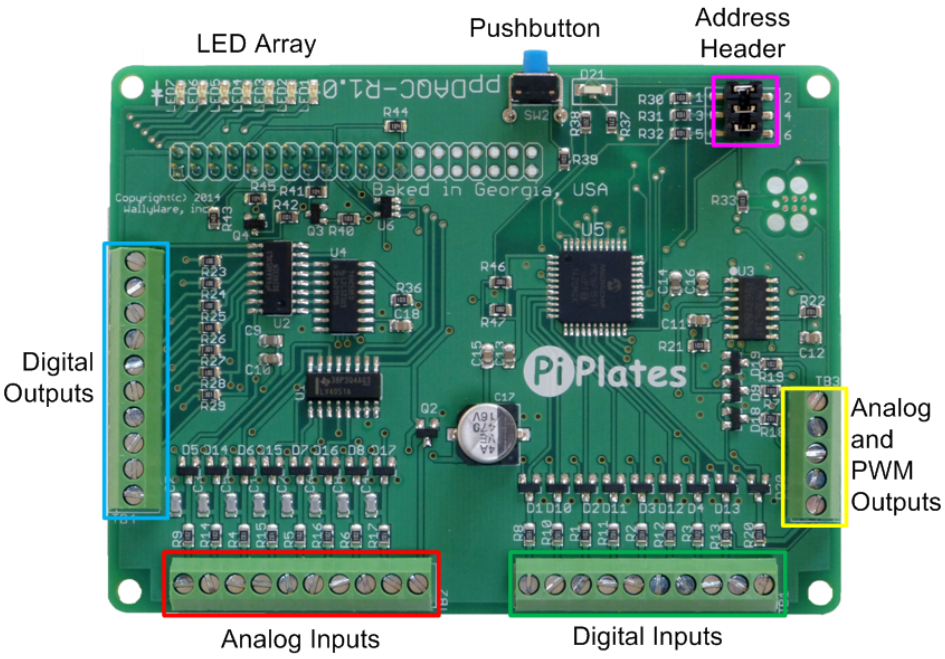
https://github.com/nahuellofeudo/piplates

The following definitions apply in the Function descriptions below:

- addr: the address of the DAQCplate board being addressed. Can be 0-7 and is set via the address selection header described below.
- bit: the channel or terminal being controlled by the command.

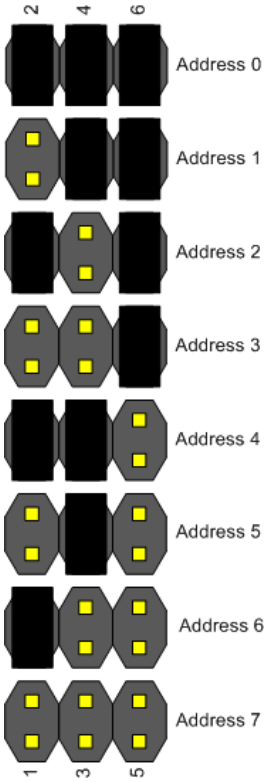
Board Layout

The terminal blocks and their functions are shown below. Note that you can also look on the bottom of the board and see the function of each terminal.



Address Selection Header

Up to eight DAQCplates 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 DAQCplate is set to address zero. The address is set by positioning jumpers on the small, six pin header in the upper right corner 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.

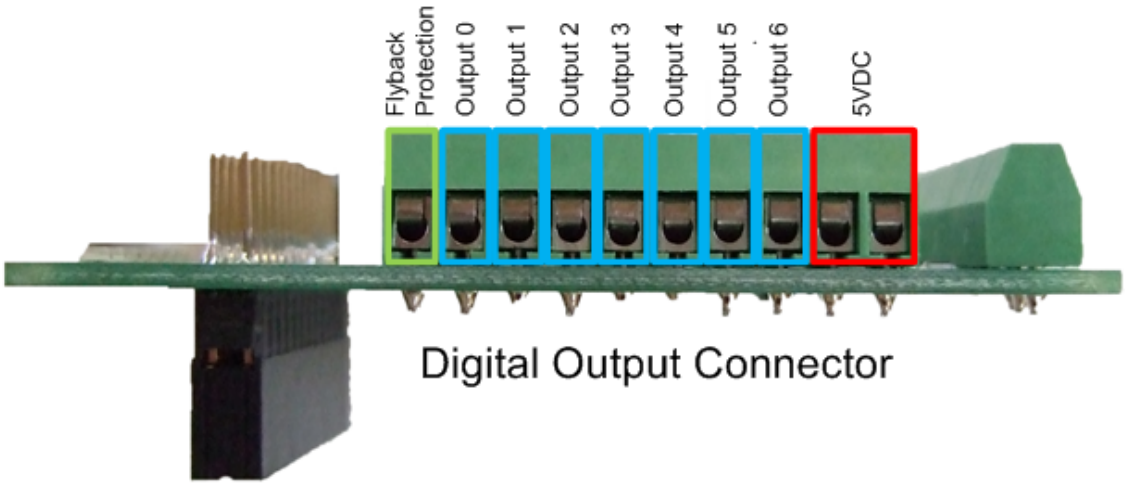
Digital Outputs (DOUT)

The digital output connector provides seven open collector outputs, a pair of 5VDC outputs 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.

https://en.wikipedia.org/wiki/Open_collector

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 7 Digital Outputs is an open collector Darlington transistor
- Maximum sink current is 350mA
- Maximum load voltage is 12VDC
- On voltage is typically 1.1V with a load current of 200mA
- 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 pins 9 and 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 (~90mA for each DAQCplate with all LEDs on) and any components you connect to a DOUT terminal.

Functions

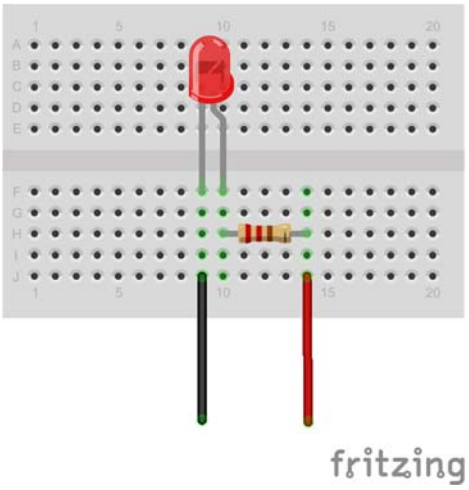
addr must be in the range of 0 through 7

bit must be in the range of 0 through 6

- setDOUTbit(addr, bit) – set single bit
- clrDOUTbit(addr, bit) – clear single bit
- toggleDOUTbit(addr, bit) – toggle a single bit
- setDOUTall(addr,byte) – control all seven bits at once. The byte value must be in the range of 0 through 127.

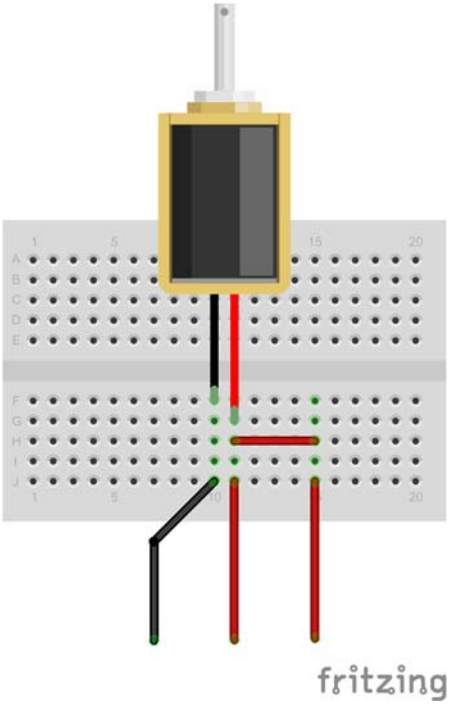
Examples

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 a 5VDC terminal on the DAQCplate digital output terminal block. Route a black wire from the cathode of the LED to Output 0 of the terminal block. Assuming the DAQCplate board is at address 0, type DAQC.setDOUTbit(0,0) to turn the LED on. Type DAQC.clrDOUTbit(0,0) to turn off the LED.

Driving an Inductive Load



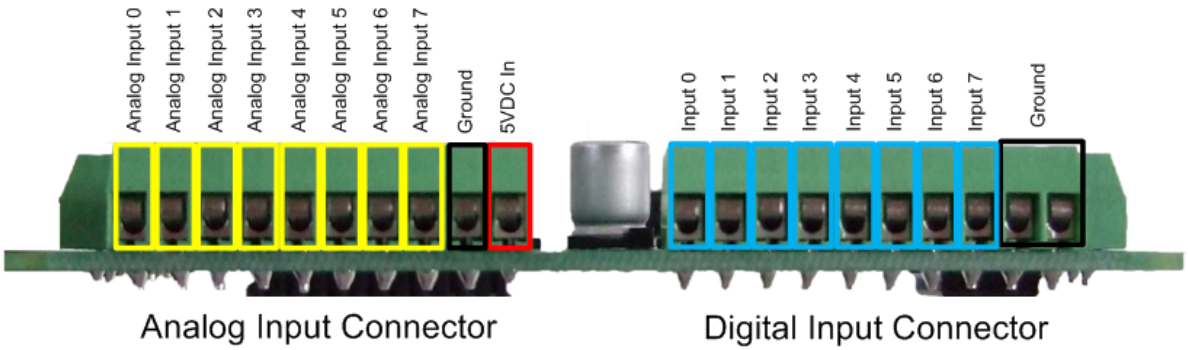
1. Use a simple breadboard to build up this 5V solenoid circuit.
2. Connect the red wire from the solenoid to a pair of red wires that route from the breadboard.
3. Attach one of these to a 5VDC terminal on the DAQCplate digital output terminal block.
4. Connect the other red wire to the Flyback Protection terminal. Route a black wire to Output 0 of the terminal block.
5. Assuming the DAQCplate board is at address 0, type DAQC.setDOUTbit(0,0) to turn the solenoid on.
6. Type DAQC.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

- 10 bit resolution
- 4mV per bit
- Maximum input voltage: 4.097 volts
- Minimum measurable voltage: ~65mV
- 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.

Functions

addr must be in the range of 0 through 7

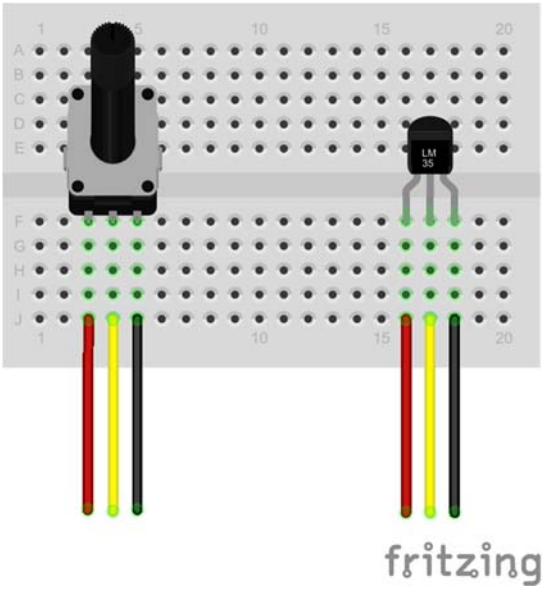
channel must be in the range of 0 through 8

- getADC(addr, channel) – return voltage from single channel. Reading channel 8 will return the 5VDC power supply voltage.
- getADCall(addr) – return voltage from all 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 one of the 5VDC terminals (terminal 9 or 10) on the Digital Output 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 DAQCplate module by typing *import piplates.DAQCplate as DAQC*
6. Assuming a DAQCplate board address of 0, type *DAQC.getADC(0,0)* from the command prompt and look at the returned value
7. Rotate the "pot" and retype *DAQC.getADC(0,0).* Turning the "pot" completely clockwise will return a value of around zero volts. Turning the "pot" almost completely counter clockwise will return a value of 4.097 volts.



Measure Temperature Using an LM35 Sensor

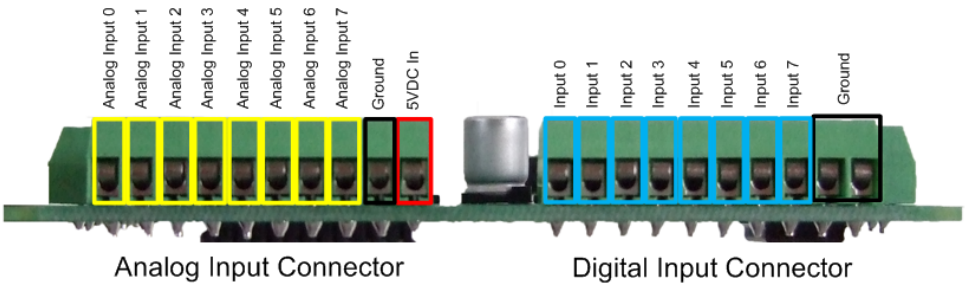
1. Attach an LM35 temperature sensor to a protoboard as shown on the right side of the diagram above.
2. Route the red wire to one of the 5VDC terminals (terminal 9 or 10) on the Digital Output Block
3. Route the black wire to the ground terminal on the Analog Input Block (terminal 9)
4. Route the yellow wire to Analog Input 0 on the Analog Input Block (terminal 1)
5. Go into the Python interactive environment and import the DAQCplate module by typing *import piplates.DAQCplate as DAQC*
6. Assuming a DAQCplate board address of 0, type *DAQC.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 DAQCplate measurement should return a value of about 250mV for a temperature of 25C.
7. For more information about the LM35, go here. http://www.ti.com/lit/ds/symlink/lm35.pdf

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. These inputs can also be used to measure temperature when connected to a DS18B20 1-wire sensor.

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
- Compatible with 3.3 and 5V logic
- All inputs capable of triggering an interrupt to the Raspberry Pi
- Special function to directly read DS18B20 1-wire temperature sensor

Functions

addr must be in the range of 0 through 7

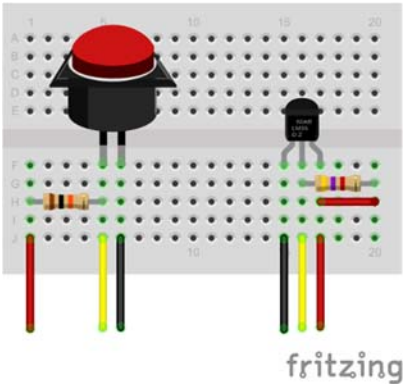
bit must be in the range of 0 through 7

- 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
- getTEMP(addr,bit,scale) – a special function to read a DS18B20 temp sensor attached to the specified bit input. Scale values can be ‘c’, ‘f’, or ‘k’ for Celsius, Fahrenheit, or Kelvin. Note that this function takes about 1 second to complete

Examples

Monitor an External Button

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

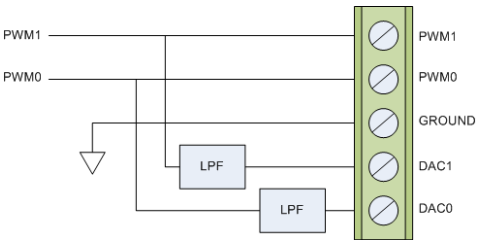


Measure Temperature with a DS18B20 Temperature Sensor

1. Attach a DS18B20 temparature sensor and a 4.7K ohm resistor to a protoboard as shown on the right side of the diagram above
2. Route the red wire to one of the 5VDC terminals (terminal 9 or 10) on the Digital Output Block
3. Route the black wire to the ground terminal on the Digital Input Block (terminal 9)
4. Route the yellow wire to Digital Input 0 on the Digital Input Block (terminal 1)
5. Go into the Python interactive environment and import the DAQCplate module by typing *import piplates.DAQCplate as DAQC*
6. Assuming a DAQCplate address of 0, type *DAQC.getTEMP(0,0,'f')* from the command prompt and look at the returned value. The DAQCplate should return the temperature in degrees Fahrenheit.
7. Repeat the above but type *DAQC.getTEMP(0,0,'k')* instead. The DAQCplate should return the temperature in degrees Kelvin.

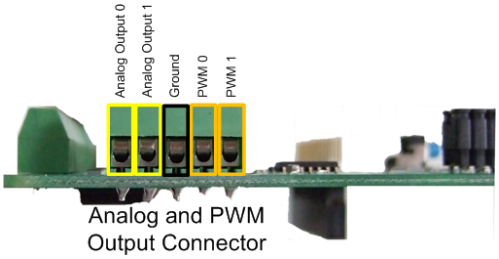
Analog (DAC) and Pulse Width Modulator (PWM) Outputs

The DAQCplate board provides two linked Pulse Width Modulator (PWM) and Digital to Analog Outputs. Each PWM can be independently set to any value between 0 and 100% The analog outputs are simply the PWM signals that have passed through low pass filters. Therefore, changing the value of PWM0 will alter the value of DAC0 and vice versa. The block diagram below shows this relationship:



Connector

Refer to the diagram at the top of this page as well as the one below to locate the Analog Out and and Pulse Width Modulator terminals:



Specifications

PWM

- All outputs have ESD Protection
- 10 bit resolution
- Oscillator frequency of 15.9Khz

Analog Outputs

- All outputs have ESD protection
- Voltage can be set from 0 to 4.097 volts
- Derived from passing PWM signal through a 15.9Hz filter.

Functions

addr must be in the range of 0 through 7

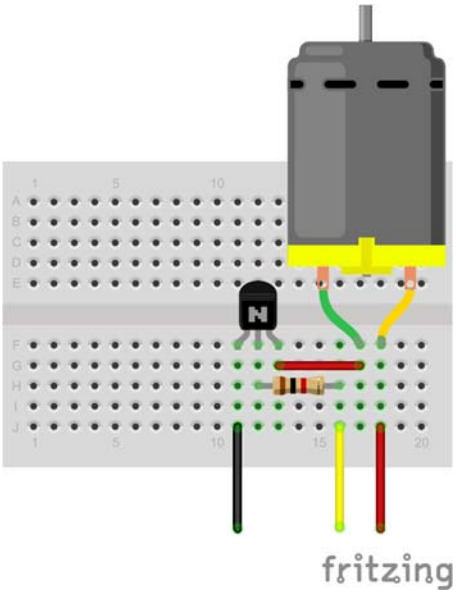
channel must be a value of 0 or 1.

- setPWM(addr,channel,value) – set PWM signal from 0 to 1023 (0 to 100%)
- getPWM(addr,channel) – return current PWM setting. Returned value will be a number between 0 and 1023.
- setDAC(addr,channel,value) – set DAC output voltage from 0 to 4.097 volts.
- getDAC(addr,channel) – return current DAC output voltage. Returned value will be between 0 and 4.097 volts.
- calDAC(addr) – calibrate the DAC outputs. Use this function before setting the DAC value if you are unsure about the quality of your power supply.

Example

Speed Control of a DC Motor

This example assume 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. https://www.fairchildsemi.com/datasheets/PN/PN2222A.pdf
2. Route the red wire to one of the 5VDC terminals (terminal 9 or 10) on the Digital Output Block
3. Route the black wire to the ground terminal on the DAC/PWM terminal block (terminal 3)
4. Route the yellow wire to PWM Output 0 on the DAC/PWM terminal block (terminal 4)
5. Go into the Python interactive environment and import the DAQCplate module by typing *import piplates.DAQCplate as DAQC*
6. Assuming a DAQCplate address of 0, type *DAQC.setPWM(0,0,'512')* to set the motor to approximately half speed.

Switch and Bicolor LED

Each DAQCplate board has a small onboard switch and general purpose bicolor status LED. At power up, the LED is set to green to indicate a successful hardware initialization. However, you can use the LED functions below to change it as you see fit for your application.

When using the power control feature of the DAQCplate, pressing the switch will ground the GPIO18 pin of the Raspberry Pi for approximately 5 seconds while turning on both the red and green LEDs in the bicolor package. The DAQCplate will then wait for approximately 15 seconds and then turn off the green LED leaving the red LED on. This indicates that the system is in the off state and that it is safe to remove power. For more information on how to set up this feature, jump to the section labeled Using the Switch to Power Down Your Raspberry Pi below.

Functions

Bicolor LED

addr must be in the range of 0 through 7

color values can be 0 for the red LED or 1 for the green.

- setLED(addr,color) – turn on one of the LEDs in the bicolor LED package
- clrLED(addr,color) – turn off one of the LEDs in the bicolor LED package

Switch

addr must be in the range of 0 through 7

- getSWstate(addr) – returns current state of on board switch. A value of 1 is returned when the switch is up and a value of 0 is returned when it's down.
- enableSWpower(addr) – pushing button on ppGPIO will short RPI GPIO23 to GND and then remove 5VDC 45 seconds later. Note that this setting is saved in nonvolatile memory and only has to be performed once. Be sure that you have downloaded and installed the ppPOWER service described above to take full advantage of this feature.
- disableSWpower(addr) – disables the above. Note that this setting is stored in nonvolatile memory and only has to be performed once.
- enableSWint(addr) – allows the switch to generate an interrupt when pressed. Global interrupts must be enabled before using this function.
- disableSWint(addr) – blocks switch on DAQCplate board from generating an interrupt.

Using the Switch to Power Down Your Raspberry Pi

Setup

Setting up your DAQCplate and Raspberry Pi for a graceful shutdown using the push button on the DAQCplate requires the following steps:

Download, Install, and Run, the ppPower Service

Using the power down feature requires a program running in the background on your Raspberry Pi that is monitoring GPIO18:

1. From the command line in your home directory execute: *wget http://pi-plates.com/downloads/ppPower-1.00.tar.gz*
2. Unpack by executing: *tar -xzf ppPower-1.00.tar.gz*
3. Enter the ppPower directory: *cd ppPower-1.00*

4. From the command prompt execute *sudo ./install-ppPower*
5. The service is now installed and never has to be run again
6. Other functions in the ppPower-1.00 directory include:
    1. *sudo ./disable-ppPower* leaves the software installed but disables auto start-up feature
    2. If auto startup is disabled you can manually start the service with *sudo ./start-ppPower*
    3. *sudo ./stop-ppPower* will stop the service at anytime. This feature is useful when debugging your software
    4. *sudo .remove-ppPowe*r completely removes the power control service from your raspberry Pi

Reboot your Raspberry Pi

The ppPower service requires a reboot before installation is complete. To do that execute *sudo reboot* from command line. After you've logged back into your Raspberry Pi proceed to the next step.

Setup the Push Button on the DAQCplate to Act as a Power Switch

This is the easy part:

1. Enter the Python interpreter with the *sudo python* statement at the command prompt
2. Once in the interpreter, load the DAQCplate module with the statement *import piplates.DAQCplate as DAQC*
3. Then, assuming an address of 0, tell the DAQCplate that you want to use the push button as a power switch with the statement *DAQC.enableSWpower(0)*
4. That's it. The DAQCplate will now tell the RPI to shut down whenever your press the button. And, since this setting is saved in the flash memory of the DAQCplate, you never have to run it again.
5. Exit the interpreter by pressing <CNTL>-D

Operation

When you're ready to shut down your Raspberry Pi for the day simply push the button on the DAQCplate. The bicolor LED on the programmed DAQCplate will turn orange for about 15 seconds to indicate that the Raspberry Pi is shutting down. Once it turns red, it is safe to remove the power to your RPI.

A few notes:

1. If you have multiple DAQCplates in your stack, only one of them has to be programmed to perform this function. The buttons on the other Pi-Plates are still available for your applications
2. This service does require that you dedicate the GPIO18 pin on your Raspberry Pi for monitoring the shutdown signal.

3. Pressing the button after the LED turns red will just turn the bicolor LED green again. It will NOT turn your Raspberry Pi back on – that requires a power cycle

Interrupts

For detecting asynchronous events such as button presses and changes on the Digital Input terminals, the DAQCplate supports the use of interrupts. An excellent tutorial on how to use interrupts in Python can be found here. Note that Pi-Plates all 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.

http://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio

Functions

General

addr must be in the range of 0 through 7

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

The interrupt flag register mapping looks like:

0	0	0	0	0	0	0	SW	In7	In6	In5	In4	In3	In2	In1	In0

Interrupt Flag Register

Digital Input

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" value can be 'r' for rising, 'f' for falling, of 'b' for both.
- disableDINint(addr,bit) – disable interrupts on the specified bit

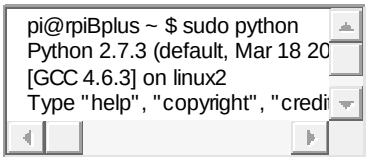
Switch

addr must be in the range of 0 through 7

- enableSWint(addr) – allows the switch to generate an interrupt when pressed. Global interrupts must be enabled before using this function.
- disableSWint(addr) – blocks switch on DAQCplate from generating an interrupt.

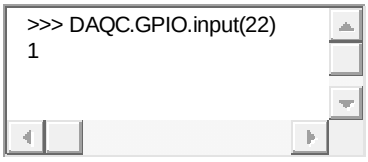
Example

Follow the steps below to get a basic idea of how to set up interrupts using a DAQCplate. To really take advantage of them in Python, visit the tutorial mentioned above. Let's start by setting things up to generate an interrupt if the button is pushed:



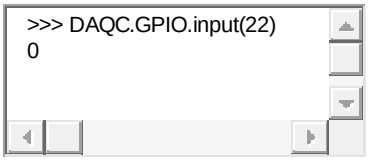

```
1  pi@rpiBplus ~ $ sudo python
2  Python 2.7.3 (default, Mar 18 2014, 05:13:23)
3  [GCC 4.6.3] on linux2
4  Type "help", "copyright", "credits" or "license" for more information.
5  >>> import piplates.DAQCplate as DAQC
6  >>> DAQC.enableSWint(0)
7  >>> DAQC.intEnable(0)
```

Verify that the INT line is high:




```
1  >>> DAQC.GPIO.input(22)
2  1
```

Now push the button and repeat the above:



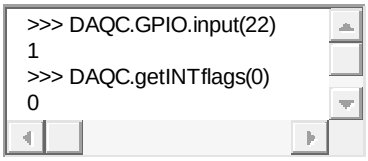

```
1  >>> DAQC.GPIO.input(22)
2  0
```

The pushbutton has generated an interrupt! Lets confirm this and the clear the interrupt by reading the Interrupt Flag Register:


```
>>> DAQC.getINTflags(0)
256
```

Reading the Interrupt Flag Register indicates that bit 8 is high which maps to the push button.

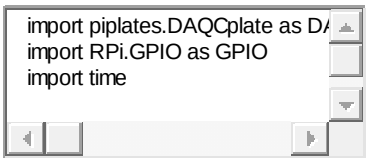
Let's confirm that we've cleared the interrupt and disable them now that we're done:




```
>>> DAQC.GPIO.input(22)
1
>>> DAQC.getINTflags(0)
0
>>> DAQC.intDisable(0)
>>>
```

And that's it. Similar functionality is available for the Digital Inputs.

Here's a second example. Open a new file called InterruptDemo.py with nano then copy and paste the following code in:




```
import piplates.DAQCplate as DAQC
import RPi.GPIO as GPIO
import time

def buttonINT(channel):
    time.sleep(.01)            #debounce button before clearing interrupt
    DAQC.getINTflags(0)       #clear the interrupt
    print " ____   _____        ___ "
    print "|  _ \ / _ \ \      / / |"
    print "| |_) | | | \ \ /\ / /| |"
    print "|  __/| |_| |\ V  V / |_|"
    print "|_|    \___/  \_/\_/  (_)"

GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(22, GPIO.FALLING, callback=buttonINT)
DAQC.enableSWint(0)   #enable the push button interrupt
DAQC.intEnable(0)     #enable global interrupts

```

```
try:
    while(1):                   #Cycle the LEDs on in the foreground
        for i in range(0,7):
            DAQC.setDOUTbit(0,i)  #turn on a single LED
            time.sleep(.1)          #wait for 100msec
            DAQC.clrDOUTbit(0,i)  #turn off the LED and go to the next one
            #wait for it...
except KeyboardInterrupt:
    GPIO.cleanup()       # clean up GPIO on CTRL+C exit
```

Once you're done, save it and exit from nano. From the command prompt type *sudo python InterruptDemo.py*

In the foreground, the above program is cycling the LEDs connected to the DOUT lines. But, if a button press occurs, GPIO22 is pulled low by the DAQCplate. When the Raspberry Pi detects this, it jumps to the buttonINT function. There it waits 10msec for a button debounce, clears the interrupt by reading the Interrupt Flag Register, and then prints out a message. When the function is complete, the RPI returns to cycling the LEDs.

Watch the lights. Push the button. POW!

Hybrid Functions

Hybrid functions use multiple signals and terminals to perform an operation.

Distance Measurement with the HC-SR04

The HC-SR04 is a low cost and ubiquitous ultrasonic rangefinder that, like a bat, uses reflected sounds to measure distance. They can be used for robotics, measuring the amount of water in a tank, or reporting the distance between your car and surrounding obstacles to name just a few applications. They are also readily available and can be purchased from a number of vendors for a very reasonable price. The interface requires a Digital Output from the DAQCplate to act as a trigger and a Digital Input to measure the acoustical flight time. Since the device requires both an output signal and an input, this is classified as a hybrid function.

Connection

Each HC-SR04 has four pins:

1. Vcc
2. Trigger
3. Echo
4. Ground

Connect Vcc to terminal 9 or 10 on the Digital Output block. Connect Ground to terminal 9 or 10 on the Digital Input . For Trigger and Echo, connection between an HC-SR04 and the DAQCplate is done via Input/Output pairs. If you choose to measure distance on channel 0 then you must connect the Trigger to DOUT0 and Echo to DIN0. If instead you want to used channel 6 then you must connect the Trigger to DOUT6 and Echo to DIN6. Up to seven HC-SR04s can be connected to a single DAQCplate.

Functions

addr must be in the range of 0 to 7

channel must be in the range of 0 to 6

units controls what the units are of the returned value:

'i' returns distance in inches

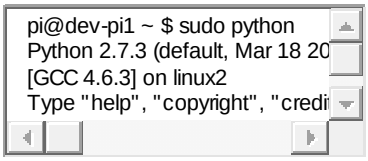
'c' returns distance in centimeters

- getRANGE(addr,channel,units) – returns distance from sensor in centimeters or inches depending on the value of *units*

Example

1. Connect Ground to terminal 10 on the Digital Input block (it's important to attach ground first)
2. Connect Vcc (5VDC) to terminal 10 on the Digital Output block.
3. Connect Trig to terminal 2 on the Digital Output Block
4. Connect Echo to terminal 1 on the Digital Input Block
5. Position the sensor so that it is pointing at something nearby like your computer monitor

Follow the steps below and measure distance:




```
1   pi@dev-pi1 ~ $ sudo python
2   Python 2.7.3 (default, Mar 18 2014, 05:13:23)
3   [GCC 4.6.3] on linux2
4   Type "help", "copyright", "credits" or "license" for more information.
5   >>> import piplates.DAQCplate as DAQC
6   >>> DAQC.getRANGE(0,0,'i')
7   10.23
8   >>> DAQCp.getRANGE(0,0,'c')
9   25.61
10  >>>
```

The above example indicates that my HC-SR04 is just over ten inches or 25.6 centimeters from my monitor. How cool is that?

Using an Auxiliary Power Supply

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

- You want better precision for your Digital to Analog converter output – cheap power supplies tend to fluctuate with loads and therefore are poor candidates for precise outputs
- You plan to stack a lot of Pi-Plates
- You need more power for your digital output loads

There are a couple of 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 three 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 mark is a good indicator of this.
4. If you use more than one DAQCplate board 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 9 and 10 of the Analog Input Connector

3. Noting the polarity, push the positive and negative wires of your power supply into the *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!