COOKING WITH TEAM 279

ANALOG SIGNALS WITH MCP3002/MCP3008 ADC

The RPi does not have analog input pins. To read analog signals, and Analog to Digital Converter (ADC) should be used. The MCP3002 and MCP3008 are two examples of commonly used, inexpensive ADCs. These connect to the RPI via the SPI protocol. So this worksheet will also introduce how to read SPI connected devices, but is not focusing on understanding the details of SPI.

In this exercise, you will build a circuit with an analog circuit and read the values from the ADC via SPI using the SPIDEV library.

GOALS

- Become familiar with the basics of how the Raspberry Pi can use analog signals
- Learn how SPI works at a high level
- Learn how to use an ADC

PARTS NEEDED

- Raspberry Pi
- Breadboard
- MCP3002, MCP3004, or MCP3008 Analog to Digital Converter
- 220Ω to 1k Ω Resistors
- An analog signal source (potentiometer, photoresistor, temperature sensor, etc…)
- Jumper Wires

REFERENCES AND MORE INFO

1. MCP3002 Datasheet - http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/MCP3002.pdf
2. MCP3004/MCP3008 Datasheet - https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf
3. Nice Overview of SPI and the MCP3008 - https://rheingoldheavy.com/mcp3008-tutorial-01-functionality-overview/
4. Nice overview of SPI on the RPI - http://www.petervis.com/Raspberry_PI/Analog_to_Digital_Converter/Raspberry_Pi_Analog_to_Digital_Converter.html
5. SPIDEV Github - https://github.com/doceme/py-spidev

ENABLE SPI

The ADCs we are going to use, talk to the RPi via SPI (Serial Peripheral Interface). The Raspberry PI SPI interface allows 1 or 2 SPI devices to connect to it maximum. It then uses a simple serial communication mechanism to communicate with the devices to send or receive values.

The RPi will be a master device, and the ADC we use will be a slave.

There are two ways to connect to SPI devices with the Pi

- Using the SPI interface that is included in the RPI hardware (makes use of the specific GPIO pins that have SPI as an alternate function)
- Using software controlled SPI that can use any GPIO pins we designate

We will use the hardware method in this worksheet to show how to enable it, and use the designated pins on the GPIO header.

The advantage of the software defined SPI is that it can all be reconfigured on the fly without having to reboot after enabling/disabling the hardware SPI interface, and use any of the GPIO pins we would desire (also works on older RPI's that don't have hardware based SPI).

By default, the hardware SPI interface is disabled on the RPi 3. To enable it, we must modify the boot config

1. Run the Raspberry Pi configuration utility (Menu -> Preferences -> Raspberry Pi Configuration)
2. Under interfaces select "Enable" next to SPI
3. Hit OK, and reboot

When you are finished, if you will not be using the SPI interface, you may want to disable the SPI interface to free the GPIO pins up. Otherwise, remember to avoid those GPIO pins when working on other projects.

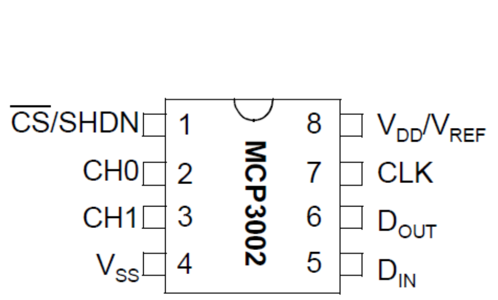
LOAD THE SPIDEV PYTHON LIBRARY

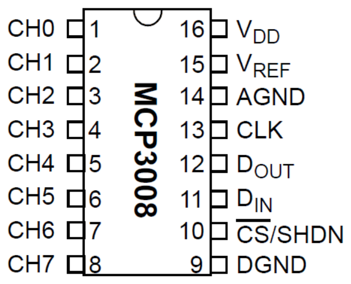
There are various libraries for communicating with SPI. One of the most popular is the SPIDEV library for Python. The SPIDEV Python packages for using SPI must be downloaded and installed. To do so, execute the following commands from a terminal window. This will install it for both python 2 and 3:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-dev python3-dev
cd ~
git clone https://github.com/doceme/py-spidev.git
cd py-spidev
sudo python setup.py install
sudo python3 setup.py install
```

CONNECT THE MCP3002 OR MCP3008

The pin out of the MCP3002 and MCP3008 are show below. If you have a 3004, check the datasheet for the pinouts, and follow the directions listed for the 3008 below.





CH0 -> CH7 connect to the analog voltage source from the sensor.

For MCP3002, connect the wires as follows:

MCP3002 Pin	Raspberry Pi 3 Pin
8 (V_{DD}/V_{ref})	3.3V
4 (V_{SS})	GND
5 (D_{IN})	MOSI (GPIO 10)
6 (D_{OUT})	MISO (GPIO 9)
7 (CLK)	SCLK (or SCK) (GPIO 11)
1 (CS/SHDN)	CE0 (GPIO 8) - if there was a second SPI device, it would go to CE1 (GPIO 7)
CH0 and CH1	Do not connect to RPi (the sensors connect to these)

MCP3008, connect the wires as follows:

MCP3008 Pin	Raspberry Pi 3 Pin
V_{DD}	3.3V
V_{REF}	3.3V
A_{GND}	GND
D_{GND}	GND
CLK	SCKL (or SCK) (GPIO 11)
D_{OUT}	MISO (GPIO 9)
D_{IN}	MOSI (GPIO 10)
CS/SHDN	CE0 (GPIO 8) - if there was a second SPI device, it would go to CE1 (GPIO 7)
CH0 -> CH7	Do not connect to RPi (the sensors connect to these)

A NOTE ABOUT ANALOG GROUND, DIGITAL GROUND, AND NOISE

The “Layout Considerations” section of the datasheet for the MCP3004/3008 explains how the AGND and DGND pins are connected inside the chip, and how to tie them back to the actual ground.

This worksheet assumes you are simply using the RPi’s 3.3V or 5V supply pins, in which case all GND pins are common. If you are measuring a circuit that has a separate supply, read through this section of the datasheet to understand how to connect them GND pins.

On a similar note, the datasheet says to attach a 1 µF capacitor between the VDD pin and DGND. We aren’t worrying about that in this worksheet, but it will improve readings in real world usage.

HOW DOES IT WORK

The ADC essentially times how fast it takes for a capacitor to fill. A capacitor holds an electric charge. The time required for that charge to fill is based on what’s called the RC time constant – this relates the voltage, resistance, and capacitance of the circuit to time. The details of how that works are a topic for another time.

The chip samples the level of the internal capacitor over time. The value is compared to a “reference voltage” that is supplied to the V_{DD} pin on the ADC. The result is given as ratio of the maximum numerical value.

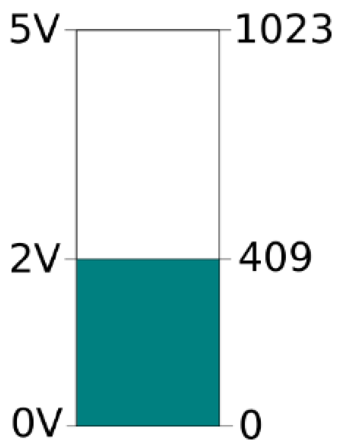
For example, the MCP3008 is a 10bit ADC. That means it will return a value from 0 (representing zero voltage), or 1023 (which is 2^10, 10bits, representing the full reference voltage – V_{DD}) . If our reference voltage is 5V, and our sensor is providing a voltage of 2V to ADC input channel,
then our ADC will return 409:

(2V/5V) * 1023 = 409

In other words, the 2V we are reading on the input in on the ADC is 40% of the reference voltage (5V). 40% of the maximum numerical value is 409.2, which is what is returned (dropping the decimal part)

If we had used a reference voltage of 3.3V instead of 5V, that same 2V would return

(2V/3.3V) * 1023 = 620



Comparing measured 2V out of 5V max to the digital result sent via SPI

The Raspberry Pi Cookbook by Simon Monk has a recipe that shows how to take the same approach manually by building a simple circuit with a capacitor

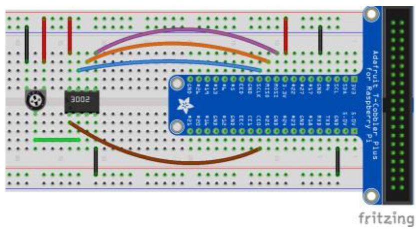
An online example can be found here:

http://www.raspberrypi-spy.co.uk/2012/08/reading-analogue-sensors-with-one-gpio-pin/

So, if you find yourself without an ADC, you can use this this method to get a close approximation to what an ADC does.

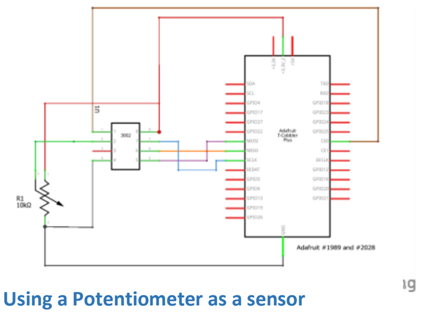
CONNECT AN ANALOG SENSOR

The ADC needs to measure the voltage potential between the sensor and ground. Hooking a sensor up to it involves setting up a point to tap into that potential, and remaining circuitry to connect the sensor to ground while controlling the current, etc…



Most sensors that you run into will vary the voltage as their output signal. There are some sensors that adjust current instead of voltage. These must be hooked up in a different manner.

Read the datasheet for the sensor to be used to ensure you understand it!

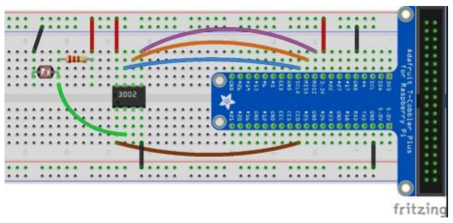


Using a Potentiometer as a sensor

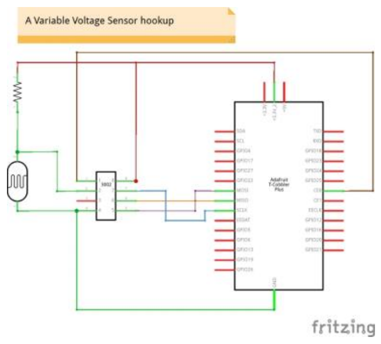
Check out the *Make: Encyclopedia of Electronic Components Vol 3* for lots of details of how different types of sensors work, and details around their use. The last chapter "Sensor Output" had a really good discussion on actually connecting the sensors to a microprocessor.

In this worksheet, we will only look at sensors that vary the voltage.

Here is an example using a potentiometer. We connect the middle pin, the wiper that actually varies the resistance (and thus the voltage), to the input of the MCP3002 (shown by the green wire). This is a very simple version of hooking up a general variable voltage sensor. The potentiometer both varies the output voltage and limits the current flowing to ground.



Other types of variable voltage sensors (such as photocells, force sensors, etc...), need an additional resister to limit the overall current.



READING A VALUE

The functions below can be used to read a value from either a MCP3002, MCP3004, or MCP3008

High level steps:

1. Import the SPI library
2. Setup the SPI device
3. Copy the correct read function for the IC you are using
4. Call the read function, passing in the channel on the IC to read from

Example importing the SPIDEV library, setting up the device, and printing the default settings:

```
import spidev
spi = spidev.SpiDev()

# opens the SPI device connected to the CE0 pin, use spi.open(0, 1) to open the device on CE1 pin instead
spi.open(0,0)

print("SPIDEV Values:")
print("mode:         {}".format(spi.mode))
print("bits_per_word: {}".format(spi.bits_per_word))
print("cshigh:       {}".format(spi.cshigh))
print("loop:         {}".format(spi.loop))
print("lsbfirst:     {}".format(spi.lsbfirst))
print("max_speed_hz:  {}".format(spi.max_speed_hz))
print("threewire:    {}".format(spi.threewire))
print("--------------------------------------------------------------------")
print("")
```

READ FUNCTION FOR MCP3002

```
def read_mcp3002(channel=0, sing_diff=1):
    # Per datasheet
    # For MCP3002:
    # Sent Bit Order:
    #  Start bit
    #  Single / Differential
    #  Channel
    #  MSBF (If high, sends in MSB once,
    #          If low, will send value in LSB after MSB value is sent)
    # all other bits don't matter

    # Received Bit Order
    # Data read in starting 2 bits past sent MSBF bit
    #      Contines for 10 bits
    #
    # Exmaple
    # Start = 1, Single = 1, Channel = 0, MSBF = 0
    # Assume returned value of 912
    #  Sent:    0110 0000 | 0000 0000
    #   Rec:    0000 0011 | 1001 0000
    #  to convert received 10bit value:  Shift first byte left 8 bits, and add second byte to it


    startbit = 64      # 0100 0000
    single = 32        # 0010 0000
    chan = 16          # 0001 0000
    msbf_mode = 8      # 0000 1000

    flags = startbit & 255  #(ensure we have 8 bits, with the 2nd bit set)
    if channel == 1:
        flags = flags | chan

    if sing_diff == 1:
        flags = flags | single

    #forcing Most Significant Byte First mode (MSBF)
    flags = flags | msbf_mode

    #notes - you will always get back the number of bytes you read
    #        when in full duplex (not three wire)
    # incoming bytes are read as you write simultaneously with the 4 wire SPI

    spidata = spi.xfer2([flags, 0])  #send and receive two bytes

    #convert received data
    #note the use of the bitwise & 3 to clear bits we don't care about (just in case)
    data = ((spidata[0] & 3) << 8) + spidata[1]
    return data
```

READ FUNCTION FOR MCP3004 OR MCP3008

```
def read_mcp3008(channel=0, sing_diff=1):
    # Per datasheet
    # For MCP3008 and MCP3004:
    # Sent Bit Order:
    #  Start bit
    #  Single / Differential
    #  D2, D1, D0 (three bits for channel)
    # all other bits don't matter

    # Received Bit Order
    # Data read in starting 2 bits past sent MSBF bit
    #       Contines for 10 bits (then will repeat LSB formatted, but we skip this)
    #
    # Exmaple
    # Start = 1, Single = 1, Channel = 4   ->  0001 1100 0000
    # Assume returned value of 912
    #  Sent:    0000 0001 | 1100 0000 | 0000 0000
    #   Rec:    0000 0000 | 0000 0011 | 1001 0000
    #
    #  pad the sent bits with leading zeros to make everything line up (start bit in 1 position of
first byte)
    #  to convert received 10bit value:  Shift first byte left 8 bits, and add second byte to it


    # Flags to send:
    #startbit = 1 (pass alone in first byte)
    flags = 0
    if channel != 0:
        flags = flags | ((channel & 7) << 4)  # convert   xxxx x111    to   x111 xxxx

    if sing_diff != 0:
        flags = flags | 128 # 1000 0000


    #notes - you will always get back the number of bytes you read when in full duplex
    # incoming bytes are read as you write simultaneously

    spidata = spi.xfer2([1, flags, 0])  #send and receive three bytes [start bit, flags, empty byte]

    #convert received data
    #note the use of the bitwise & 3 to clear bits we don't care about (just in case)
    data = ((spidata[1] & 3) << 8) + spidata[2]
    return data
```

EXAMPLE USAGE

```
import spidev
spi = spidev.SpiDev()
spi.open(0,0)


#comments removed from function for brevity
def read_mcp3002(channel=0, sing_diff=1):
    startbit = 64      # 0100 0000
    single = 32        # 0010 0000
    chan = 16          # 0001 0000
    msbf_mode = 8      # 0000 1000

    flags = startbit & 255  #(ensure we have 8 bits, with the 2nd bit set)
    if channel == 1:
        flags = flags | chan

    if sing_diff == 1:
        flags = flags | single

    flags = flags | msbf_mode

    spidata = spi.xfer2([flags, 0])  #send and receive two bytes
    data = ((spidata[0] & 3) << 8) + spidata[1]
    return data


try:
    while True:
        channeldata = read_mcp3002(0)  #read channel 0
        print("Raw ADC:        {}".format(channeldata))
        time.sleep(1)


#hit ctrl-C to exit…
except KeyboardInterrupt:
    print("closing spidev")
    spi.close()
```