visar 1 (/tag/visar)

Beaglebone Black I2C DAC

How to use an I2C DAC with the Bone.

Overview

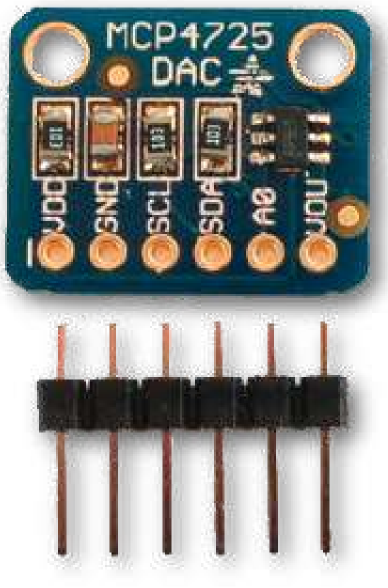
To better understand DACs and what they do, I recommend checking out all allaboutcircuits's notes on Digital-Analog Conversion (http://www.allaboutcircuits.com/vol_4/chpt_13/1.html)

Also, check out Sparkfun for information on I2C communication (https://learn.sparkfun.com/tutorials/i2c/all)

and Adafruit, for the I2C python commands (http://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/i2c) to communicate with the chip.

Setup

The DAC I used for this was the MCP4725 (https://www.adafruit.com/products/935)



The MCP4725 needs between 3-5V to operate [1], so either the V_{DD} 3.3V or V_{DD} 5V / SYS 5V supplies from the BBB will do.

Pin 9_19 and Pin 9_20 are used for the default (I2C1) bus on the BBB. With Pin 9_19 being our clock signal (SCL) and Pin 9_20 being the data transfer line (SDA). To wire this up, we just connect these to the corresponding pins on the DAC.

Both angstrom and Ubuntu for the Beaglebone come with some commands to make interfacing with I2C easier. By typing `i2cdetect` into bash, the bone will scan for all devices currently connected.

```
root@THE-BONE:~# i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- 62 -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

This shows there is a device connected at address `0x62` . If we look at the 'Technical Details' for the MCP4725: this is the default address [2].

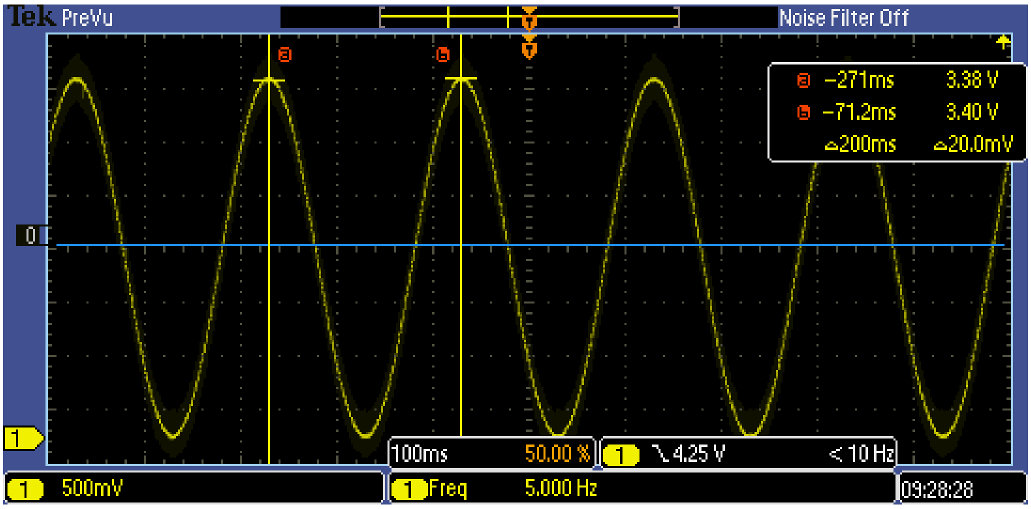
You can download my MCP4725 code (http://thebrokendesk.s3-us-west-1.amazonaws.com/documents/mcp4725.py) (largely taken from Adafruit [2]) by typing the following in your BBB project directory.

```
wget  https://github.com/priyanshus1/BBB_DAC/blob/master/mcp4725.py
```

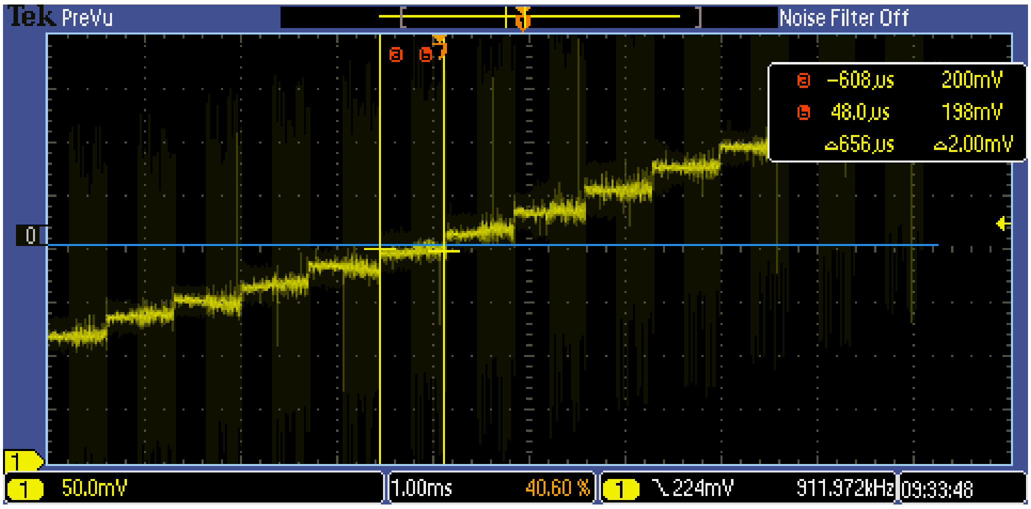
and it can be used like so:

```
import numpy as np # if not installed, use: pip install numpy
import time
import os
import sys

ROOT_DIR = os.path.dirname(os.path.abspath(__file__))

if ROOT_DIR not in sys.path:
    sys.path.append(ROOT_DIR)
import mcp4725

dac = mcp4725.DAC()

sig = lambda t: 50*np.sin(t*5*2*np.pi) +50
start_time = time.time()

while True:
    dac.set_voltage(sig(time.time()-start_time))
```

*If you're interested in understanding how it works, jump down the the explanation outputing a sin wave at 5 Hz.

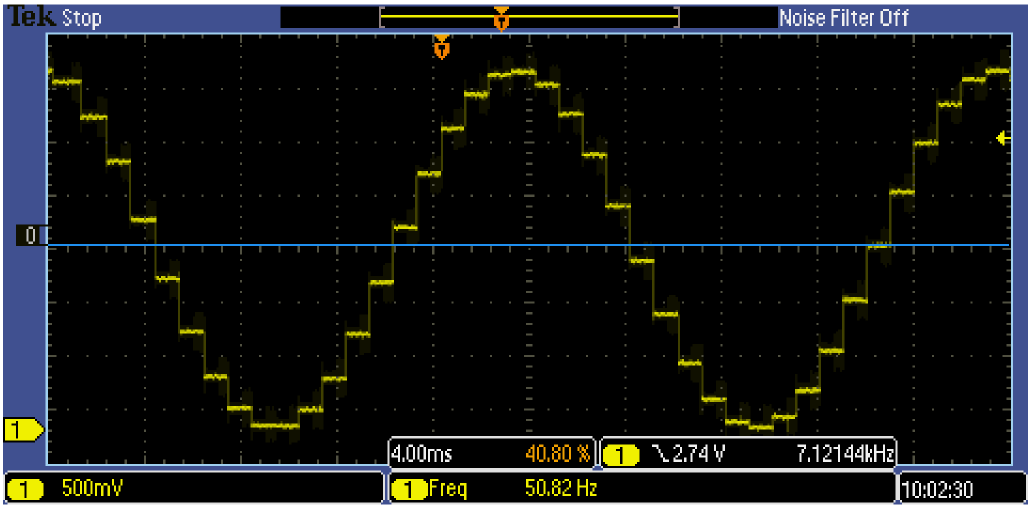


if we zoom in on a section of the waveform, we can see the step-size incrementation.



The I^{2}C clock speed on the Bone is about 100 KHz [3]. This will greatly limit what you can use this DAC for (e.g. can't use it to play music).

At a 50 Hz waveform the step size is very evident.

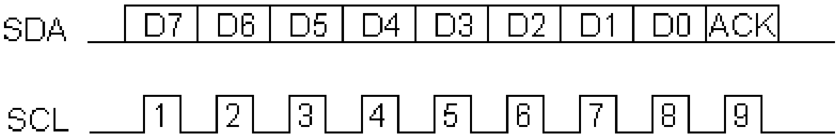


Explanation

```
class DAC:

    RESOLUTION = 2**12 - 1
    WRITE_REGISTER = 0x40

    # Set the address to the default
    def __init__(self, address=0x62):
        self.i2c = Adafruit_I2C(address)

    # Send the voltage in two bytes
    def send_voltage(self, bits):
        bytes = [(bits >> 4) & 0xFF, (bits << 4) & 0xFF]
        self.i2c.writeList(self.WRITE_REGISTER, bytes)

    # Limit the max and min voltage that can be set
    # and convert to the resolution of the dac
    def set_voltage(self, voltage):
        if voltage > 100:
            voltage = 100
        elif voltage < 0:
            voltage = 0

        bits = int(voltage/100. * self.RESOLUTION)
        self.send_voltage(bits)
```

The DAC has a 12 bit resolution.
Each bit can be a 0 or 1 - two possible values. Therefore, we have $2^{12} = 4096$ possible combinations to make our wave form.

Now, I^{2}C will only transfer one byte (8 bits) at a time [3].



We need to divide our voltage signal into two 8 bit packages to send it. This is accomplished through the following:

```
bytes = [(voltage >> 4) & 0xFF, (voltage << 4) & 0xFF]
```

It might be easiest to explain this with an example:

Lets say, we want to output 3/4 of our maximum voltage.

$$\text{voltage} = \textbf{int}(\frac{3}{4} \cdot 4095) = 3071$$
$$\text{or, in binary:}$$
$$1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$$

Because we can transfer 4096 possible values *including 0* our maximum value is 4095.

```
voltage >> 4
```

This is a 'right-shift' bitwise operation. It shifts our binary value for voltage to the right, 4 bits [4]

$$\begin{array}{cccccccccccc}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & & & \longrightarrow & & & & & & \\
 & & & & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}$$

The shifted value is then operated on by the 'bitwise &'

```
(voltage >> 4) & 0xFF
```

`0xFF` is the hex value for 11111111, or the max value for 8 bits (1 byte)

$$\begin{array}{ccccccccc}
 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\& & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
 & & & & & & & & \\
 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}$$

This is now the first byte we send.

To generate the second byte, we use the 'left-shift' operator:

```
voltage << 4
```

Similarly to the 'right-shift' operator, this one shifts the binary number to the left 4 places - filling in the new bits with 0s

$$\begin{array}{cccccccccccccccc}
 & & & & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & & & & \longleftarrow & & & & & & \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}$$

we & the shifted value to get the second byte

```
(voltage << 4) & 0xFF
```

$$
\begin{array}{ccccccccccccccccc}
 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\& & & & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
 & & & & & & & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}
$$

Now:

```
bits = [ 10111111, 11110000 ]
```

The last 4 bits in the second byte are ignored [1]-section 6.1.2

References

1. MCP4725 Datasheet (https://www.sparkfun.com/datasheets/BreakoutBoards/MCP4725.pdf)
2. MCP4725 Breakout Board (https://www.adafruit.com/products/935)
3. I2C tutorial (http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html)
4. Bitwise operation (https://wiki.python.org/moin/BitwiseOperators)