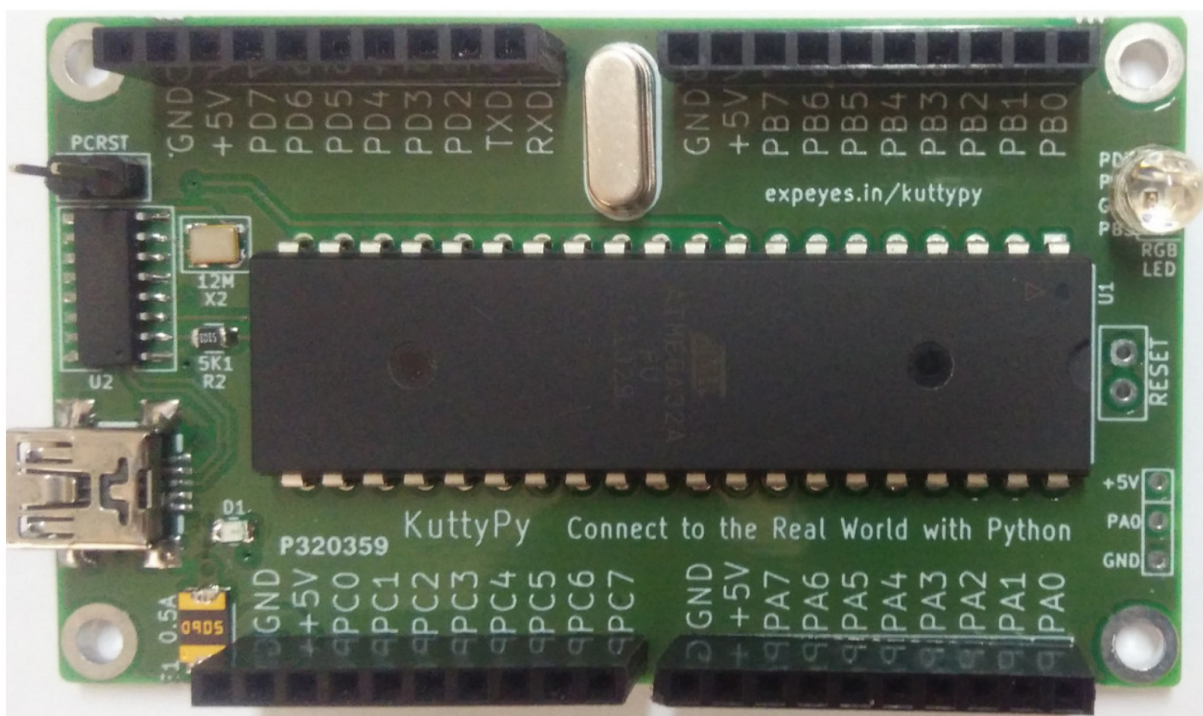


KuttyPy

INTERACTIVE PLAYGROUND
[MICROCONTROLLER TRAINING UTILITY]



- * 1.5kB Bootloader with real-time access to registers
- * Python library and Graphical interface
- * * ATMEGA32 Development board

Contents

1	Introduction	3
1.1	Example applications for KuttyPy	3
2	Software	4
2.1	Introduction to the graphical utility	5
2.2	Installation	6
2.3	Installation from source	6
3	Executing Python code	7
3.1	Example: logging and plotting ADC values with matplotlib	7
4	Compiling and uploading C code for standalone apps	8
4.1	Example C Code: Blink all pins on PORTB	8
4.2	Difference between Arduino and KuttyPy Bootloaders	9
4.3	Appendix	10
4.3.1	Atmega32 pin diagram	10
4.3.2	Python Code for writing to MCP4725 I2C DAC, and readback using ADC	10

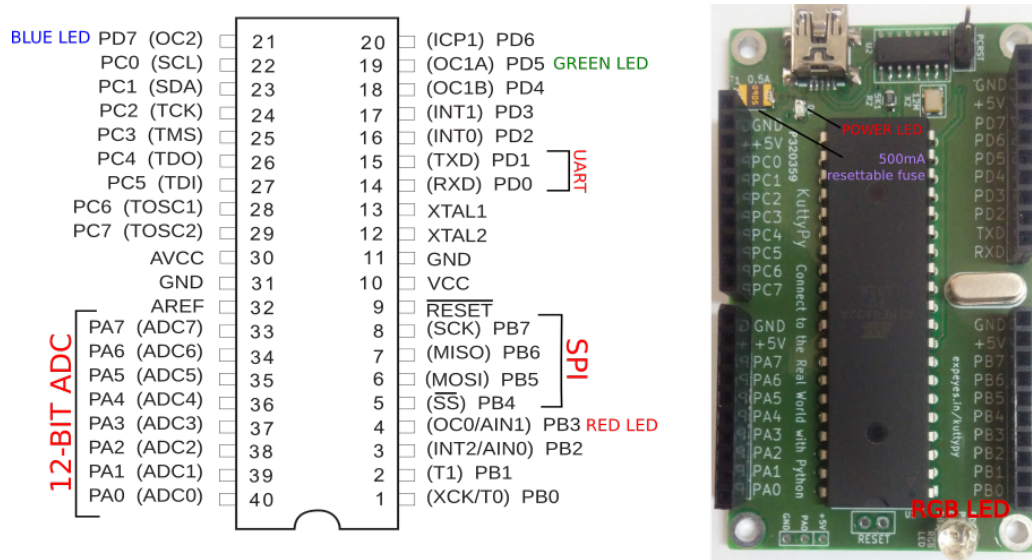


Figure 1: Atmega32 pinout, and KuttyPy hardware

1 Introduction

The kuttyPy (*/kʊtipAI/*) Microcontroller training utility allows easy reading and writing of the registers in microcontrollers via a connected computer containing its python library. setReg and getReg function calls serve debugging and monitoring purposes, and combined with Python's visualization and analytical utilities, this approach has immense pedagogical potential for beginners.

The kuttyPy hardware is an ATMEGA32 microcontroller development board developed by the [ExpEYES project](#), and is currently supported by this software. It contains the kuttyPy firmware, but can also be used to run other programs via its bootloader.

The KuttyPy can also be used as an Arduino board using the MightyCore board type for Atmega32, but the idea is to encourage learning microcontrollers via register access, and discourage using wrapper functions which hide the actual implementation.

1.1 Example applications for KuttyPy

- Learn how registers are used to control microcontroller functions
- Monitor up to 28 input signals in automated devices

- Use the 0-5V ADC inputs available on PORTA as voltmeters. differential mode operation with up to 200x gain is also possible.
- Club with other python libraries such as opencv to make interactive models such as a motion sensing camera mount.
- Create an ADC data-logger and plotter using matplotlib (example code included)
- Read data from I2C/SPI devices, and control stepper and servo motors.
- Compile C code with avr-gcc, and upload using the 'upload-hex' option, or via Avrdude in a terminal

2 Software

The software *KuttyPy-GUI* is capable of real-time access to the KuttyPy hardware as shown in [Figure 2](#).

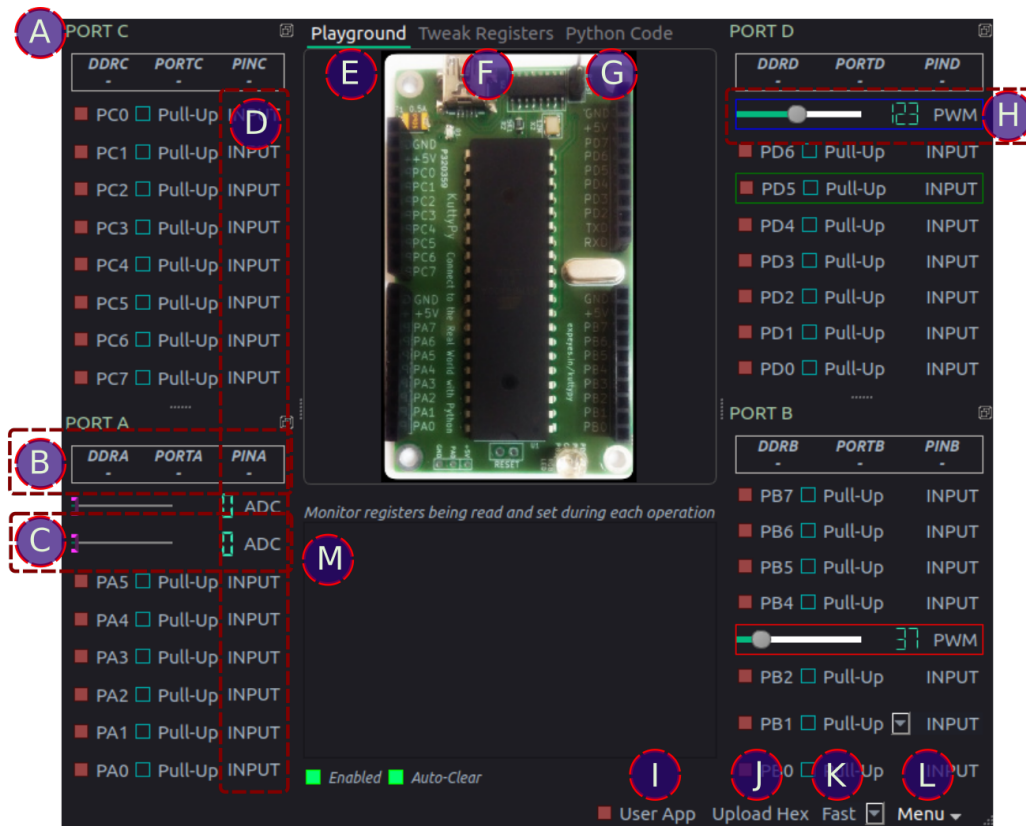


Figure 2: Open-source software for the KuttyPy '**KuttyPy-GUI**':

2.1 Introduction to the graphical utility

- (A) PORT Docks : The hardware has segregated all 4 PORTs of 8 pins each into 4 separate pin headers . The software has also segregated them in a similar fashion.
- (B) PORT Register values: The current value of the registers associated with the I/O functions of the ports are shown here. Click on them to cycle between display modes such as hex, decimal, or binary. Pin states are best understood when viewed in binary mode.
 - DDRx : Each bit of this register represents the corresponding pin of the 8-bit PORT. bit value 1 implies output functionality, and 0 means the pin is a high impedance input
 - PORTx: In output mode, the corresponding pins are connected to 5 volt if the corresponding bit is 1, or 0 Volts(Ground) if it is 0. In input mode, if the bit is high, an internal pull-up is enabled to prevent the input from being in an undefined state.
 - PINx: This register stores the value of the inputs. If an input pin is connected to the supply voltage, the corresponding bit will have value 1, otherwise 0.
- (C) ADC monitor: The pins on PORTA have a 10 bit ADC included. The ADC value which lies between 0 and 1023 is shown on an LCD display as well as a slider. Click on the LCD display to change the acquisition mode to differential or amplified.
- (D) Monitor type: Click to change between Input/Output/ADC/PWM/Counter depending on function availability
- (E) Playground : Automatic monitoring and control of I/O pins
- (F) Tweak registers : Manually specify registers to read or write, and also the sequence in which this should be done.
- (G) Python code : Write python code for automated tasks
- (H) PWM : Set the PWM duty cycle [Available only on PD5,PD7, PB3].
- (I) User App : The kuttypy hardware can also contain a user uploaded hex file. By default this is an LED fading app which also continuously dumps letters into the serial port. Click this button to switch execution to the use app, and freeze monitoring utilities. The log window specified by (M) turns into a serial monitor. Uncheck to restore monitoring.
- (J) Upload a hex file compiled with AVR-GCC for ATMEGA32
- (K) Speed: Change the refresh rate of the monitoring utility

- (L) Menu : Save the window as an svg file, change the theme, and more...
- (M) Log window : Show the registers being read/written in during operation of the monitoring utility.

2.2 Installation

The software has been packaged for Windows as well as Ubuntu.

- **Installation on Windows** Download the setup file from the [drive link](#), and follow the instructions to install it.
- **Installation on Ubuntu** Download the deb file from <https://csparkresearch.in/assets/installers/kuttypy-1.0.0.deb>, and install it using a package installer such as Gdebi. It will be an official Debian package soon, and should also be installable via the Software Centre on Debian based OSes such as Ubuntu.

2.3 Installation from source

The source code for the software is Python based, and can be downloaded from github. On Ubuntu :

Install dependencies (Command to be run only once. requires an internet connection)

- `sudo apt-get install python3-pyqt5 python3-pyqt5.qtsvg python3-serial`

Run the software:

- `python3 KuttyPyGUI.py`

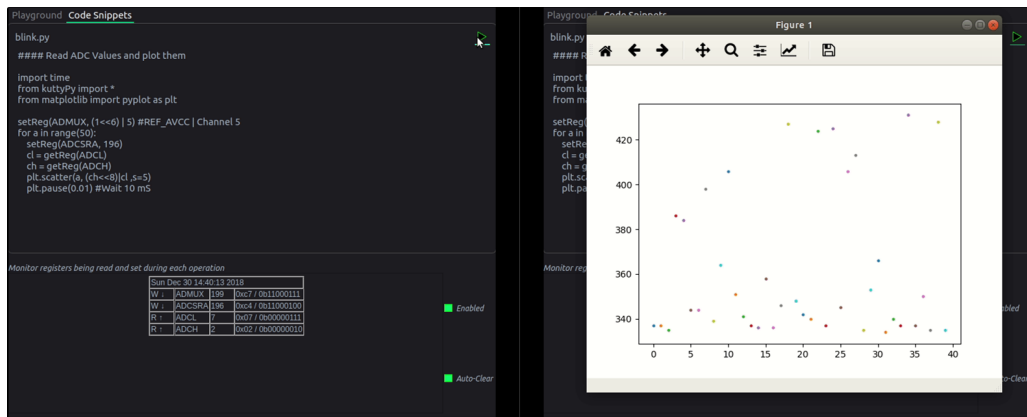


Figure 3: Output of ADC logger code. The pin was floating, so values are fluctuating wildly

3 Executing Python code

The graphical utility includes a Python code editor with examples such as a data logger. You can also write python scripts which import the kuttyPy library, and execute them.

3.1 Example: logging and plotting ADC values with matplotlib

```
# Read ADC Values and plot them
import time
from kuttyPy import *
from matplotlib import pyplot as plt

setReg(ADMUX, (1<<6) | 5) #REF_AVCC | Channel 5 (PA5)
for a in range(50):
    setReg(ADCSRA, 196)
    cl = getReg(ADCL)
    ch = getReg(ADCH)
    plt.scatter(a, (ch<<8)|cl ,s=5)
    plt.pause(0.01) #Wait 10 mS
```

You can run this code in a terminal using the following command: `python3 logger.py`

You may also paste this code into the python code editor tab of the GUI, and press the play button to start execution. The results are shown in [Figure 3](#).

4 Compiling and uploading C code for standalone apps

The KuttyPy monitoring functions are a part of the 1.5kB bootloader, and the rest of the flash memory is available for executing user uploaded hex files. The following code can be used to compile and upload a file called blink.c

```
avr-gcc -Wall -O2 -mmcu=atmega32 -Wl,-Map,blink.map -o blink blink.c
avr-objcopy -j .text -j .data -O ihex blink blink.hex
avrdude -b 38400 -P /dev/ttyUSB0 -pm32 -c arduino -U flash:w:blink.hex
```

The third step (uploading) requires Avrdude . However, you can also upload the hex file from the graphical utility without the need for avrdude.

4.1 Example C Code: Blink all pins on PORTB

```
#include <avr/io.h>
void delay_ms (uint16_t k) { // idle for k milliseconds, 8MHz clock assumed
    volatile uint16_t x;
    while(k--) {x=532; while (x--);}
}
int main (void) {
    DDRB = 255;    // Data Direction Register for port B. All outputs
    for(;;) {
        PORTB = 255; // All pins HIGH (5V)
        delay_ms(100); // wait 100mS
        PORTB = 0;    // All pins LOW (0V)
        delay_ms(200); // wait 100mS
    }
    return 0;
}
```

To run this uploaded hex file, simply connect the kuttyPy to a 5V power source such as a phone charger, or USB port. The GUI is not required since the microcontroller is in standalone operation.

If the GUI is active, then execution can be switched to your app by clicking the 'user app' button in the statusbar. This temporarily suspends the monitoring functions, and the log window transforms into a serial monitor.

4.2 Difference between Arduino and KuttyPy Bootloaders

On power-on or reset, a bootloader, if included, is the first piece of code which runs.

It briefly listens on a communications port for any incoming commands, and if none are detected, skips execution to a predefined section of the flash memory where user uploaded code is stored. The bootloader primarily includes commands to read data from the communications port, and store incoming hex code to the user-space of the flash memory.

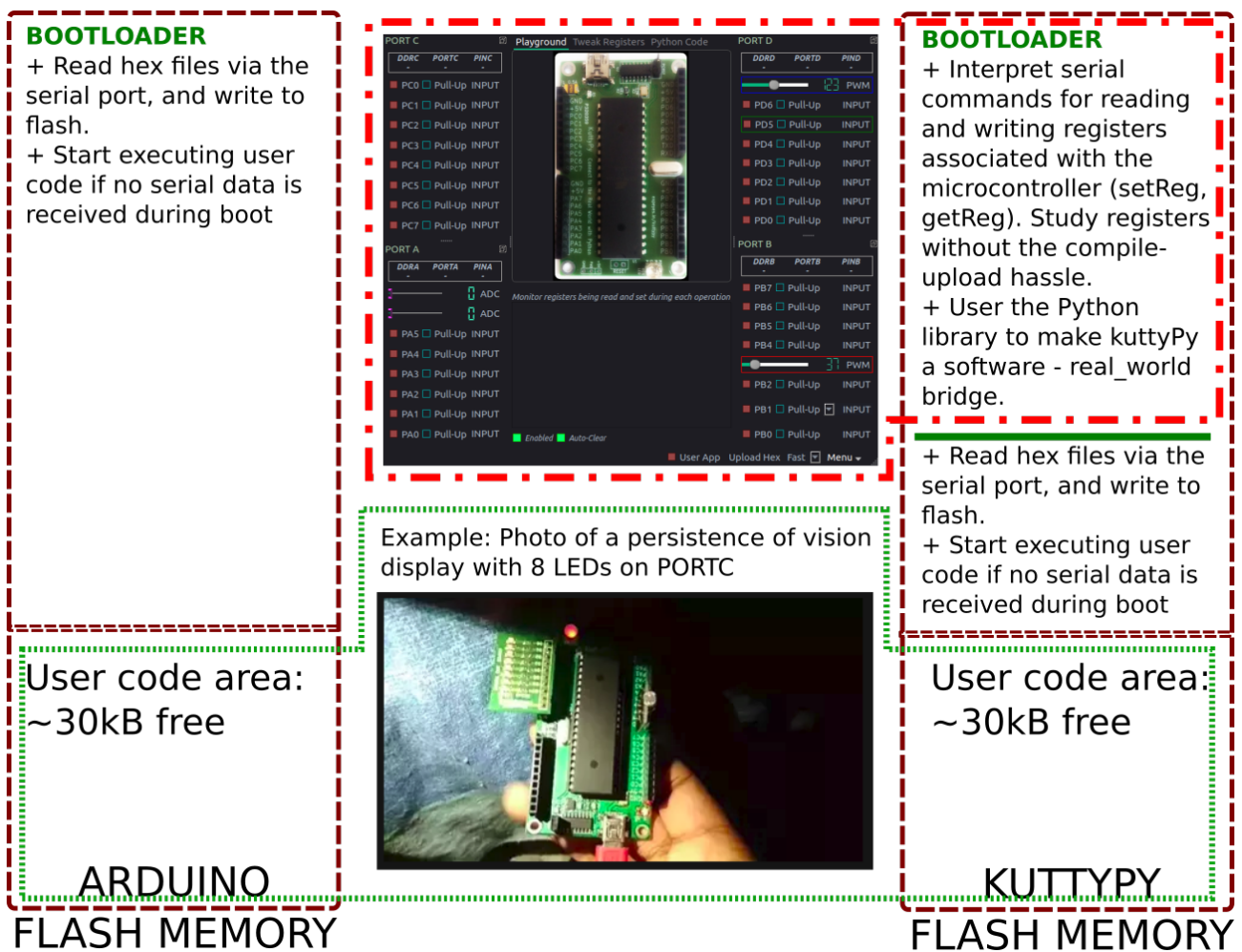


Figure 4: Flash memory is segregated into a bootloader, and a user space. The kuttypy bootloader has enhanced functionality which enables it to be used as a data acquisition and control utility, but also works as the regular arduino bootloader with a 38400 BAUD.

4.3 Appendix

4.3.1 Atmega32 pin diagram

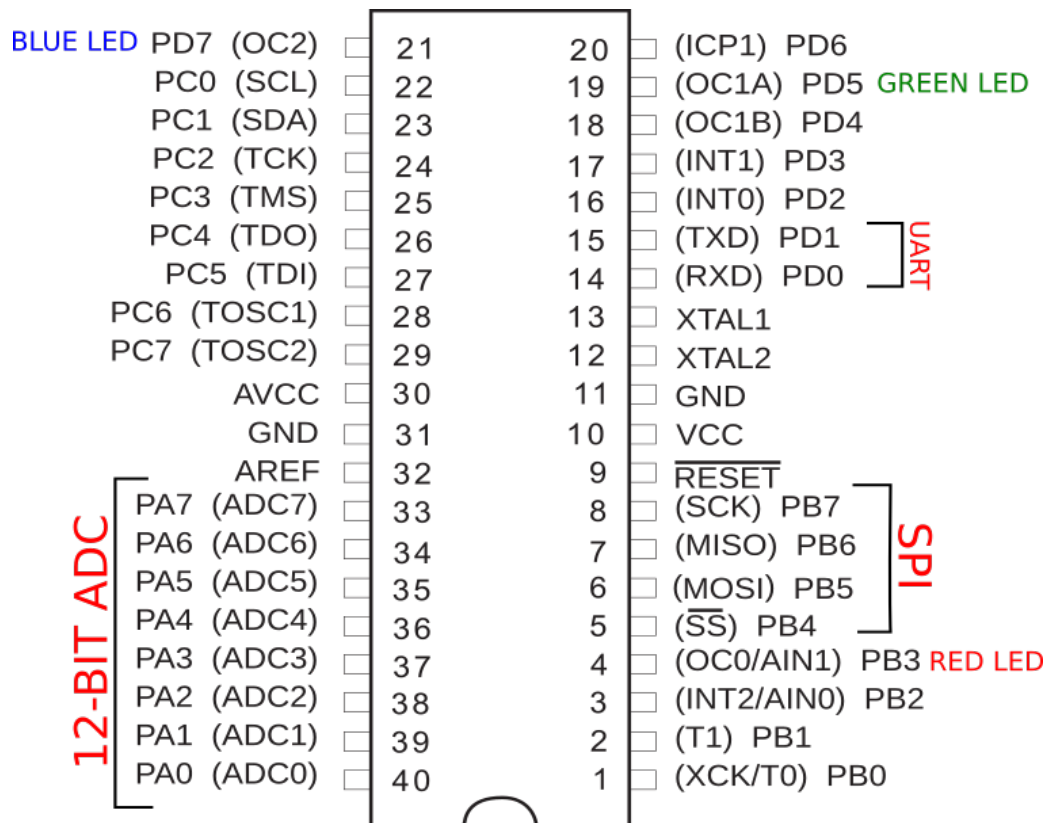


Figure 5: Pin labels and functions for the ATMEGA32 processor used with the kuttyPy

4.3.2 Python Code for writing to MCP4725 I2C DAC, and readback using ADC

Explains the construction of I2C and ADC communications via simple functions.

```
'''
Change the voltage at PA0 using a potentiometer, using a connection like
5V__/\__/\__GND
'''

import time
from kuttyPy import *

ADSP2 = 2;REFS1 = 7;REFS0 = 6
```

```
ADC_SPEED = 7    # ADCClk = (8 MHz/ 128) = 62.5 KHz =>208 usec
```

```
ADEN  = 7; ADIF  = 4; ADSC  = 6
```

```
setReg(DDRA,0) # PORT A set to input type
```

```
def readADC(ch):      # Read the ADC channel. set REF_AVCC(5V) instead of
    REF_INT(2.4V)
    setReg(ADMUX, REF_AVCC | ch)
    setReg(ADCSRA, (1 << ADEN) | (1 << ADSC) | ADC_SPEED) # Enable the ADC
    low = getReg(ADCL) # Read 8 LSB
    hi = getReg(ADCH) # Read 2 MSB
    return (hi << 8) | low
```

```
def initI2C(): # Initialize I2C
    setReg(TWSR,0x00)
    setReg(TWBR,0x46)
    setReg(TWCR,0x04)
```

```
def startI2C():
    setReg(TWCR,(1<<TWINT) | (1<<TWSTA) | (1<<TWEN))
    while ( not (getReg(TWCR) & (1<<TWINT))): pass
```

```
def stopI2C():
    setReg(TWCR,(1<<TWINT) | (1<<TWSTO) | (1<<TWEN))
    time.sleep(0.001)
```

```
def writeI2C(val):
    setReg(TWDR,val)
    setReg(TWCR,(1<<TWINT) | (1<<TWEN))
    while (not (getReg(TWCR) & (1<<TWINT))): pass
```

```
def readI2C(ack):
    setReg(TWCR,(1<<TWINT) | (1<<TWEN) | (ack<<TWEA))
    while (not(getReg(TWCR) & (1<<TWINT))): pass
    return getReg(TWDR)
```

```
def setVoltage(value): #Write to MCP4725 single channel, 12-bit DAC.
    startI2C()
    writeI2C(0x62<<1)
```

```
writeI2C(0x40) #write DAC. 0x60 for writing to eeprom also
writeI2C((value>>4)&0xFF)
writeI2C((value&0xF)<<4)
stopI2C()

initI2C()
V=0
while V<4095: # Write DAC codes from 0-4095 in steps of 100, and readback using ADC
    setVoltage(V)
    time.sleep(0.01)
    print 5*V/4095.,5*readADC(0)/1023.
    V+=100
```
