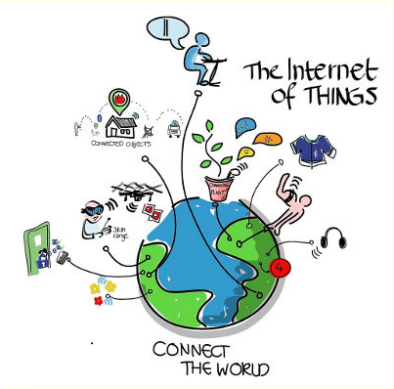
SE-IOT: Internet of Things

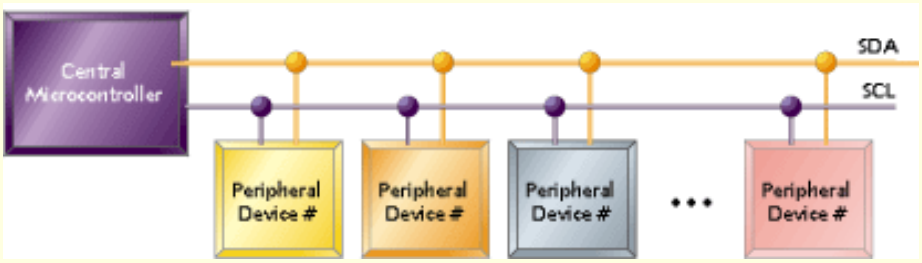


Working with Analogue I/O

Derek Kiong
dkiong@nus.edu.sg

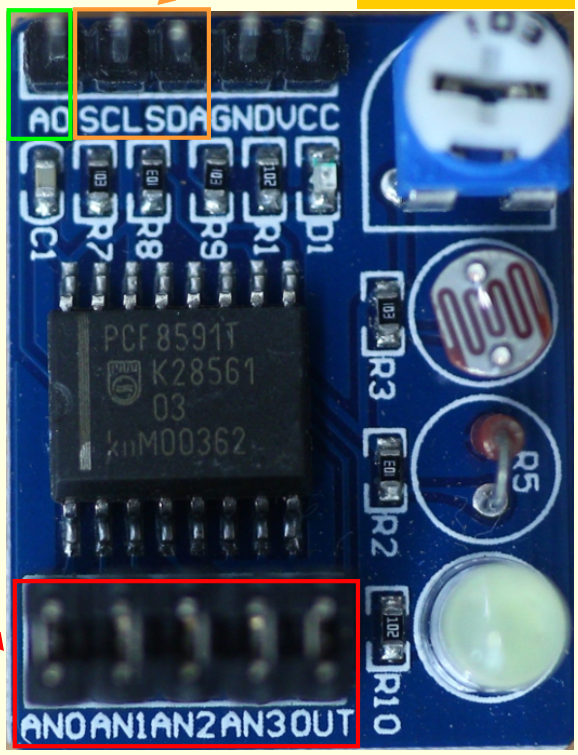
I2C (Inter-Integrated Circuit)

- I2C is a serial computer bus with two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems.



AD/DA conversion via I2C module

- selectable base address
- 4 analogue voltages converted to byte values
- byte value converted to proportional voltage



LCD display: 6 GPIOs vs 2-wire I2C module

i2cdetect

- Utility program **`i2cdetect`** scans for I2C components

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```


Alternate addresses

- AD/DA board has A0 pin to change board address

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- 49 -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```


Working with I2C components

- Each I2C component
  - Port number
  - Address
  - Read/write data/commands

```
i2cget 1 0x48 0x01
i2cset 1 0x48 0x40 0x80
```

- (what to read/write is dependent on component)

I2C_device class

```
import smbus

class I2C_device:
   def __init__(self, addr, port=1):
      self.addr = addr
      self.bus = smbus.SMBus(port)

   def read(self):
      return self.bus.read_byte(self.addr)

   def read_data(self, cmd):
      return self.bus.read_byte_data(self.addr, cmd)

   def read_block_data(self, cmd):
      return self.bus.read_block_data(self.addr, cmd)
```


I2C_device class

```
def write_cmd(self, cmd):
    self.bus.write_byte(self.addr, cmd)
    sleep(0.0001)

def write_cmd_arg(self, cmd, data):
    self.bus.write_byte_data(self.addr, cmd, data)
    sleep(0.0001)

def write_block_data(self, cmd, data):
    self.bus.write_block_data(self.addr, cmd, data)
    sleep(0.0001)
```


AD/DA Operations

- 4 analogue inputs **`AIN0`**, **`AIN1`**, **`AIN2`**, **`AIN3`** read with cmd **`0x00`**, **`0x01`**, **`0x02`**, **`0x03`**
- Corresponds to

```
write_cmd(0x00)
write_cmd(0x01)
write_cmd(0x02)
write_cmd(0x03)
```

- AOUT output controlled via write with cmd 0x40 with byte value

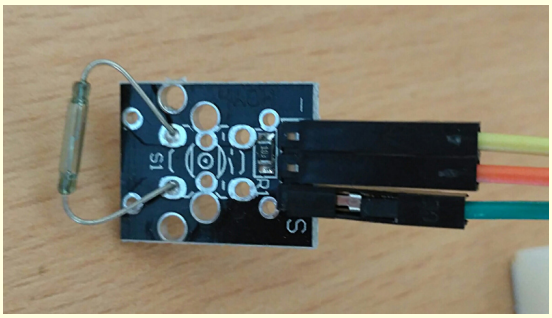
```
write_cmd_arg(0x40, val)
```


AD/DA Board class

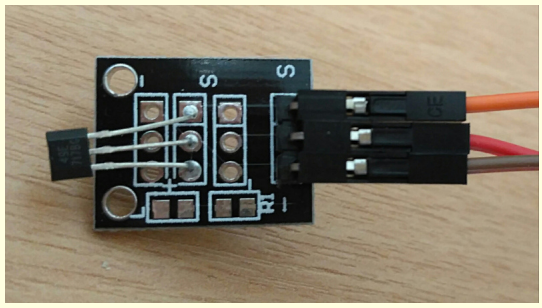
```
class Board:
    def init__(self, addr=0x48, port=1):
        self.device = I2C_device(addr, port)

    def control(self):
        self.device.write_cmd(0x01)
        self.device.read()
        return self.device.read()

    def light(self):
        self.device.write_cmd(0x02)
        self.device.read()
        return self.device.read()
    ...
    def output(self, val):
        self.device.write_cmd_arg(0x40, val)
```


Sensor Pack

(1) Digital Sensors
(micro reed switch)



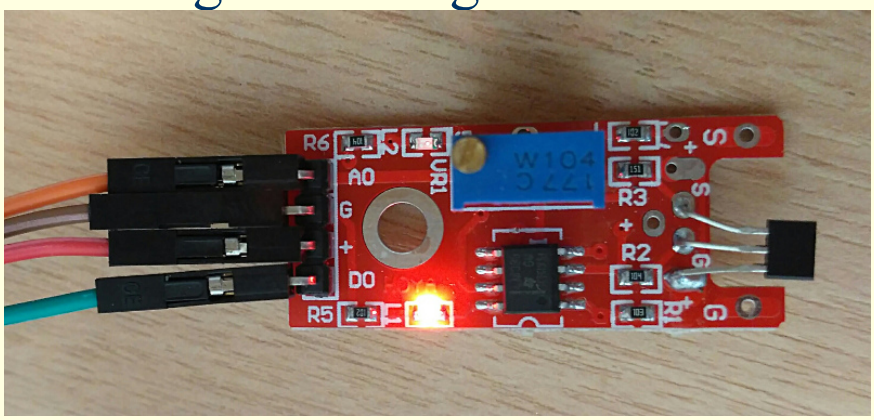
(2) Analogue Sensors
(class Bihor magnetic sensor)



- -ve/negative/ground
+ +ve/positive
s signal/**out**

Sensor Pack

(3) Combo digital-analogue sensors



G -ve/egative/ground

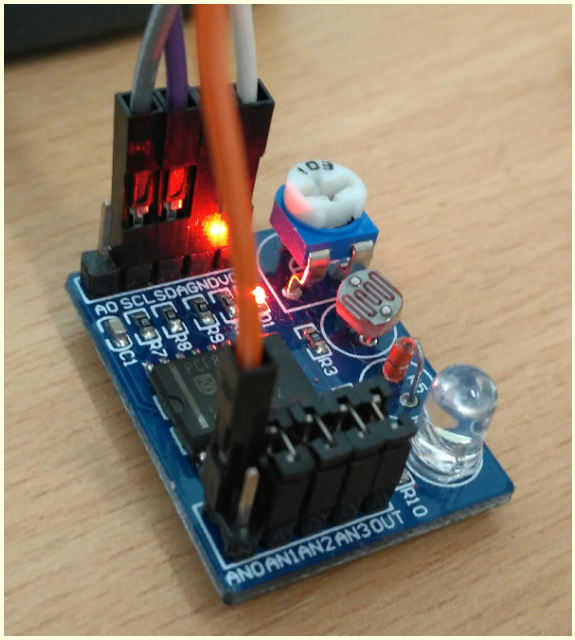
\+ +ve/positive

AO **a**nalogue **out**

DO **d**igital **out**

Using AD-DA board

- Analogue signals read/written by AD-DA convertor

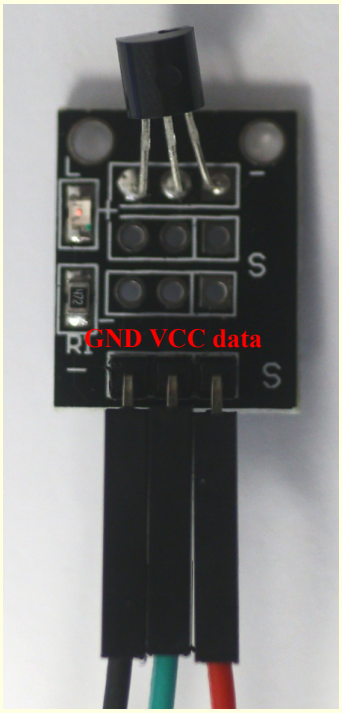


1-Wire communication

- 1-Wire is a device communications bus system
- similar in concept to I2C bus; but lower data rates
- Apple MagSafe and Dell power supplies, displays, and Mac laptops use the 1-Wire protocol to send and receive data

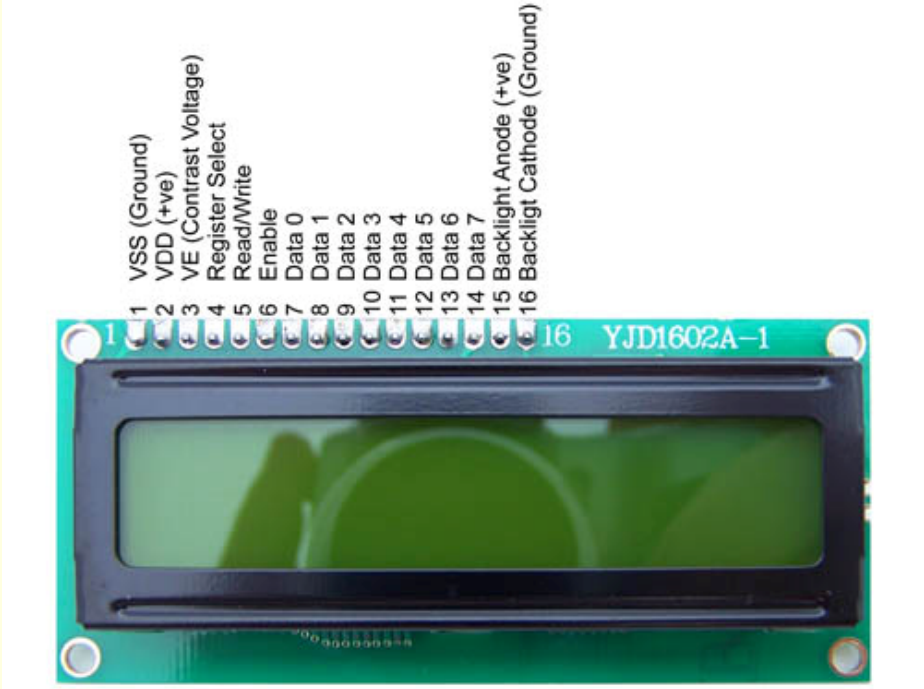
DS18B20 digital temperature sensor

- Each device has a unique identifier code
- Raspberry Pi drivers loaded via **`dtoverlay=w1-gpio`** in **`/boot/config.txt`**
- Device recognised and list it in **`/sys/bus/w1/devices/`**
- 1-wire data pin defaults to **`GPIO 4`** (pin 7)
- Data read from ***`id`*`/w1-slave`**



LCD display

- Data pins **`Data[0-7]`** and control pins **`Rs,En`**



LCD display

- Values at data pins taken as **data** if **`Rs=1`** and **commands** if **`Rs=0`**
- Values may be written in 4-bit mode □ upper 4-bits followed by lower 4-bits
- Values used when **`En`** pin is strobed □ ie brought **high** and then **low**

Write data format

- Writing to LCD via I2C

D3	D2	D1	D0	BL	En		Rs

Fragment of Lcd class

```
class Lcd:
   address = (0, 0x80, 0xC0, 0x94, 0xD4)

   def write_four_bits(self, data):
      self.device.write_cmd(data | LCD_BACKLIGHT)
      self.strobe(data)

   def write(self, cmd, mode=0):
      self.write_four_bits(mode | (cmd & 0xF0))
      self.write_four_bits(mode | ((cmd << 4) & 0xF0))

   def display_string(self, string, line):
      self.write(address[i])        # command
      for char in string:
         self.write(ord(char), Rs) # data
```

Summary

- Analogue signals require conversion to integer values
- I2C only requires 2 lines: SDA and SCL
- More convenient compared with multiple GPIO pins and analogue data