# SE-IOT: Internet of Things


## Working with Binary Signals

Derek Kiong
dkiong@nus.edu.sg





## GPIO Module

- RPi.GPIO module allows working with GPIO devices

```
import RPi.GPIO as GPIO
```

- Pin numbering mode
  - Board number

```
GPIO.setmode(GPIO.BOARD)
```

  - Broadcom SOC channel designation

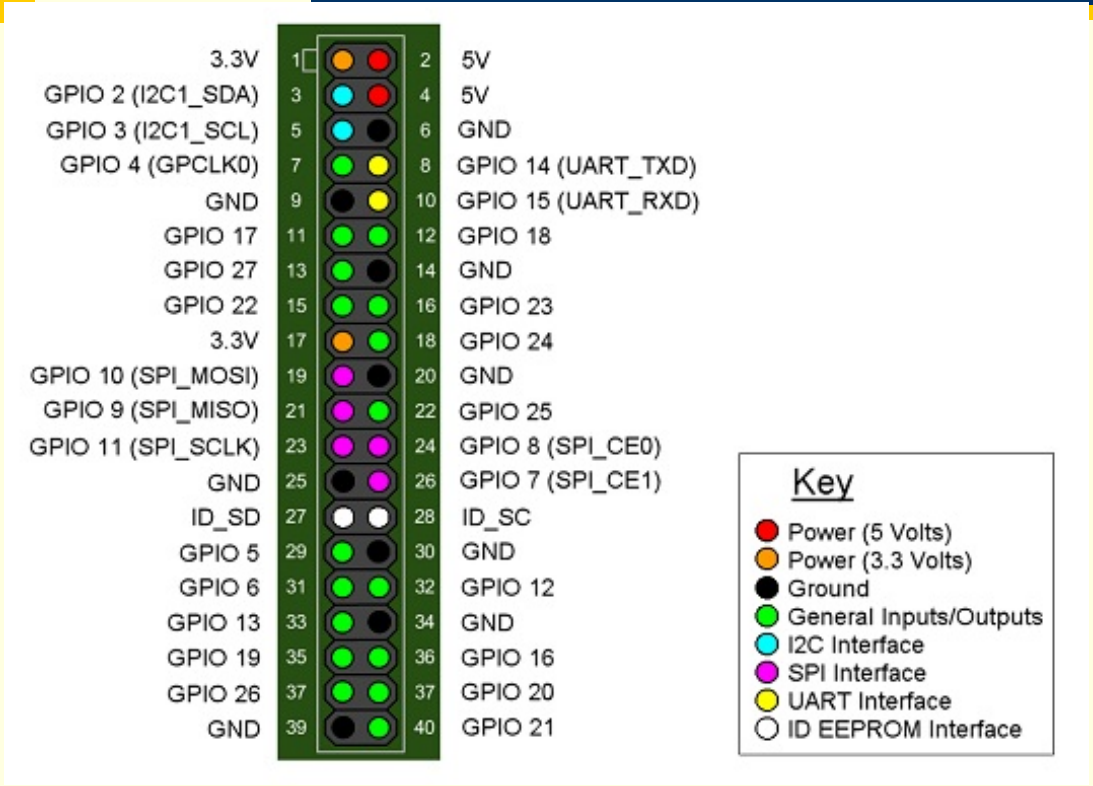
```
GPIO.setmode(GPIO.BCM)
```

- Set warnings

```
GPIO.setwarnings(False)
```

# GPIO.BOARD vs GPIO.BCM



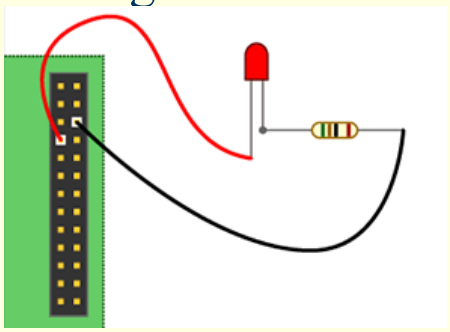
# GPIO operations (output)

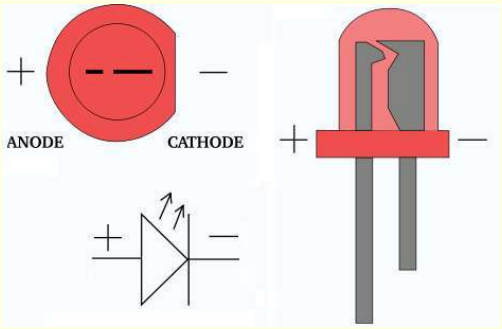
- Output mode
  `GPIO.setup(pin, GPIO.OUT)`
- Bring pin low □low potential at pin
  `GPIO.output(pin, GPIO.LOW)`
- Bring pin high □high potential at pin
  `GPIO.output(pin, GPIO.HIGH)`
  - may test via LED and limiting resistor in series

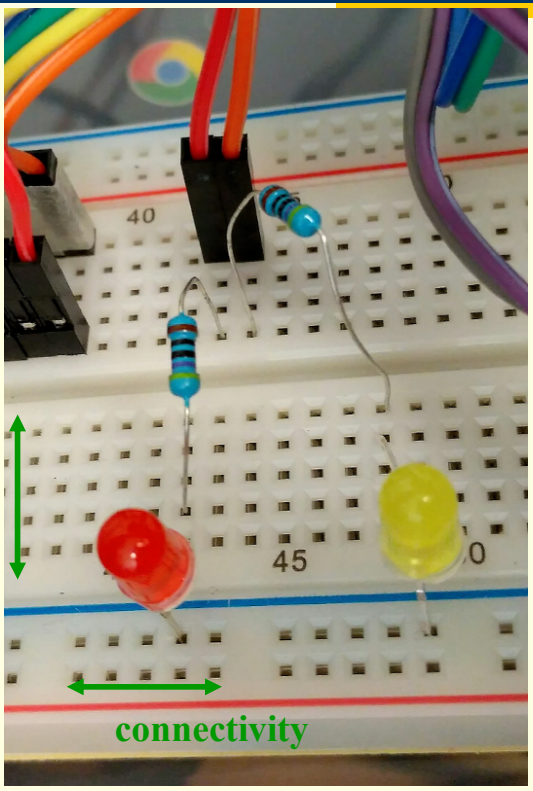
  

# LED connection

- LED with 240 ohms limiting resistor





# Wiring 4 channel Relay

| Relay pins | RP pins |
|---|---|
| GND | 6 (ground) |
| IN1 | 15 (GPIO 22) |
| IN2 | 16 (GPIO 23) |
| IN3 | 18 (GPIO 24) |
| IN4 | 22 (GPIO 25) |
| VCC | 2 (+ve) |

# Operating 4-channel Relay

```
import RPi.GPIO as GPIO
import sys

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
pinList = [22,23,24,25]
if len(sys.argv)>2:
    [cmd,lamp,state] = sys.argv
    lamp = int(lamp)    # which lamp?
    state = int(state) # off/on?
    GPIO.setup(pinList[lamp], GPIO.OUT)
    GPIO.output(pinList[lamp],
                (GPIO.HIGH if state == 0 else GPIO.LOW))
else:
    print "Usage: %s <relay> <0/1>" % sys.argv[0]
```

# Operating 4-channel Relay

```
$ python relay.py 0 1    # turns IN1 on
$ python relay.py 0 0    # turns IN1 off
$ python relay.py 1 1    # turns IN2 on
$ python relay.py 1 0    # turns IN2 off
$ python relay.py 2 1    # turns IN3 on
$ python relay.py 2 0    # turns IN3 off
$ python relay.py 3 1    # turns IN4 on
$ python relay.py 3 0    # turns IN4 off
```

# Encapsulate Relay device

```
import RPi.GPIO as GPIO
import sys

class Relay:
    def __init__(self):
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        self.pinList = [22,23,24,25]

    def switch(self, lamp, state):
        pin = self.pinList[lamp]
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin,
           (GPIO.HIGH if state == 0 else GPIO.LOW))
```

# Using/Testing Relay class

```
def main():
    if len(sys.argv) > 2:
        [cmd,lamp,state] = sys.argv
        lamp = int(lamp)   # which lamp?
        state = int(state) # off/on?
        relay = Relay()
        relay.switch(lamp, state)
    else:
        print "Usage: %s <relay> <0/1>" % sys.argv[0]

if __name__ == "__main__":
    main()
```

# GPIO operations (input)

- Input mode

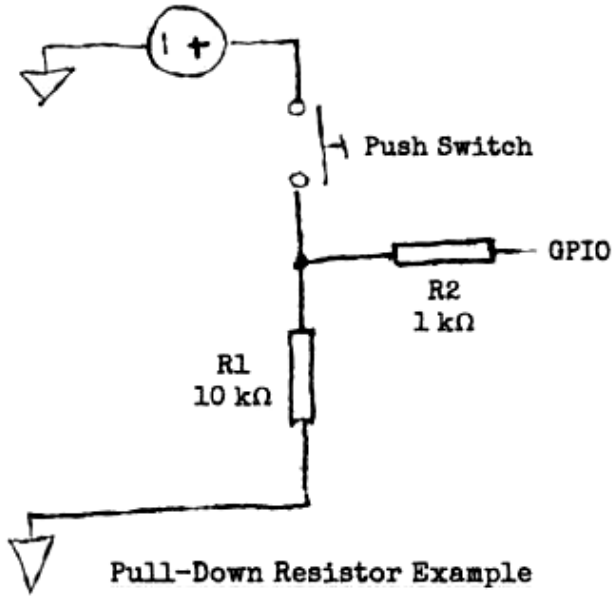
```
GPIO.setup(pin, GPIO.IN)
```

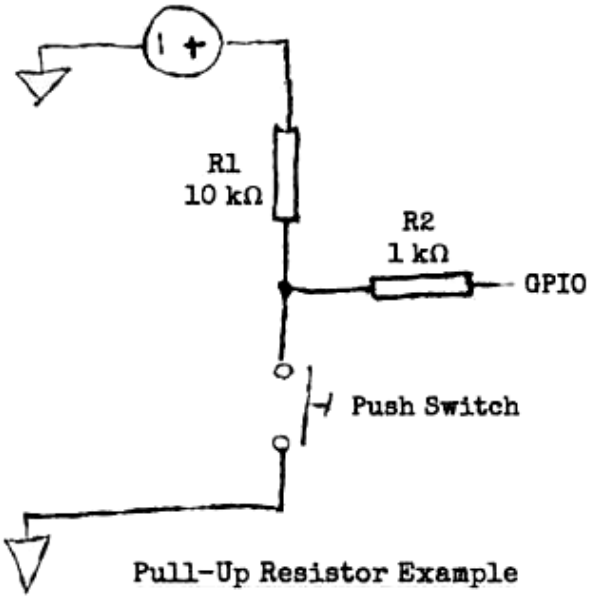
- Read pin (after setting low/high potential at pin)

```
GPIO.input(pin)
```

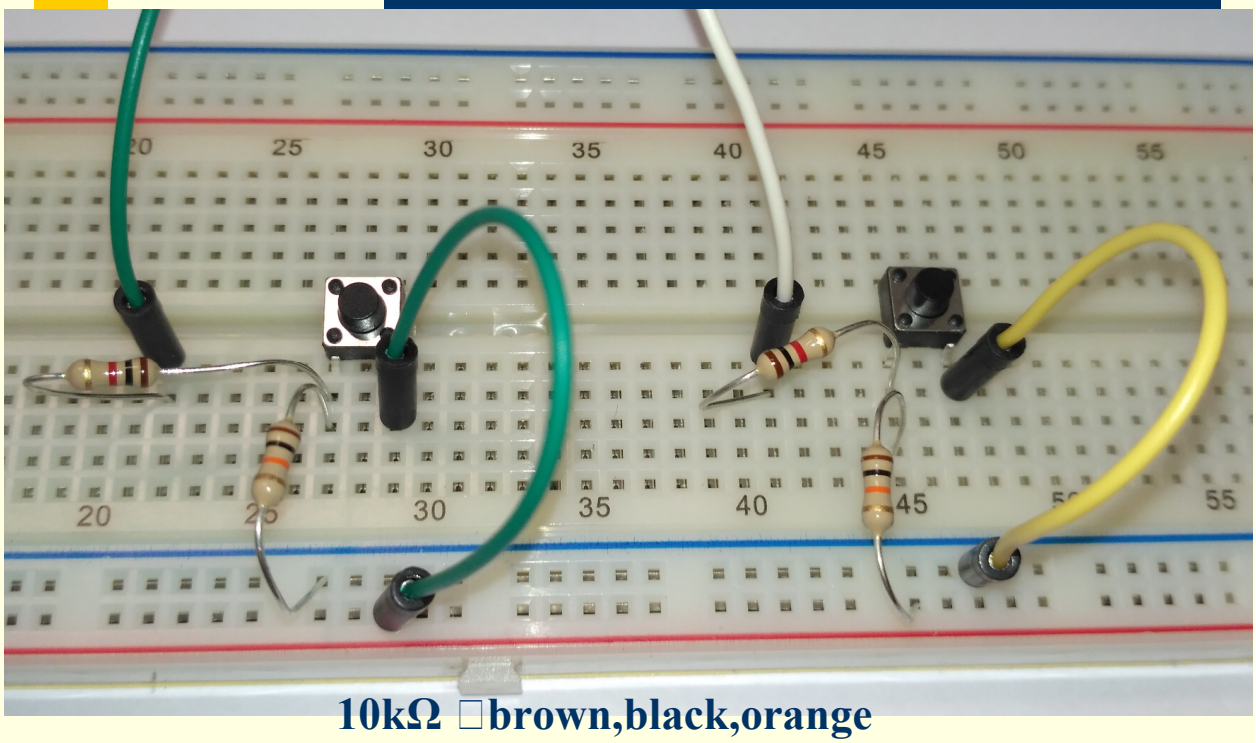
# Setting GPIO voltage level



# Push switch on Breadboard




**10kΩ □brown,black,orange**

**1kΩ □brown,black,red**

# Wiring Switch GPIO

| Switch setup pins | RP pins |
|---|---|
| VCC | 2 (+ve) |
| OUT | 11 (GPIO 17) |
| GND | 6 (ground) |

# Encapsulating Switch (or Sensor)

```
import RPi.GPIO as GPIO
import time

class Switch:
    def __init__(self, pin):
        self.pin = pin
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.pin, GPIO.IN)

    def getState(self):
        return GPIO.input(self.pin)
```

# Using/Testing Switch class

```
def main():
    sw = Switch(17)
    state = sw.getState()
    while True:
        time.sleep(1)
        r = sw.getState()
        if (r != state):
            state = r
            print "%s: status is %d" %
                    (time.asctime(), state)

if __name__ == "__main__":
    main()
```

# Edge detection

```
import RPi.GPIO as GPIO
import time

class Switch:
    def __init__(self, pin):
        self.pin = pin
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.pin, GPIO.IN)

    def getState(self):
        return GPIO.input(self.pin)

    def waitFor(self, event):
        GPIO.wait_for_edge(self.pin, event)
```

# Using/Testing Sensor class

```
def main():
    sw = Switch(17)
    state = sw.getState()
    while True:
        sw.waitFor(GPIO.RISING);
        print "Sensor is %d" % (sw.getState(),)
        sw.waitFor(GPIO.FALLING);
        print "Sensor is %d" % (sw.getState(),)

if __name__ == "__main__":
    main()
```

# Event-driven

```
class Switch:

    def onStateChange(self, channel):
        sys.exit("should have overriden: onStateChange()")

    def setEvent(self, event):
        GPIO.add_event_detect(self.pin,
                              event,
                              callback=self.onStateChange)
```

# Using Event-driven via subclassing

```
class SwitchEvent(Switch):

    def __init__(self, pin):
        Sensor.__init__(self,pin)

    def onStateChange(self, channel):
        print "%s: pin: %d state: %d"
                % (time.asctime(), channel, self.getState())

def main():
    sensor = SwitchEvent(17)
    sensor.setEvent(GPIO.BOTH)
    while True:
        time.sleep(1)
```

# Summary

- Relay may turn on/off variety of equipment
  - Turn on lights/air-conditioner/heater
  - Turn on alarm
  - Release door lock
- Sensor output is binary 0/1 representing binary state
  - Open window
  - Broken lamp
  - Over-heating, dangerous pressure