Imperial College London

Lecture 10

Motor Drive, Polling and Interrupt

Prof Peter YK Cheung

Dyson School of Design Engineering

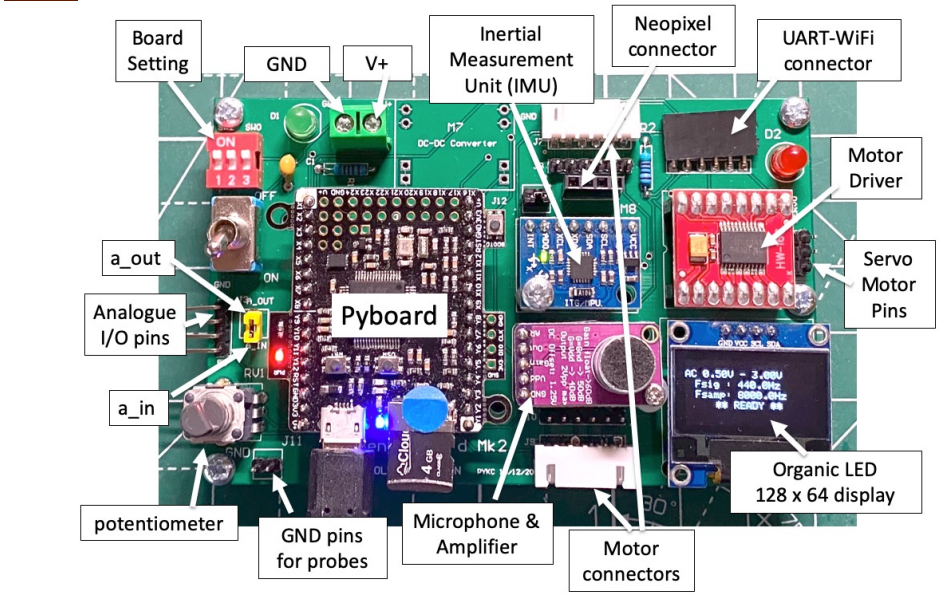
URL: www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/
E-mail: p.cheung@imperial.ac.uk

In this lecture, we will learn about the processor board on Pybench, how it works with the motor driver module, and how timers are used to produce PWM signals to control the speed of the motors.

You will learn why interrupts are important and useful. You will also learn how to programme Pybench to handle them.

Interrupts are generated from some hardware sources. In our case, these are generated using the on-chip timers. In this lecture, we will also have a quick look at the timers built into the microcontroller.

Pybench Board and its components



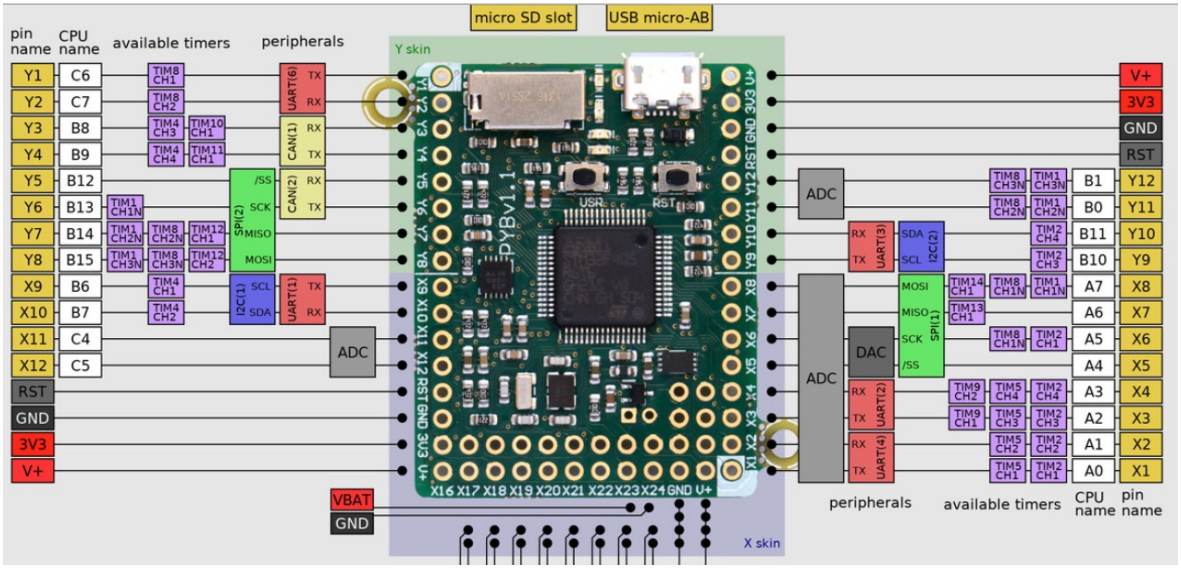
PYKC 18 Feb 2026 DE2 – Electronics 2 Lecture 10 Slide 2

The Pybench hardware contains many modules. The heart of these is the **Pyboard**, which is a microcontroller board designed by the person who wrote MicroPython.

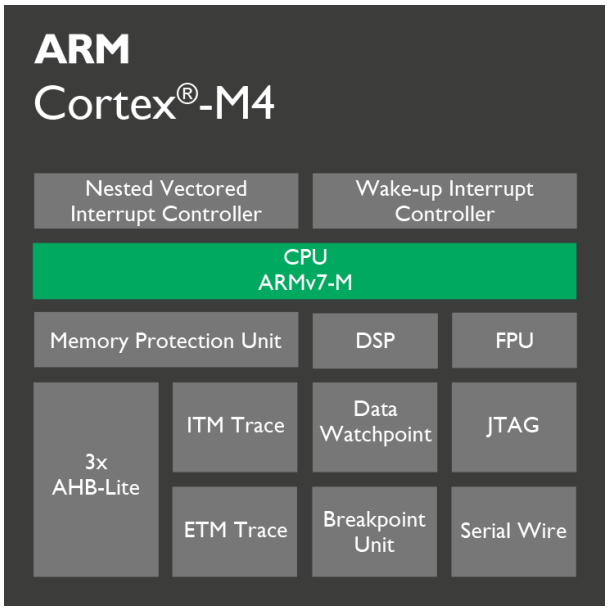
You have already used the microphone/amplifier module, the IMU module and the OLED display module in previous Labs.

However, so far we have not considered what is inside the **Pyboard**. This microcontroller is a much more powerful version than the Arduino you used for Gizmo. The instruction set architecture (**ISA)** is the same for both – they both use the ARM processor architecture. However, the microcontroller we use here is significantly faster in clock speed and includes much better built-in peripherals.

Here is the pins that are brought out of the Pyboard to interface to various modules on the Pybench hardware.

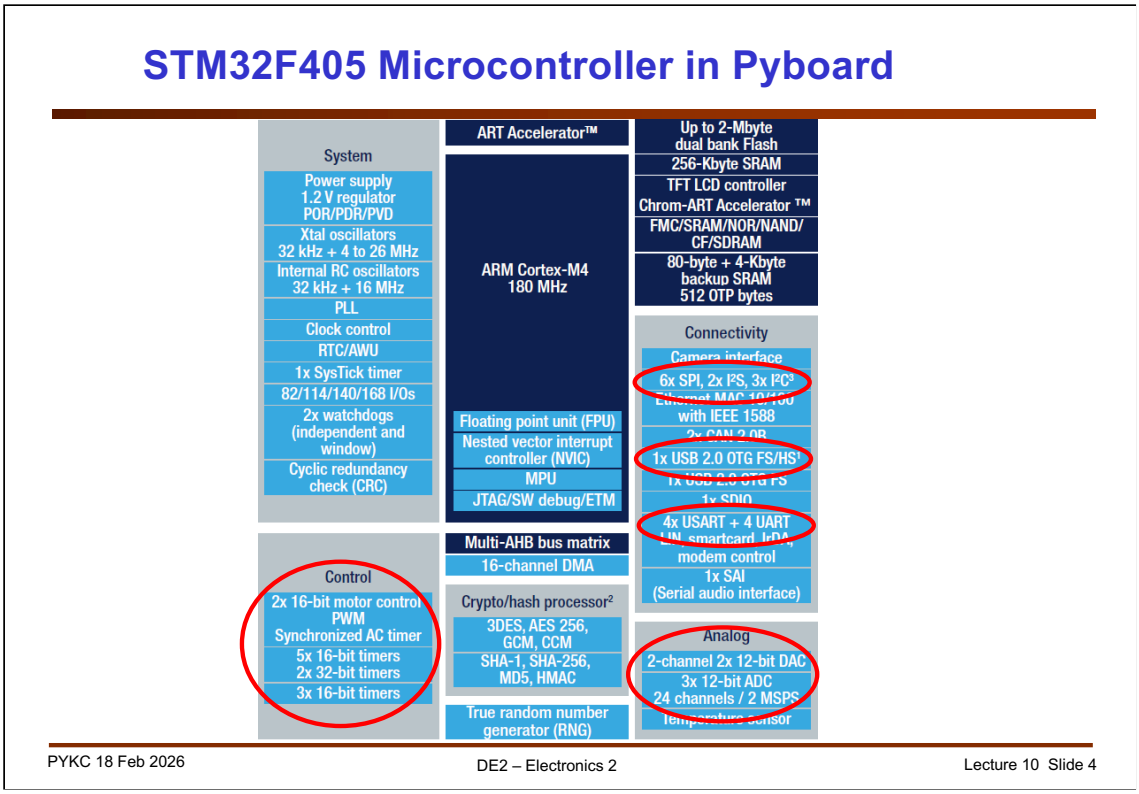


ARM Cortex-M4 Processor



The microprocessor inside the **Pyboard** is known as ARM Cortex-M4. The central part of this processor is the **ARM 7 CPU**. However, this ARM core (as it is called) is more than just the CPU. Surrounding the CPU are also many other useful modules that makes the ARM much easier to use.

For example, it contains various instruments in order to capture various data go to and from the ARM processor. It has protection circuits and other digital circuitry to makes the interface between the CPU and everything external to it much easier. The most important here is the 3x AHB (Arm Higher Performance Bus) which allows the CPU to talk to the peripheral devices that attach to it.



We are not only just using the ARM CPU. Instead, the Pyboard has a microcontroller as its main engine. This microcontroller, the STM32F405 is made by ST Micro, and here is an overview block diagram for this chip.

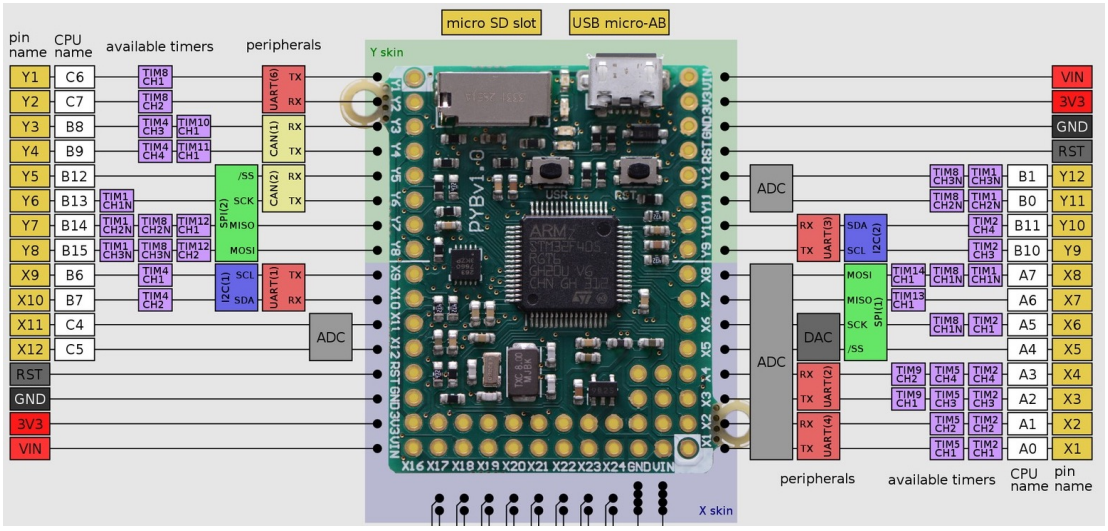
The ARM Cortex-M4 (which has all the stuff from the last slide) is only a small part of the entire chip – it is shown in dark blue here.

The light blue parts are added to the ARM core by ST Micro. These provide many useful functions. The ones that we use for this course are:

- Many timers for PWM signal generation, sampling clock and other general timing functions
- USB interface to communicate with your laptop
- ADC and DAC for analogue signal capture and output
- I2C interfaces for IMU and OLED
- UART interface for WiFi module.

It is interesting to note that ARM does not make chips. They provide the ARM-Cortex M4 design to ST Micro (who pays a royalty to ARM for the IP). ST Micro then design all the other stuff around the ARM core to make their product. ARM was the first company to make this fabless IP business model successful. There are many times more ARM processors currently being used in the world than Intel x86-based processors.

The Pyboard



The Pyboard further add to the ST Micro chip by adding an accelerometer, a MicroSD card reader, all sort of power regulation and protection circuits.

This is a **QuickRef Guide** from MicroPython. It shows which pins on the Pyboard is used for what. Most of them are programmable, meaning that they have multiple functions. We enable them for a specify function as required. For example, you may use UART(6) for something and this would use pins Y1 and Y2. If you are not using these pins for UART, you could program the pins to drive LEDs or one of the timer pins.

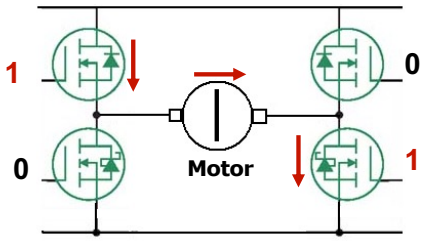
For this course, you will only be using limited features on the Pyboard and the STM microcontroller. Labs 4 to 6 teach you how to program the Pyboard to do things that are useful for the Segway Challenge.

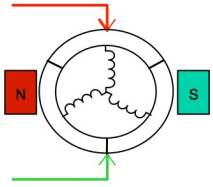
PIN	FUNCTION
X1	Motor PWM_A/Servo 1
X2	Motor PWM_B/Servo 2
X3	Motor control AIN1/Servo 3
X4	Motor control AIN2/Servo 4
X5	Analogue OUTPUT
X6	SW2
X7	Motor control BIN1
X8	Motor control BIN2
X9	IMU-I2C SCL
X10	IMU-I2C SDA
X11	POT10K
X12	Analogue INPUT

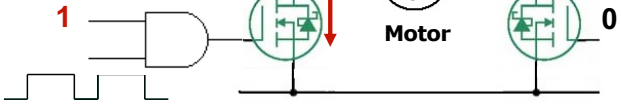
PIN	FUNCTION
Y1	DT-06 Tx
Y2	DT-06 Rx
Y3	SW1
Y4	Motor sensor A_A
Y5	Motor sensor A_B
Y6	Motor sensor B_A
Y7	Motor sensor B_B
Y8	SW0
Y9	OLED-I2C SCL
Y10	OLED-I2C SDA
Y11	Microphone amplifier
Y12	NEOPIXEL

Driving a DC Motor – H-Bridge

- The DC motor needs four transistors to control its speed and direction.
- In Lab 5, we used the TB6612 chip to drive the motor with four transistors.
- The combination of transistors is called an H-Bridge, due to the obvious shape. (See diagram.)
- Transistors are switched diagonally to allow DC current to flow in the motor in either direction.
- The transistors can be Pulse Width Modulated to reduce the average voltage at the motor, useful for controlling current and speed.







Since motor coils are essentially inductors, they have low DC impedances (resistance of the wiring). Hence when driving motors, we need to use a special driver chip.

The driver chip you will use in Lab 5 (the TB6612) is often called the H-Bridge Driver. Shown here is the simplified block diagram. There are four transistors connected to the supply rail and ground. The motor is connected in the middle forming the horizontal link of the H. The transistors are MOSFETs (metal oxide silicon field effect transistors) which are acting like a voltage-controlled switches. When a '1' or high voltage is applied to the gate control terminal, the transistor turns ON and conduct electricity. If a '0' or low voltage is applied, the transistor is OFF.

The top diagram shows a configuration that results in the supply voltage being applied to the left terminal of the motor. The right terminal of the motor is grounded, and the motor turns in one direction. Reversing the control to the transistors results in the motor turning in the other direction.

If you use an AND gate at the control input, you can also add a PWM signal to control the speed of the motor.

Basically the '1' and '0' control signals are the A0 and A1 signals on the TB6612. The PWM signal is what you apply to the input of the AND gate.

Now you know how the TB6612 works.

Driving the motor with TB6612

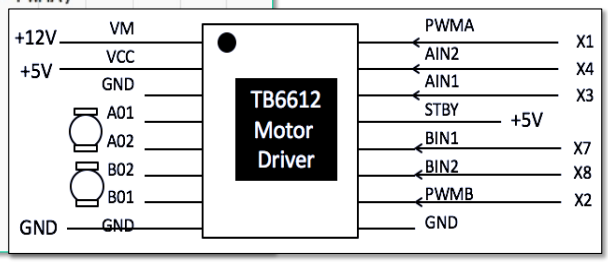
```
import pyb
from pyb import Pin, Timer

# Define pins to control motor
A1 = Pin('X3', Pin.OUT_PP)      # Control direction of motor A
A2 = Pin('X4', Pin.OUT_PP)
PWMA = Pin('X1')                # Control speed of motor A

# Configure timer 2 to produce 1KHz clock for PWM control
tim = Timer(2, freq = 1000)
motorA = tim.channel (1, Timer.PWM, pin = PWMA)

def A_forward(value):
    A1.low()
    A2.high()
    motorA.pulse_width_percent(value)

A_forward(50)
```



Exercise 1 of Lab 5 is just a revision from last year's Electronic 1 module. However, the TB6612 is NOT the same as the motor driver DRV8866 you used last year. TB6612 has two signals AIN1, AIN2 to control direction, and a separate PWMA signal to control the speed.

Here are some interesting questions to ask yourself to check whether you have learned what is expected of you:

1. Why do you need this driver chip at all? Could you drive the motor directly from the microprocessor?
2. How are the two pins (IN1 and IN2) used to control the direction of the motor?
3. What is PWM and why is it desirable to use PWM to control the speed of the motor instead of using analogue voltage level (e.g. from a DAC signal)?
4. What is meant by "Creating a pin object A1" in the Python code?

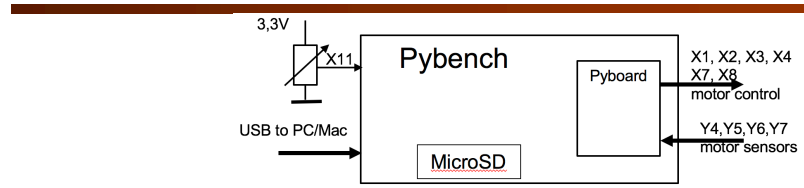
```
A1 = Pin('X3', Pin.OUT_PP)
```

5. Explain how timer 2 is programmed to produce the PWM signal to drive motor in the following lines.

```
# Configure timer 2 to produce 1KHz clock for PWM control
tim = Timer(2, freq = 1000)
motorA = tim.channel (1, Timer.PWM, pin = PWMA)
```

6. How should you choose the frequency of the PWM signal to drive the motor?

Controlling the speed with potentiometer




```
pot = pyb.ADC(Pin('X11'))      # define potentiometer object as ADC conversion on X11
value = pot.read()             # value = 0 to 4095 for voltage 0v to 3.3v
```

```
while True:                  # loop forever until CTRL-C
    speed = int((pot.read()-2048)*200/4096)
    if (speed >= 0):         # forward
        A_forward(speed)
        B_forward(speed)
    else:
        A_back(abs(speed))
        B_back(abs(speed))
```

PYKC 18 Feb 2026 DE2 – Electronics 2 Lecture 10 Slide 8

Next, we use the potentiometer (5kΩ) to control motor speed and direction. Here are the questions to test yourself:

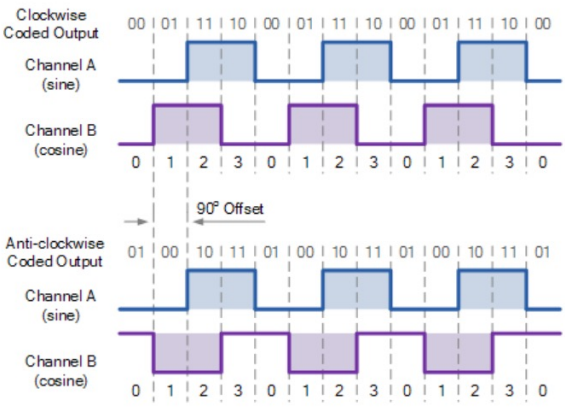
1. In Micropython, how do you create an object to perform ADC conversion? Why in this case, we use pin X11?
2. How do you define and work out the resolution of the ADC converter?
3. Explain the meaning of the statement:

```
speed = int((pot.read()-2048)*200/4096)
```

4. Explain the meaning of the format statement in Python:

```
oled.draw_text(0,40,'Motor Drive:{:5d}%'.format(speed))
```

Measuring Motor speed with Hall Effect Sensors



- Circular magnet has 13 pole pairs
- The gearbox of the motor has a 1:30 gear ratio
- How many pulses are produced for each revolution of the motor?
- Speed of motor (in rps) can be measured by counting the number of pulses in a given time window (say 100msec)

```
# Define pins for motor speed sensors
A_sense = Pin('Y4', Pin.PULL_NONE)  #  Pin.PULL_NONE = leave this as input pin
B_sense = Pin('Y6', Pin.PULL_NONE)
```

PYKC 18 Feb 2026 DE2 – Electronics 2 Lecture 10 Slide 9

Next, we use the Hall Effect Sensors (two) on the motor to determine the speed of the motor and direction of the motor. The questions to ask yourself are:

1. Refer to the sensor output signals, what happens when you increase the motor speed?
2. How would the two sensor signals differ when you change the direction of rotation in the motor?
3. Given the waveform of the two signals (Channel A and B) from the sensors, the relative phase is always $\pm\pi/2$. Why?
4. Given the circular magnet has 13 pole pairs, and that the gear of the motor has a 1:30 reduction ratio, how can you derive the speed of motor (in revolutions per second) from the number of rising edges E in a period T? (answer: 390 pulses per revolution. Therefore, the speed of motor is:

motor_speed (in rps) = (number of pulses/390) / T in seconds

Pseudo code to measure speed by polling

- Initialize variables to zero: motor_speed, sensor_state, pulse_count
- Repeat forever:

```
Mark current time  (as tic)
If sensor has gone from low to high (rising edge)
        increment pulse_count
Update sensor_state by reading hall effort sensor value
If elapse_time >= 100ms
        motor_speed = pulse_count
        reset pulse_count
        display speed on OLED as motor_speed/39
```

Discuss: what is the limitation of polling?

This is typically how one can measure the motor speed by polling – continuously checking in a tight loop whether something has happened or not.

In the code above, there are TWO polling operation happening. The first if-statement checks to see if the Hall sensor signal has a rising edge (goes from low to high). The second if-statement checks for a time window of 100msec. By counting the number of pulses detected in 100ms window, we can calculate the speed of the motor using the formula:

motor_speed (in rps) = number of pulses/39

39 because each revolution of the motor generates 390 pulses. Therefore in a 100msec period, one revolution will give us 39 pulses!

Measure motor speed by polling

- Polling means checking for some event in a loop, then do something
- Here we check sensor signal of motor A changing from low to high in the polling loop
- When this occurs, increment a counter A_count
- We also check elapsed time = 100msec in polling loop (tic-toc)
- If time out, save count as speed measurement A_speed, and reset counter

```
# Initialise variables
A_state = 0         # previous state of A sensor
A_speed = 0         # latest speed of motor A
A_count = 0         # positive transition count
tic = pyb.millis(); # keep time in millisecond

while True:             # loop forever until CTRL-C
    # detect rising edge on sensor A
    if (A_state == 0) and (A_sense.value()==1): # rising edge detected on A
        A_count += 1
    A_state = A_sense.value()   # read value on pin A_sense

    # Check to see if 100 msec has elapsed
    toc = pyb.millis()
    if ((toc-tic) >= 100):
        A_speed = A_count

        # drive motor - controlled by potentiometer (as before)
        .........

        A_count = 0         # reset transition count

    # Display new speed
        oled.draw_text(0,20,'Motor A:{:5.2f} rps'.format(A_speed/39))
        oled.display()
        tic = pyb.millis()
```

PYKC 18 Feb 2026 DE2 – Electronics 2 Lecture 10 Slide 11

We measure the speed of rotation by counting the number of low-to-high transitions on one of the two Hall sensor signals.

This can be achieved by polling – checking in the code when such transition has occurred. If yes, up a counter value. Then check if 100msec has elapsed. If yes, remember the count value and reset the counter.

Questions to ask yourself:

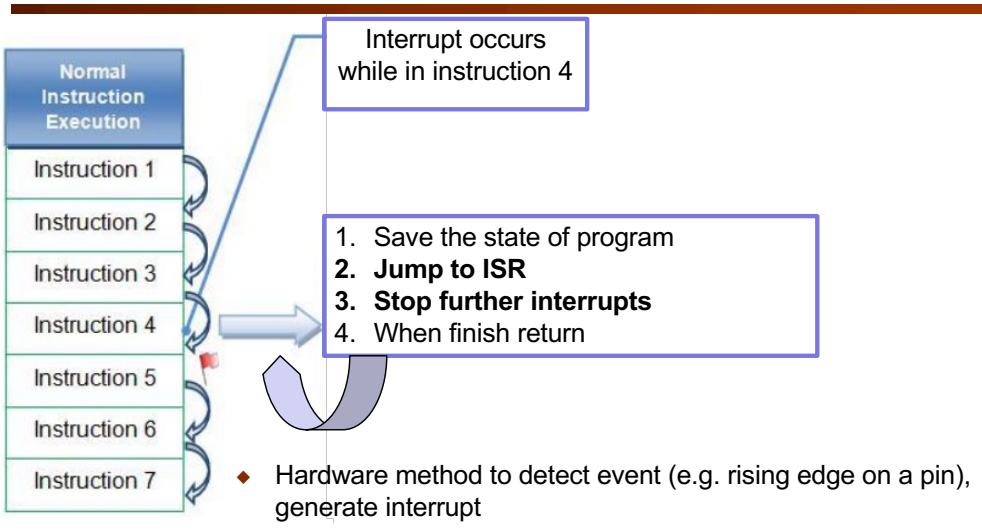
1. What is the purpose of these two lines?

```
if (A_state == 0) and (A_sense.value()==1): # rising edge detect
    A_count += 1
A_state = A_sense.value()   # read value on pin A_sense
```

2. How are tic and toc, which are built-in functions in Matlab, be implemented in Micropython?
3. Explain the following codes:

```
# Check to see if 100 msec has elapsed
toc = pyb.millis()  3
if ((toc-tic) >= 100):
    A_speed = A_count
```

Lab 5: The idea of interrupt



- Hardware method to detect event (e.g. rising edge on a pin), generate interrupt
- Processor forced to do something else – defined in the Interrupt Service Routine (ISR)
- Return when finished

The reason why polling is not a good method to measure speed of motor is that microprocessor can only execute ONE instruction stream at a time. If you are checking (polling) for rising edge, you cannot do other things. Conversely if you are doing other things, you will miss the rising edges. That's why in the experiment, you found that the polling method gives a speed reading that is "noisy", meaning that it is jumping all over the place!

Interrupt is different. You use HARDWARE method to detect the occurrence of an event. This forces the processor to suspend whatever it is doing at the time, and go to another segment of the code to service the interrupt (hence we call this the "Interrupt Service Routine" or ISR).

When finished, return to the interrupted code and continue as before.

Question to ask yourself:

1. Why is interrupt better than polling?
2. What happens if your interrupt service routine is long and complex?
3. How should you think about a system with multiple interrupts?
4. What is it meant by "saving the state fo the program"? Why is this necessary?

Lab 5: Interrupt Service Routines

- Need to detect and handle two types of events:
  1. Rising edge on Hall effect sensor signal on Y4
  2. 100ms elapsed time on a Timer
- Need two ISRs for these two interrupt events
- Need to provide a dummy variable as shown here

```
#-------  Section to set up Interrupts ----------
def isr_motorA(dummy):  # motor sensor ISR - just count transitions
    global A_count
    A_count += 1

def isr_speed_timer(dummy):     # timer interrupt at 100msec intervals
    global A_count
    global A_speed
    A_speed = A_count           # remember count value
    A_count = 0                 # reset the count
```

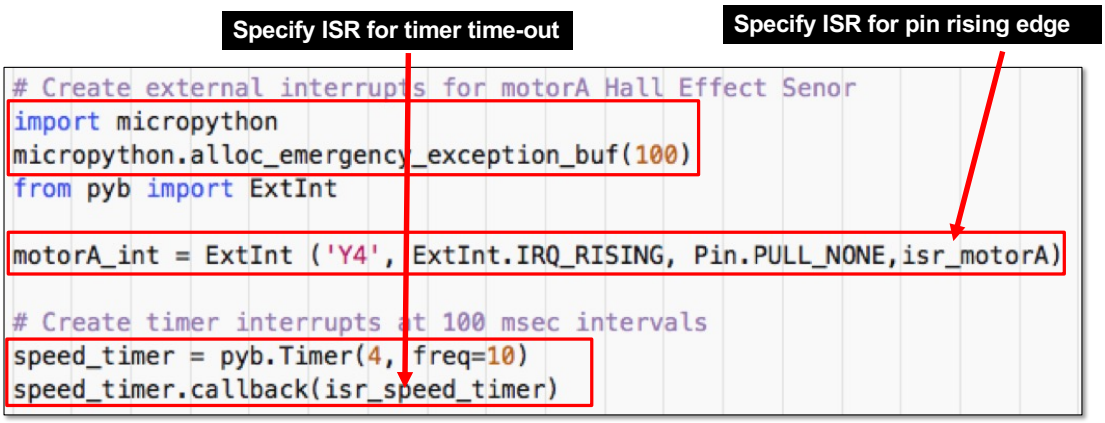

Here are two interrupt service routines. The first to handle low-to-high transition on the senor signal from Motor A. The second to handle timer alarm which happens every 100msec.

Question to ask yourself:

1. When will the functions isr_MotorA and isr_speed_timer be executed?
2. What are the purposes of these two functions?
3. Why you need to define A_count and A_speed as global?

Lab 5: setting up the interrupts

- Allocate some buffer space to handle errors
- Specify Pin Y4 as source of interrupt, rising edge
- Define timer 4 as a 100msec period timer (10Hz)
- **timer.callback** (ISR) - tell timer to generate an interrupt at end of period, and execute ISR



How does one set up interrupts in MicroPython using the Pyboard and the Pybench System? First you need to include the following statement to allocate memory to store the state of the program:

```
micropython.alloc_emergency_exception_buf(100)
```

Then you have to tell that hardware that pin Y4 will generate an interrupt on every rising edge, and that the interrupt service routine is isr_motorA:

```
motorA_int = ExtInt ('Y4', ExtInt.IRQ_RISING, Pin.PULL_NONE,isr_motorA)
```

Then, you need to program Timer 4 to time out every 100msec:

```
speed_timer = pyb.Timer(4, freq=10)
```

Finally, you need to tell this Timer that it should generate an interrupt when time out, and run isr_speed_timer:

```
speed_timer.callback(isr_speed_timer)
```

Lab 5 – Interrupt MAGIC

```
while True:             # loop forever until CTRL-C

    # drive motor - controlled by potentiometer
    speed = int((pot.read()-2048)*200/4096)
    if (speed >= 0):        # forward
        A_forward(speed)
        B_forward(speed)
    else:
        A_back(abs(speed))
        B_back(abs(speed))

    # Display new speed
    oled.draw_text(0,20,'Motor A:{:5.2f} rps'.format(A_speed/39))
    oled.display()
```

Wheel rotating at 1 rps will produce 39 rising edges in 0.1 sec

- Program loop assumes A_speed has the correct value!
- There is no reference to 100ms time window, nor counting of edges.

Once interrupt is set up properly, the main program loop only controls the motor. Measuring the speed of motor is done automatically.

The global variable A_speed will contain the correct number of transitions in a 100msec window ALL THE TIME, and updated every 100msec automatically.

Three Big Ideas

1. **PWM** is the efficient way to drive **motors** or **LEDs**. The **H-bridge** motor driver allows PWM signal to control the speed with separate digital signals to control the direction of the motor.
2. **Interrupt** is a much better way of detecting hardware events than using **polling** method.
3. Interrupt makes software hard to debug because once set up, it runs in the background all the time and is difficult to stop. So **make interrupt service routine** as **simple** as possible.

How does one set up interrupts in MicroPython using the Pyboard and the Pybench System? First you need to include the following statement to allocate memory to store the state of the program:

```
micropython.alloc_emergency_exception_buf(100)
```

Then you have to tell that hardware that pin Y4 will generate an interrupt on every rising edge, and that the interrupt service routine is isr_motorA:

```
motorA_int = ExtInt ('Y4', ExtInt.IRQ_RISING, Pin.PULL_NONE,isr_motorA)
```

Then, you need to program Timer 4 to time out every 100msec:

```
speed_timer = pyb.Timer(4, freq=10)
```

Finally, you need to tell this Timer that it should generate an interrupt when time out, and run isr_speed_timer:

```
speed_timer.callback(isr_speed_timer)
```