Dyson School of Design Engineering

Imperial College London

DE2 Electronics 2

Lab Experiment 5: Motor & Interrupt

(webpage: http://www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/)

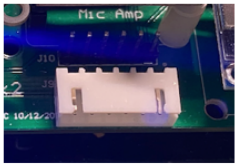
Objectives

By the end of this experiment, you should have achieved the following:

- Learn how to drive the dc motors using the H-bridge driver chip TB6612
- Control motor speed with potentiometer via ADC
- Use hall effect sensors to detect speed of motor using polling and interrupt
- Characterize the two motors' speed against PWM duty cycles

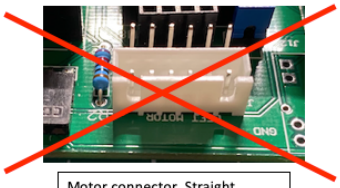
Before you start

In this lab, you will be using the Pybench board, the motor assembly that you will be using later with the Segway Challenge. There is no need to assemble the Pybench board to the motor assembly and do not install the wheels. You may want to put a tape on the axle of the motors so that you can see it spinning.



Motor connector
Right angle - Best

Motor connector Straight
Notches inward facing - OK



Motor connector Straight
Notches outward facing - Wrong

Connect the ribbon cables between the Pybench board and the motors provided. Make sure that Pybench configuration switch is set at '000' (user mode). From now on, we will be using the Pybench board on its own without tethering it to Matlab.

IMPORTANT NOTE: Check the orange module on Pybench board. It should have the label HW-166 on it. If not, you have the wrong motor drive module installed – I will have to change your motor driver module.

The SD card is preloaded with my version of user.py, which is the solution to Lab 5. You could run this to test that the motors are working properly. You may choose to save a copy of this for checking purposes.

Create your own user.py program with this single python instruction:

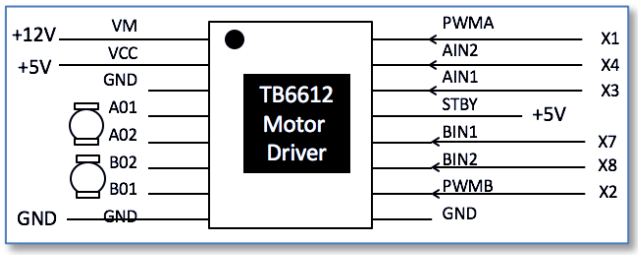
execfile('lab5task1a.py')

You can modify the .py file to run other user created programs.

Task 1: DC Motor and H-bridge

The goal of this task is to remind you how to drive a DC motor using the H-bridge chip TB6612. You have already used a similar chip last year in electronics 1. However, this chip is slightly different and is capable to drive a 12V motor if required.

The task in hand is to use the potentiometer on Pybench to control the speed of the two DC motors using the H-bridge. The interface connection between the TB6612 and the Pyboard module is shown here. Unlike last year, you don't have to wire up the chip this year. All necessary connections have already been made for you on the Pybench board PCB.



The direction of the motor is controlled according to the table shown here. The speed of the motor is determined by the duty cycle of the PWM signal.

IN1	IN2	Action
L	L	Stop
L	H	Counter Clockwise – controlled by PWM

Connect Pybench to your PC using a USB cable. Open a terminal window using PuTTY.exe or, if you are using Mac, use Mac's terminal program. With the VSC editor, create a file: **lab5task1a.py** on the SD card containing the following uPy code. Change the **user.py** file on the SD card to the statement: **execfile('lab5task1a.py').**

```
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)
```

Now do a "soft" reboot by typing CTRL-D if you see >>> in the terminal window. One of the two motors should now be turning. Change the speed of the motor. (How?)

Now modify this program to provide two more motor control functions: **A_back(value)** and **A_stop()**. Test that these functions are working.

Note that we use an on-chip timer circuit inside the microcontroller to generate a 1000Hz PWM signal to drive the motor. To understand how to programme this timer circuit in uPy, read, the following document page: https://docs.micropython.org/en/latest/library/pyb.Timer.html.

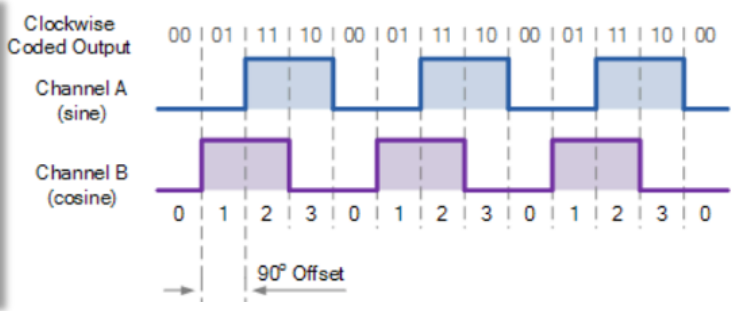
Write a new version of the program to drive **BOTH** motors (instead of just motor A). Note that you can use channel 2 of Timer 2 to control the second motor. (See MicroPython manual.)

Finally, create **lab4task1b.py** which uses the 10kΩ potentiometer to control the motors to go forward and backward at various speed, up to the maximum. The potentiometer is connected to pin 'X11'. To read the potentiometer voltage, you need the following uPy code:

```
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
```

Modify your program into **lab4task1c.py** to display the PWM duty cycle on the OLED display.

Task 2 – Detect the speed of the motor



Each motor is equipped with two Hall effect sensors, which detect changes in magnetic field strength as shown in the photograph. The circular magnet has 13 pairs of N-S poles. The gear ratio of the motor is 1:30. This causes the hall effect sensors to produce 13 x 30 = 390 square pulses per revolution of the wheel. Since the two sensors are slightly offset from each other, the square waves from the two sensors are at different phase angles.

Appendix A is a summary of the hardware pins on the Pyboard, and how they are used to connect to the hardware modules and motors etc.

The two Hall effect sensors generate two square waves that are offset by 90°. The frequency of the square waves is proportional to the speed of rotation. The relative phase of the two sensor signals provides the direction.

You can detect the speed of the motor by **counting the number of positive transitions** (low-to-high) on Y4 or Y5 in a time period of 100 msec, and divide the count value by 39 to obtain the number of revolutions of the wheel per second (rps).

You can also detect the direction of rotation of the motor by observing whether Y4 is LEADING Y5 in phase, or vice versa. However, since you are driving the motor with your own program, *you know* which direction the motor is turning. So, this is not so useful.

To count the pulses on pin Y4 for motor A and Y6 for motor B, you would need to define two Pin objects:

```
# 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)
```

The code segment to detect the speed of motor A is given here:

```
# 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()
```

Note how I have implemented the equivalent of "tic" and "toc" from Matlab using Pyboard's pyb.millis() function. This function returns the internal real-time clock in millisecond. By keeping tic and toc variables, elapsed time can be calculated as (toc-tic).

Modify **lab5task1c.py** to a new program **lab5task2a.py** to detect the speed of motor A.

Test the program to check the speed of motor A (in rev/sec). Note that the reported speed is very "noisy". Why?

Further, add the following statement to add a delay of 1ms to the loop and see what happens. You will find that the speed reading is all wrong. Why?

```
while True:             # loop forever until CTRL-C
    # Do something for 1ms
    pyb.delay(1)
```

Explanation

The while-loop is continuously looking for a low-to-high transition on motor A sensor signal on pin Y4. Within the loop, you also continuously check to see if 100msec has elapsed. If yes, you save the transition count (in A_speed) and reset the counter. The continual checking program loop is known as "**polling**". It is analogous to owning **a telephone that has NO ringer**. To see if anyone is calling, you need to "**poll**" the phone by picking it up and check to see if someone is one the line! This is a simple way to check, but it is extremely inefficient.

Adding just 1msec delay in the loop results in many rising edges being missed. This is the reason why the speed measurements are wrong.

Remove this spurious statement, and modify your program into **lab5task2b.py** so that you measure and display the speed of **both** motor A and motor B.

Task 3 – Speed measurement using interrupt

Instead of using **polling** as a method to check the status of the Y4 pin, you could install a "ringer" in the form of interrupts. **Interrupt** is a hardware feature on the microprocessor, which forces the processor to do something for you when an event occurs. For example, you can set up the Y4 pin in such a way that when a low-to-high transition occurs, a special function known as the "**interrupt service routine**" or **ISR** will be executed. The ISR runs the program code that deals with whatever the interrupt demands. When the ISR is completed, the processor returns to the original code and continues its execution. This is like having a ringer on your telephone. You may be in the middle of a meal and the phone rings. This interrupts you eating your meal and forces you to answer the phone. When you finish that, you return to your meal. This is far more efficient than you having to check if someone is calling you at regular intervals! AND you are will NOT miss a call.

You will now learn to program Pybench to work with interrupts. Modify your program **lab5task2a.py** to **lab5task3a.py**, replacing the section after all the motor control functions (i.e. **B_forward**(value), **B_back**(value)) with the code segment shown on the next page.

This code is rather complicated. I will explain how this works at a future tutorial. For now, I want you to focus on the following interesting points:

1. The while-loop is almost the same as that in **lab5task1c**, where the potentiometer is read, and its value is used to drive the motors. There is NO polling function: the sensor signal on Y4 is not used here. There is no checking of elapsed time either.
2. The loop assumes that the variable "A_speed" *magically* contains the number of transitions on Y4 and reports the rotational speed for motor A.
3. The magic is occurring in the "INTERRUPT" section immediately above.
4. There are two **interrupt service routines**: 1) **isr_motorA(.)** and 2) **isr_speed_timer(.)**.
5. **isr_motorA(.)** increments "A_count" each time an interrupt occurs on the Y4 sensor signal.
6. **isr_speed_timer(.)** is called when Timer 4 period is over. Timer 4 is programmed to run at 10Hz, therefore a timer_4 interrupt occurs every 100msec. When this happens, the ISR saves the count value in "A_count" to "A_speed" and reset the counter.
7. A_count and A_speed are declared as global variable inside the ISRs (otherwise it is not visible in the ISRs).
8. The line: **micropython.alloc_emergency_exception_buf(100)** is required by MicroPython.

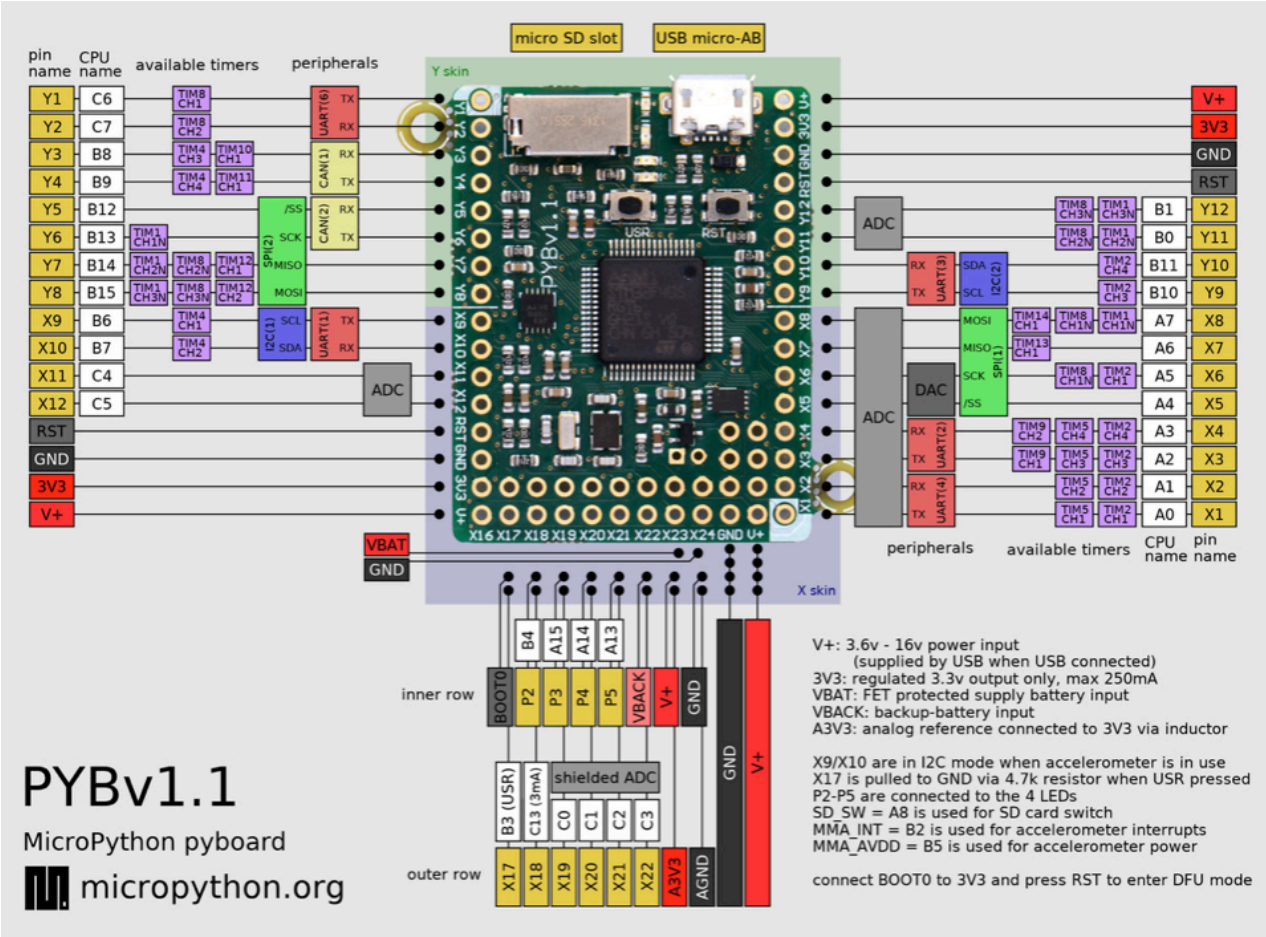
```
# Initialise variables
speed = 0
A_speed = 0
A_count = 0

#-------  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

# Create external interrupts for motorA Hall Effect Senor
import micropython
micropython.alloc_emergency_exception_buf(100)
from pyb import ExtInt

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

# Create timer interrupts at 100 msec intervals
speed_timer = pyb.Timer(4, freq=10)
speed_timer.callback(isr_speed_timer)

#-------  END of Interrupt Section  ----------
```

Modify **lab5task2a.py** to **lab5task3a.py** so that you sense the speed of both motors and report them on the OLED display. Make sure you understand how the entire program works.

Modify **lab5task3a.py** to **lab5task3b.py** so that you sense and display the speed of both motors A and B using interrupts.

Use the program **lab5task3b.py** to find out the rev/sec of motors A and B for different PWM duty cycles. Plot the characteristics of motor speed vs duty cycle for the two motors. What can you say about these two motors?

Appendix A – Pin Assignment on Pybench



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