Imperial College London

Design Engineering 1.3 – Electronics

Laboratory Experiment 4

Drive, Sense and Link

Professor Peter Y K Cheung
Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/
E-mail: p.cheung@imperial.ac.uk

In this experiment, you will learn all sorts of things that will help you in your Team Project. It is therefore NOT TO BE MISSED unless you are not well or have unavoidable reasons. If you do have to miss this Lab, make sure that you catch up another time – you will learn loads in this one single Lab session.

Learning Outcome

- This Laboratory session is designed to prepare you for the Team Project.
- The intention for this Lab is that, by the end of the Lab Session, you should be familiar with the following:
  1. To program the PyBoard in python using your PC as the host.
  2. To control the I/O pins of the PyBoard to provide digital logic levels of 0 and 1 as digital outputs.
  3. **DRIVE** - To use the pulse-width modulation (PWM) feature of the PyBoard counters to control the speed of the dc motor, which is driven by a **motor driver** chip.
  4. To use the counter/timers inside the PyBoard to measure time period.
  5. **SENSE** – to use the infra-red and Hall effect sensors to sense obstacles and presence of a magnetic field.
  6. **LINK** – to use a blue-tooth module with a UART interface in order to communicate between a mobile phone and the Pyboard.
  7. **DO** – to use the various pyBoard libraries provided to do various computation and control tasks.

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 2

It is important for you to appreciate the intended learning outcome for each Lab session, no more so than this Lab. I will be giving you four lectures on the applications of electronics in the next few weeks. They are entitled: **Drive, Sense, Link and Source**. In this Lab, you will experience the first three topics.

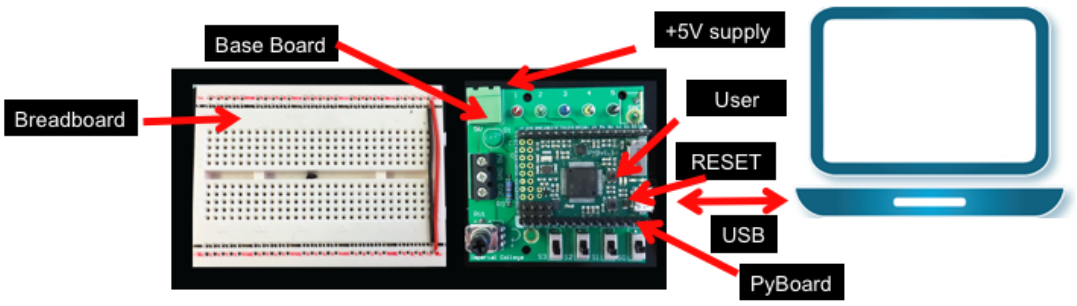
Furthermore, you will be using Python to control things that are physical. Don't worry if you are not good in Python programming yet. You will be learning through examples.

In this experiment, you will be using the hardware board known as the **Pyboard** which at its heart is a processor chip. (Note that this is different from the Raspberry Pi.) This is a fully functioning computer on a small printed circuit board, with lots of different built-in peripherals (meaning it has all sort of electronics NOT part of processor, but would help the processor to do useful things). A microprocessor designed to be embedded inside a piece of equipment, instead of going into a computer system, is called a microcontroller. The one we use is based on the ARM architecture, which is the same one used in almost all smart phones and tablets. The microcontroller chip is designed and made by ST Micro, a European chip company.

The board we use, the **Pyboard**, has one distinctive feature. It has integrated into the board a fully working **Python interpreter**. The beauty of this is: you plug a terminal into the board via the USB cable, you can "speak Python" to the processor right away!

Task 0: The PyBoard and the Breadboard (1)

- Connect the PyBoard on your Black Board (BB) to your PC using a USB cable and to the 5V power supply on the bench.
- A list of components needed for this Lab (as shown below) would have been put in on your bench.
- The PyBoard should now appears as a USB disk drive called PYFLASH under Explorer (MS Windows) or Finder (Mac)



PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 3

From now on, including the Team Project, you will be writing python programs for the Pyboard on your BB. For Lab 4, you need to pick up the following list of components from Peter's Bench in the Lab:

1. TB6612 dual motor driver chip
2. Two light-emitting diodes (LED) in red and green
3. A USB cable
4. An infra-red obstacle sensor module
5. A Hall Effect sensor module
6. A Bluetooth-UART module
7. A small fridge magnet
8. USB cable
9. Two motors mounted on a chassis with attached cable
10. One ten-ways male-female ribbon cable to connect PyBoard to breadboard
11. One servo-motor

For Mac users, linking between your computer and the PyBoard is easy. Plug in the USB cable and open the "terminal.app" program.

Enter the command:

screen /dev/tty.usb*

and the **REPL** prompt >>> will appear in the terminal window.

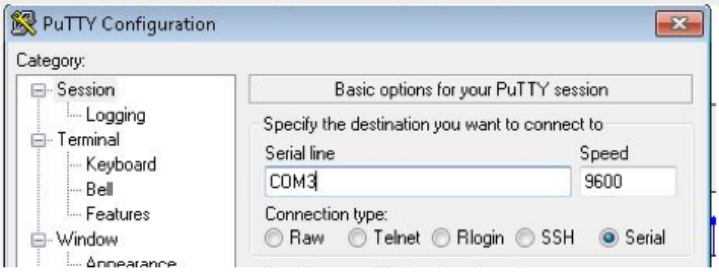
For Windows users, it is more complicated ….

Task 0: Talk to the Pyboard via USB (2)

- **For Mac users**, you only need to open a terminal program "**Terminal.app**" in the Application > Utility folder.
- Once you have a terminal window, enter the command:

  screen /dev/tty.usb*

- **For Windows users**, the PyBoard will appear as a serial PORT device "COMx".
- You may need to install a device driver, which you can find on the PYBFLASH drive. It is called: "pybcdc.inf" unless you are using Windows 10
- To talk to the PyBoard, download and install the program "putty.exe" from the internet.
- You then configure putty to use serial COMx as shown.
- Ask a GTA to help if you have problem configuring your PC to talk to the PyBoard



PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 4

In order to "talk" to the Py**b**oard, you must run a "terminal emulation" program on your host computer, which can be an Apple Mac or a PC. This program simply read the keys you type on the computer as ASCII characters, and send them to the PyBoard via the USB cable. At the same time, it takes the ASCII characters from the PyBoard, and send them to your screen on the computer.

Follow the instruction above, and seek help **immediately** if you are stuck. You cannot do anything else until you get through this stage where you are talking to the PyBoard on your computer.

Task 0: Your first Python Program on Pyboard (3)

- Whenever you are communicating with the Pyboard via a terminal program, you will see:

  >>>

- This is a prompt known as REPL (**R**ead-**E**valuation-**P**rint-**L**oop). Whenever you see REPL, you can simply type Python program code directly and the program will run.
- For example enter the following, and you will see the effect.

```
>>> print("hello pyboard!")
hello pyboard!
```

- Alternative to entering Python codes directly by typing a line at a time, you can also save your codes in a file (say, **main.py**).
- When you power up the Pyboard, it goes through a "boot" sequence ("boot" stands for **bootstrap**).
- It first run the program **boot.py** on Pyboards flash drive, which tells the ARM processor to run the file **main.py**.

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 5

Once everything is working and you are running the terminal program correctly, you will see a welcome message from me. This is followed by the "prompt" message: >>>.

This prompt basically says that Micropython (a version of python designed for small microncontroller with limited memory resources) is now running on the Pyboard waiting for you to type in python codes.

It is important for your to appreciate that everything you do now is **ON THE PYBOARD**. Your PC is merely acting as a dumb terminal, sending characters you type to the Pyboard, and displaying characters from the Pyboard on your screen. Your computer does not do anything intelligent at all! The Pyboard is functioning as a standalone computer, including handling all the python codes you type in.

When you first power up the Pyboard, it performs a start up sequence known as **bootstraping** booting for short). This includes running the program boot.py which is stored in the on-chip flash memory. This appears as a USB drive to your PC. You can edit this boot.py file as any other files on the drive.

The file boot.py has very simple codes, which tells it what to do next. In our case, it runs the program main.py which, at the moment, do nothing but print a message.

Task 0: Handling Emergency (4)

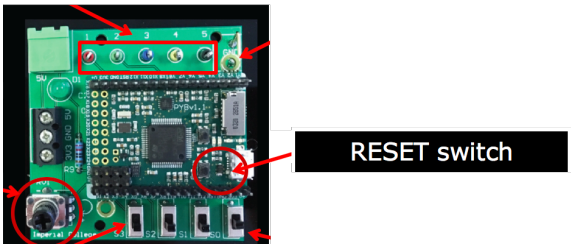
- If your Pyboard hangs, e.g. you cannot get the REPL >>> on your terminal program, you can do the following:
  1. Type CTRL-C to INTERRUPT whatever Pyboard is doing at that time. This will usually get you the >>> back, and you can then control Pyboard.
  2. Perform SOFT RESET by typing CTRL-D. This like using CTRL-ALT-DEL on a PC, and force Pyboard to reboot. This means that Pyboard will run the boot.py program again.
- HARD RESET – this is to be avoided unless absolutely necessary
  - Hard reset is like pressing the reset switch on a PC, and is achieved by resetting the RESET button on the Pyboard (see notes below).
  - Only perform this operation when no LED on the Pyboard is flashing. Otherwise you may corrupt the contents of file stored on the Pyboard Flash Drive
- Disconnect power – the last resort is to remove USB cable to Pyboard. Doing so will loose your terminal (or putty) connection and may corupt files if LED on Pyboard are flashing

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 6

This page is about recovery from errors. Read carefully – failing to do so may waste you lots of time!

Sometimes if you make a mistake, the Pyboard may fail to respond. What do you next? There are three levels of action as explained above.

1. Interrupting Pyboard by pressing CTRL-C is the most useful action. This tells the Pyboard to stop whatever it is doing and return control to you. If this works, you should see >>> on the terminal screen. Typically you would then edit your Python program, correct your mistakes, and restart by typing CTRL-D.
2. Press the RESET switch on Pyboard. So doing will loose your USB terminal connection and may even corrupt files. NEVER press RESET switch while the LEDs on the Pyboard are flashing.
3. Remove the power to Pyboard.



Task 0: Flashing LEDs on the Pyboard (5)

- Pyboard has a RESET switch, a USER switch (both momentary), and FOUR LEDs (RED, GREEN, BLUE and YELLOW).
- In response to Pyboard's REPL >>>, enter:

```
from pyb import LED

RED = LED(1)    # onboard red LED
GREEN = LED(2)  # onboard green LED
YELLOW = LED(3) # onboard yellow LED
BLUE = LED(4)   # onboard blue LED

RED.on()
RED.off()
```

Line 1: import the LED object class from the pyb library.
Line 3-6: create 4 LED objects for the four different colour LEDs.
Line 8,9: turns the RED LED on and off.

- Now try turning the other colour LEDs ON and OFF.
- Now try this python code after you see the >>> prompt:
- sw() returns 0 or False it the USER switch is not pressed
- Now press on the USER switch and type sw(). It should return a '1' or True.

```
sw = pyb.Switch()
sw()
```

Test yourself

Write a short python program interactively with the Pyboard so that when USER switch is not pressed, GREEN LED is ON and all others are OFF. When USER switch is pressed, BLUE LED is ON and others OFF.

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 7

Before you start the experiment properly, let us try to write some simple python code interactively with the Pyboard. Enter the code above in response to the REPL prompt >>>.

What we are doing here is just to flash the LED on the Pyboard. Experiment with this so that you know how to control all four colour LEDs.

Next, you should learn to use the user switch on the Pyboard and check whether it has been pressed or not. To do this, you need to create a switch object "sw" with:

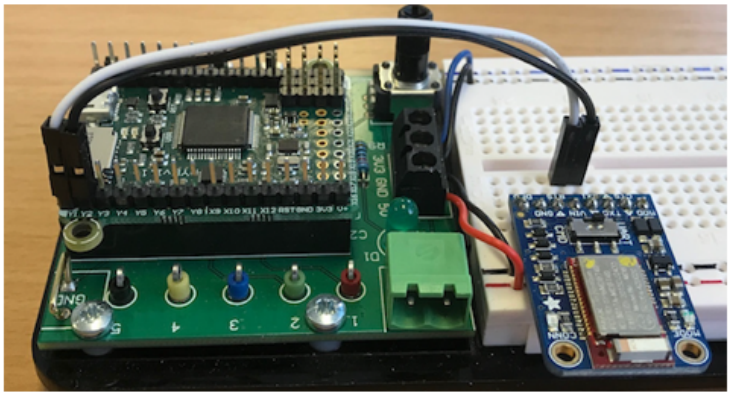
sw = pyb.Switch()

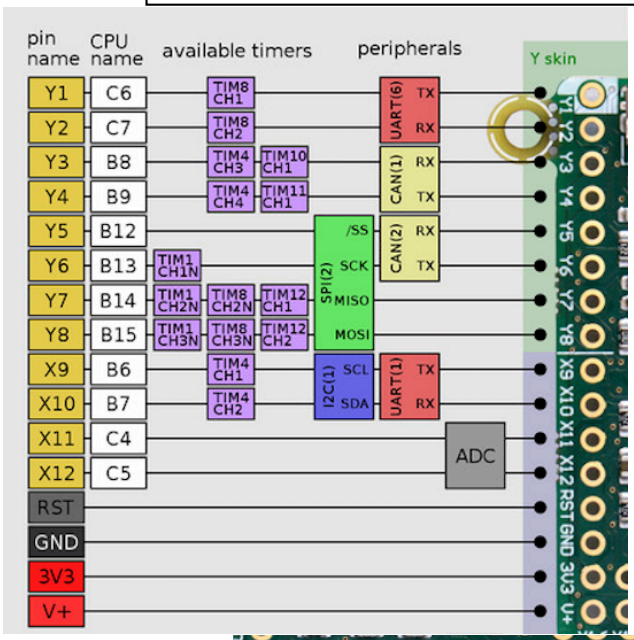
Thereafter, sw() function just returns either 0 (not pressed) or 1 (pressed).

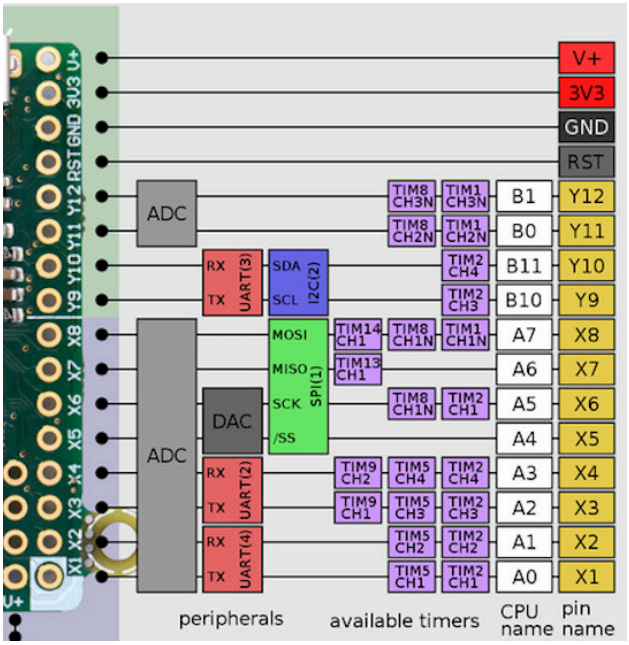
This is a useful feature for later use in the Team Project.

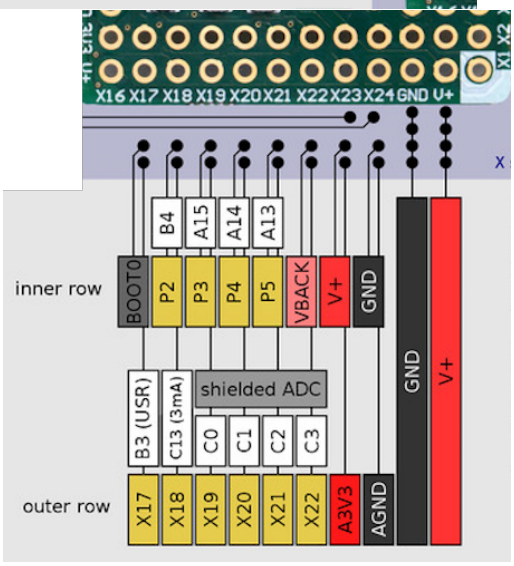
Task 0: Connecting Pyboard to Breadboard

- Pyboard has a many pins sticking out – this is intentional
- These pins can be programmed to be analogue pin or digital pin, input or output, all done using Python code
- The diagram below shows the name of the pins on Pyboard, and they are also marked on the board itself
- You will be connecting some of these pins to your circuits on the breadboard using male-female prototype wires as shown here:









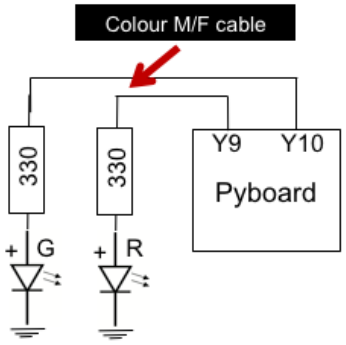
V+: 3.6v - 16v power input
(supplied by USB when USB connected)
3V3: regulated 3.3v output only, max 250mA
VBAT: FET protected supply battery input
VBACK: backup-battery input
A3V3: analog reference connected to 3V3 via inductor

X17 is pulled to GND via 4.7k resistor when USR pressed
P2-P5 are connected to the 4 LEDs
SD_SW = A8 is used for SD card switch
MMA_INT = B2 is used for accelerometer interrupts
MMA_AVDD = A10 is used for accelerometer power

connect BOOT0 to 3V3 and press RST to enter DFU mode

Task 1: Digital output and flashing LEDs (1)

- In this task, you will learn to program two pins on the Pyboard as digital output pins, and use them to drive external RED and a GREEN light emitting diodes (LEDs).
- Wire up the two LEDs to pins Y9 and Y10 on the Pyboard as shown below.
- Note that the **positive terminal** is the **longer leg** on the LED.
- Enter the following Python codes directly in response to Pyboard's REPL >>>.




```
1  from pyb import Pin
2  RED = Pin('Y9', Pin.OUT_PP)
3  GRN = Pin('Y10',Pin.OUT_PP)
```

- Line 1: import from the pyboard library class **Pin** (to control I/O pins)
- Line 2 and 3: create two **pin objects** RED and GRN, and assign them to the corresponding pins on the Pyboard.
- **Pin.OUT_PP** specifies that these are **output** pins with both internal pull up and pull down resistors (**OUT_PP**).
- Now try flash the red LED with code:

```
RED.high()
RED.low()
```

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 9

We will now start to use the Pyboard to do something useful. In this task, you will use the Pyboard (and its microcontroller) to send two digital output signals to drive a red and a yellow LED on the breadboard (not the onboard ones).

The circuit you need to build on the breadboard is shown above. You are driving the LEDs directly from Pin Y9 and Y10, which are names given to the pins on the board. (They have different names on the microcontroller chip. But we will use the board name in this Lab.)

An LED (light emitting diode) is a component which will only conduct current in one direction, as indicated by the symbol. That is, current will flow in the direction of the triangle as if it is an arrow. Therefore, for the LED to work, the terminal with the + sign must be at a higher voltage than the terminal with the horizontal line.

When a diode has a positive voltage V_F at the +ve terminal (with the name "anode") relative to the –ve terminal (named "cathode), and current is following through the diode, we say that the diode is **forward biased**. If you reverse the voltage (so that –ve terminal is at a higher potential than the +ver terminal), we say that the diode is **reverse biased**.

V_F needs to be above a minimum threshold before the diode is conducting current. This threshold is known as the forward or threshold voltage. For these LEDs, they are around 2V. Since the digital output Y9 and Y10 are around 3.3V when it is high, the LEDs will have 4mA current flowing through when they are ON.

We will heavily rely on libraries that come with the Pyboard. The statement "**from pyb import Pin**" states that we will be using functions in the **Pin Class** library in **pyb**.

Task 1: Digital output and flashing LEDs (2)

- Now create a file named **task1.py** with your favourite editor program on your computer. (I use ***Textwangler*** or ***Atom*** program on my Mac.)
- Add the following python code to task1.py.
- Edit the main.py file to include the line:

  execfile('task1.py')
- When you **reboot** the Pyboard by pressing the **RESET** button, it will be running the task1.py program instead.
- We import the **Timer** library in order to access **pyb.delay()** function. This function will wait for the specified number of milliseconds before moving on.
- The **while loop** will loop forever. You can breakout from the infinite loop by typing CTRL-C on the keyboard.

```
#_____________________________________
# Task 1: Blink LEDs in 1 second interval

import pyb
from pyb import Pin, Timer

print('Task 1: Flashing LEDs at 1 second period')

RED = Pin('Y9', Pin.OUT_PP) # rLED is driven by Pin Y9
GREEN = Pin('Y10',Pin.OUT_PP)   # yLED is driven by Pin Y10

while True:
    RED.high()
    pyb.delay(250)      #  delay by 250 msec
    GREEN.high()
    pyb.delay(250)
    RED.low()
    pyb.delay(250)
    GREEN.low()
    pyb.delay(250)
```

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 10

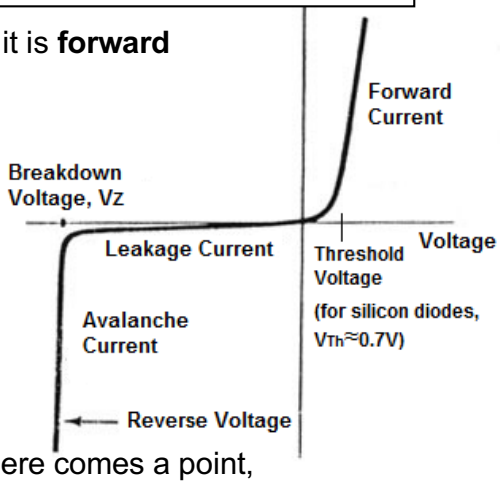
LED is a type of diode component, and it emits light when it is **forward biased**, meaning its voltage is above a threshold.

A diode has a current vs voltage characteristic like this:

For silicon diodes, the threshold voltage is around 0.7V.

For LEDs, it varies from 1.5V up to a few volts.

If the voltage across the diode is below the threshold, or if the voltage is reversed, the diode will not light up.



If you keep increasing the reverse voltage on the diode, there comes a point, at V_Z, when suddenly the diode conducts again in the reverse direction. This is known as the **breakdown voltage**.

For normal diode, it is not good to get to this stage. However, there is a class of diodes designed to operate at this breakdown region. They are known as **Zener diodes**.

In the program shown above, we loop forever using the "while" statement. You can always break out from an infinite loop running on the Pyboard by typing CTRL-C.

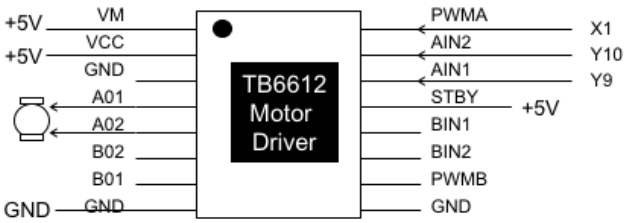
We use two library classes here: **Pin** and **Timer**. To find out more about these library classes, visit the links:

https://micropython.org/doc/module/pyb/Pin

https://micropython.org/doc/module/pyb/Timer

Task 2: Driving a dc motor

- Build the circuit shown here using the TB6612 motor driver chip to interface between the I/O pins of the Pyboard and the DC motor.
- Leave the LED circuit as it was, so you can see whether Y9, Y10 are high or low.
- X9 and X10 are used to control the direction of the motor.
- X1 provides a pulse-width modulation (PWM) signal to control the speed of the motor.
- The TB6612 chip can drive TWO motors, but we are only using use channel A here. You will need to use both channels for the Team Project.
- We need this driver chip because the motor could draw over 1A current, and the microprocessor output CANNOT drive such a load.
- The motor is connected to outputs A01 and A02.
- The datasheet for TB6612 can be found on the course webpage.



PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 11

In order to drive a motor, which takes lots of current, we need to provide an interface circuit between the microcontroller chip (and its pins) and the motor. For this, we use a motor driver chip, TB6612, which is a small board with a chip in the middle.

Connect the power and ground wires on the breadboard, and then connect Y9 and Y10, with the LEDs stay connected as previously, to the Pyboard. In this way, you can see the state of Y9 and Y10 with the LEDs.

PWMA pin is connected to pin X1. X1 will be programmed to produce a pulse-width modulation (PWM) signal. This is a signal that produce a positive-going pulse periodically (say at 1kHz). The width of the pulse is variable. In this way, the average voltage of the signal is directly proportional to the duty cycle of the PWM signal:

average_voltage = VM x duty cycle = VM x pulse width/ period

The motor is connect to the two output pins AO1 and AO2.

Task 2: Driving a dc motor

- The function of the various signals on the driver chip is shown in this table.
- The following python program controls the chip to turn motor in either directions.

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

- In order NOT have to keep typing this, create file task2.py, change boot.py and run it as a file.

```
from pyb import Pin, Timer
A1 = Pin('Y9',Pin.OUT_PP)
A2 = Pin('Y10',Pin.OUT_PP)
A1.high()
A2.low()
motor = Pin('X1')
tim = Timer(2, freq = 1000)
ch = tim.channel(1, Timer.PWM, pin = motor)
ch.pulse_width_percent(50)
```

- We use timer 2 to produce a 1000Hz clock for the PWM signal.
- We then set up the PWM function as shown here.
- Thereafter, you can change the pulse width as duty cycle in percent to control the speed of motor.

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 12

The way the TB6612 chip works is shown in the table here. The shaded entry is what is being set up with the sample Python program.

If you drive AIN1 high, and AIN2 low, then the PWM signal from X1 to PWM pin will determine the speed of the motor. This is achieved by having the outputs AO1 high and AO1 low for part of the period of the PWM signal.

The Python code shown here is quite simple and obvious. There are three lines that provide the PWM signals:

```
tim = Timer(2, freq = 1000)     # This produces a 1000Hz clock
ch = tim.channel(1, Timer.PWM, pin = motor)
```

This code programs the timer channel to provide a PWM signal.

Finally,

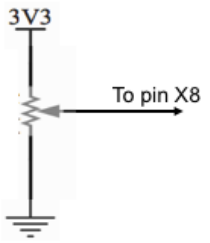
```
ch.pulse_width_percent(50)
```

This generates a PWM signal with 50% duty cycle on the pin object “motor” which has been assigned to X1 previously.

You can now use the **ch.pulse_width_percent( )** function to change the speed of the motor, 0% is stop, and 100% is full speed. Try this yourself.

Task 3: Analogue-to-digital conversion

- The last two tasks are about OUTPUTTING something from the Pyboard. This task is about analogue and digital INPUTS to the Pyboard.
- On the BB, there is a 5k potentiometer (with a round knob) connected to the 3.3V supply and ground as shown here.
- Turning the knob moves the "wiper" of the potentiometer resistor, which provides a voltage from 0 to 3.3V.
- This voltage is measured on pin X8 of the Pyboard as an analogue voltage via an Analog-to-Digital Converter (ADC) inside the microcontroller.
- Create a file task3.py containing the following python code and test the function of the ADC.
- You will find that the ADC provides a reading in the range of 0 to 4095. Why?




```
from pyb import Pin, Timer, ADC
pot = ADC(Pin('X8'))
while True:
    print('Potentiometer voltage: ',pot.read())
    pyb.delay(1000)
```

Read the potentiometer voltage

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 13

So far we have only been OUTPUTTING signals from the Pyboard – performing DRIVE function.

Here we are performing SENSE function. This task is about sensing the position of the potentiometer (with a knob).

The potentiometer is connect to 3.3V at one end, and GND at the other end. As you turn the black knob, the voltage at the wiper output of the potentiometer varies between 0V and 3.3V.

This signal is fed to X8 pin of the Pyboard (already connected). An internal ADC device convert the analogue signal to a digital signal between 0 and 4095 (why?).

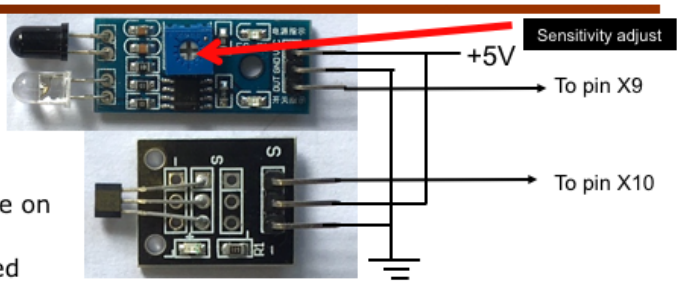
Task 4: Control motor speed with the potentiometer

- This task is a challenge for you. You have learned how to use PWM to control the speed of the motor (Task 2) and how to sense the position of the potentiometer by converting its voltage to digital values (Task 3). You are now required to combine the two, so that you **use the potentiometer to directly control the speed of the motor.**
- You should create a Python program called task4.py, and modify the boot.py file (so that it runs task4.py when you enter CTRL-D while in the "Terminal" for Mac or "putty" for PC).
- As always, you can find the solution to this task under the folder "solutions" if you wish to learn from the solution.

This task is for you to complete on your own. It is intended to let you test yourself, so that you are sure that you have understood tasks 1 to 3. If you take too long on this, consult the solution in the "solution" folder.

Task 5: Infrared and Magnetic Sensors (1)

- Connect the infrared sensor and the Hall effect sensors on the breadboard to the Pyboard as shown here.



- Now try to the following python program to read the digital value on pins X9 and X10.
- Test the sensitivity of the infrared sensor. You can adjust its sensitivity to obstacle distance by adjusting the potentiometer on the board.
- The magnetic sensor is known as a Hall Effect sensor. Explore how it works with the N and S poles of the small magnet provided. (Remember to return the magnet to the bin at the end of your Lab session. It is small and is easily lost.)
- Note that both sensors are "low-active" – output is high when the is no obstacle or magnetic field.

```
import pyb
from pyb import Pin, Timer, LED
print('Task 5: Sensing obstacle and magnetic field')

IR_sensor = Pin('X9',Pin.IN)
hall_sensor = Pin('X10',Pin.IN)
RED = LED(1)
BLUE = LED(4)

while True:
    if IR_sensor.value():  # no signal
        RED.off()
    else:
        RED.on()
    if hall_sensor.value():
        BLUE.off()
    else:
        BLUE.on()
```

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 15

This another SENSE task. We have two sensors: one to sense obstacle via emitting an infrared signal using one infrared diode, and detect the presence or absence of an echo signal back using another infrared diode. It then compares the echo signal with an adjustable threshold (using the blue potentiometer and a screwdriver). It then sends a digital signal to pin X9 on the Pyboard.

The second sensor measures magnetic field using a detector called Hall Effect sensor. We will consider this later in the lecture. For now, you can test this with the magnet provided.

Note that both sensors have an output that is normally a logical '1', and they go low, or logical '0' when activated.

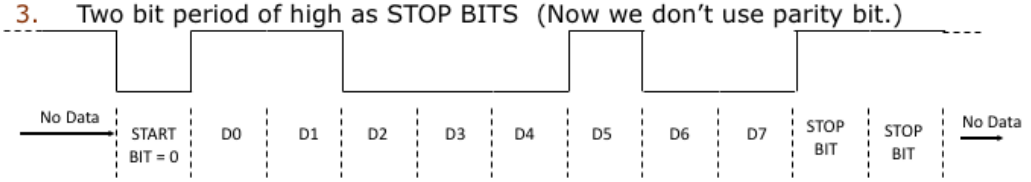
Make sure that you understand the program here.

Task 6: UART signal format – '#'

- Enter the following code directly to Pyboard and connect the scope probe to pin Y1.

```
import pyb
from pyb import UART
uart = UART(6)
while True:
    uart.init(9600, bits=8, parity = 0, stop = 2)
    uart.writechar(ord('#'))
    pyb.delay(5)
```

- Observe on the scope the ASCII character '#' being sent to pin Y1 via the UART (universal asynchronous receiver transmitter) 6's transmit output.
- You have already done this in Lab 1. Data rate (also called baudrate) is 9600 bits/second.
- The UART data format is:
  1. One bit period of low as START BIT
  2. 8 bit periods for data, least significant bit first, containing the ASCII code for '#', which is hexadecimal 23
  3. Two bit period of high as STOP BITS (Now we don't use parity bit.)



PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 16

The intention for this task is for you to understand how a single digital signal could be used to convey text information in the form of an ASCII code. Here is the ASCII table again for you reference. Make sure you understand the digital waveform before moving on.

Note that in Python, ord( ) is a built-in function known as "ordinal".

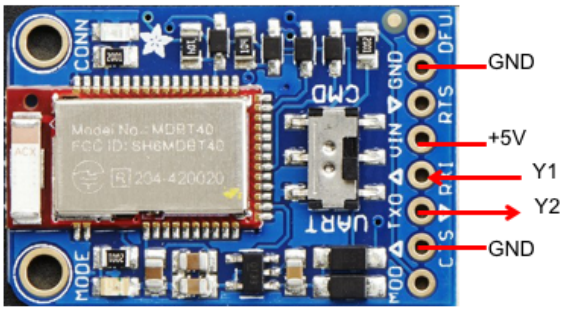
ord('#') returns the ASCII numerical value of the character '#', which is 35.

So ord('A') returns a value of 65.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Task 7: Sending text messages to the phone via Bluetooth

- Connect the Bluetooth module (known as Bluefruit by Adafruit) as shown here. If the module is working, you will see a flashing red LED.
- Install the free mobile app "Adafruit Bluefruit LE Connect" on your Android or Apple phone.
- Run the app, and pair your phone with your Bluefruit module (should be obvious).
- Check that your module has the latest firmware version (ask if you don't know how)
- Select "uart" option in the ICON at the bottom. You are in a mode that shows all ASCII characters received, and you can also send your own text message from your phone to your remote device via Bluetooth.
- Run task 6, and you should receive a continuous stream of '#' characters.
- Modify your program so that you send continuously the stream:
  ABCDE ... XYZ0123456789 <CR> <LF>
- <CR> is carriage return character with an ASCII code of decimal 13.
- <LF> is line feed character with an ASCII code of decimal 10.



PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 17

Now you should understand the idea of using serial data format to send and receive ASCII text characters. The serial interface is called UART. There are a number of hardware UART interfaces in the microcontroller chip.

What we want to do now is to connect a UART on the Pyboard to a Bluetooth module (known as "Bluefruit UART friend".

Bluetooth is a wireless standard used for short distance connection between two electronic widgets. For example, your notebook computer could be connected to a wireless mouse through Bluetooth. The distance is usually limited to a few metres.

In this task, we want to use the Pyboard's UART interface hardware to connect to the Bluefruit module (shown above) to link with your iPhone or Android phone. (Sorry – won't work on a Window phone.)

The Tx and Rx pin on the module are obvious. The two signals RTS (ready to send) and CTS (clear to send) signals are used for checking whether either end is ready to send or receive another character. These are known as HANDSHAKE or FLOW-CONTROL SIGNALS. Both signals are low active (i.e. low voltage means True or logical '1').

We connect CTS to ground because Pyboard is so fast that it is already ready to receive another character from the Bluefruit board.

You need to download the mobile app for Android or iPhone in order to receive text messages from the Pyboard and your program.

You may also need to upgrade the firmware on the Bluefruit UAR Friend module. (Run the App on your phone, pair with the board and you can check on the setting button.)

Task 8: Remote keypad messages from phone to Pyboard

- In task 7, you were receiving messages on the phone via Bluetooth. In this task, you will be doing the opposite: sending messages from the phone via Bluetooth.
- Run the mobile APP and select CONTROLLER and Control Pad. You will see the following control pad displayed on the phone.
- Whenever you press a button, you send the ASCII code: !B*xyz*

  x = '1', '2', '3' or '4' for numeric keys, and '5' to '8' for the cursor keys

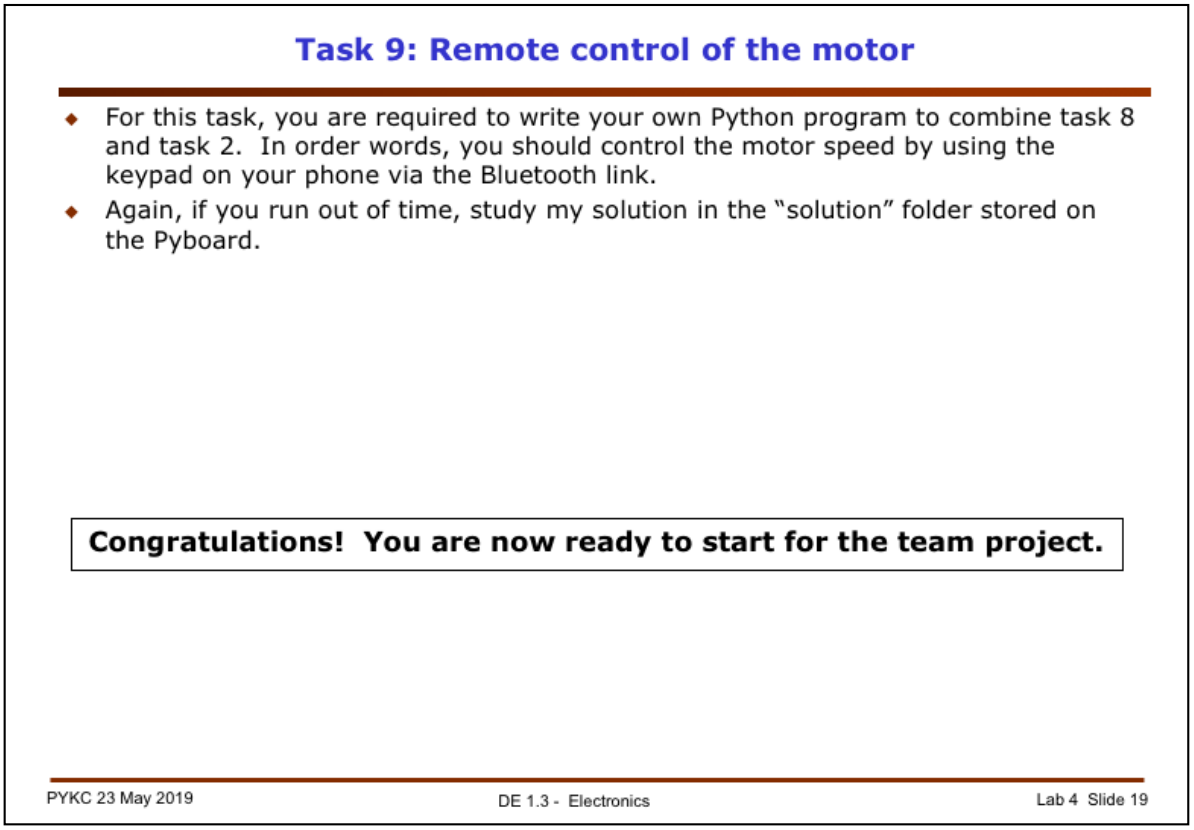
  y = '0' for pressing down, '1' for releasing key

  z = a check character known as CRC (cyclic redundancy check)

- Create task8.py which contains the program shown here.
- Change boot.py and run task8.py.
- Now you should detect a key being pressed and then released with the appropriate message on your computer (via the Pyboard and the Bluetooth module).

```
import pyb
from pyb import Pin, Timer, UART
print('Task 8: Test keypad communication with Pyboard')

key = ('1','2','3','4','U','D','L','R')
uart = UART(6)
uart.init(9600, bits=8, parity = None, stop = 2)
while True:
    while (uart.any()<10):    #wait for 10 chars
        pass
    command = uart.read(10)
    key_index = command[2]-ord('1')
    if (0 <= key_index <= 7) :
        key_press = key[key_index]
    if command[3]==ord('1'):
        action = 'pressed'
    elif command[3]==ord('0'):
        action = 'released'
    else:
        action = 'nothing pressed'
    print('Key',key_press,' ',action)
```


This task is most important. It teaches you how to use your mobile device to control the Pyboard remotely, which in turn controls the robotic car. This exercise is very simple. The program currently only detects a key click down then up, i.e. press then release.



This final task is another challenge for you. You are required to combine task 8 and task 2, so that you use the mobile phone keypad with the Adafruit app, via the Bluetooth interface module, communicate (link) with the Pyboard and control the speed of the motor in both forward and reverse directions.

Again if you take too long to do this, the solution is available.

Task 10 (optional): Driving Servo Motors

- This optional task is to introduce you to the use of servo motor (to be covered in a the lecture on "Drive" later)
- You can directly connect the motor to the Pyboard as shown here:

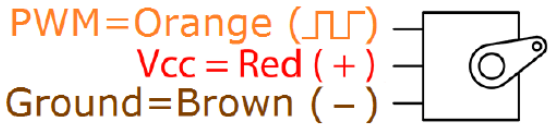
- You may use X1 to X4 to directly control up to FOUR servo motors
- To turn the motor arm, simply use the following Python code:

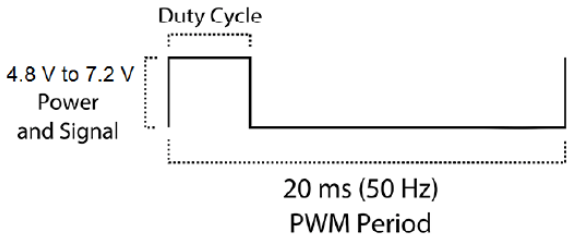
```
import pyb

s1 = pyb.Servo(1)   # create a servo object on position X1
s2 = pyb.Servo(2)   # create a servo object on position X2

s1.angle(45)        # move servo 1 to 45 degrees
s2.angle(0)         # move servo 2 to 0 degrees

# move servo1 and servo2 synchronously, taking 1500ms
s1.angle(-60, 1500)
s2.angle(30, 1500)
```

PYKC 23 May 2019 DE 1.3 - Electronics Lab 4 Slide 20

This is an optional task to show you how to use Pyboard to drive servo motors. We have not done servo motors in lectures yet. The idea is actually very simple – you can instruct a motor's arm to move to any angle you specify within two limits. Typically, the limits could be $\pm 60^{\circ}$.

You can download the servo motor datasheet from the Course webpage. The connection of the servo motor is straight forward – it has only three wires as shown below:





Pyboard is designed to connect directly to a maximum of four servo motors on X1, X2, X3 or X4. Micropython as a Servo Class which makes controlling the angle of the motor trivial, as explained in the slide above.