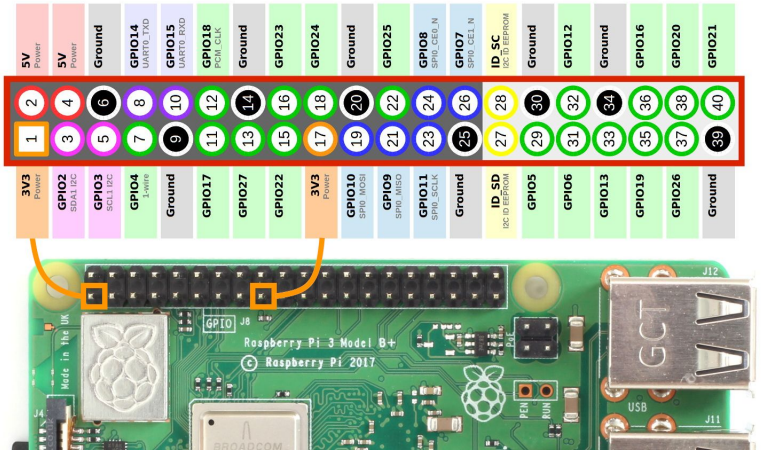
Skill Mapping IoT Workshop

Introduction to raspberry pi & GPIO programming for an IoT application

Nattapong Wattanasiri

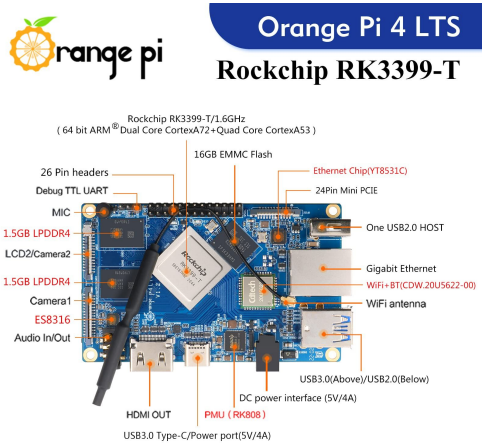


Embedded Linux SCB

- For the workshop, Raspberry Pi will be used.
- **RPi** is **Beginner-Friendly** embedded linux board.

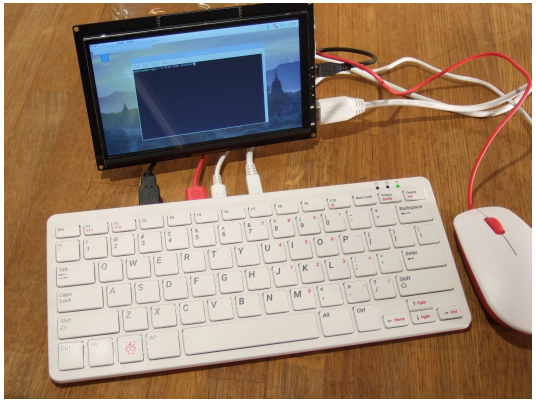
The following table compares the specs of each board:

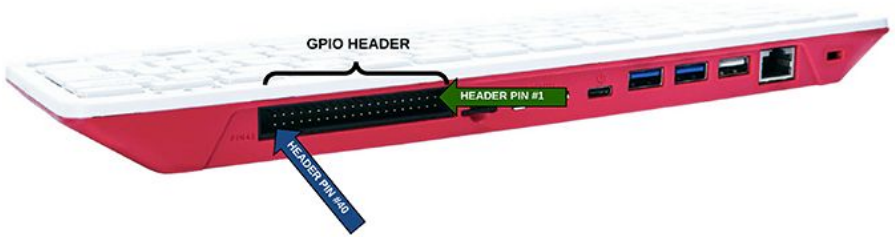
	Arduino Yun	Beaglebone Black	Intel Galileo	Raspberry Pi
Picture				
SoC	Atheros AR9331	Texas Instruments AM3358	Intel Quark X1000	Broadcom BCM2835
CPU	MIPS32 24K and ATmega32U4	ARM Cortex-A8	Intel X1000	ARM1176
Architecture	MIPS and AVR	ARMv7	i586	ARMv6
Speed	400mhz (AR9331) and 16mhz (ATmega)	1ghz	400mhz	700mhz

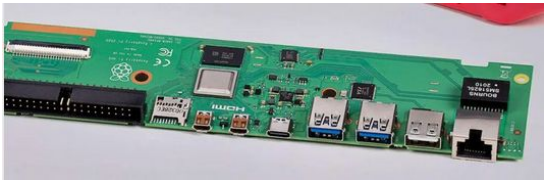


Raspberry Pi 400

- Based on **Raspberry Pi 4**. Featuring the same powerful processor







Raspberry Pi GPIO assignment

For pigpio, Use broadcom (BCM) GPIO assignment system.

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

Raspberry Pi	Pin code	Pin code	Raspberry Pi
+ 3.3V	1	2	+ 5V
(SDA1) GPIO 2	3	4	+ 5V
(SCL1) GPIO 3	5	6	GND
(GPIO_GCLK) GPIO 4	7	8th	GPIO 14 (TXD0)
GND	9	10	GPIO 15 (RXD0)
(GPIO_GEN0) GPIO 17	11	12	GPIO 18 (GPIO_GEN1)
(GPIO_GEN2) GPIO 27	13	14	GND
(GPIO_GEN3) GPIO 22	15	16	GPIO 23 (GPIO_GEN4)
+ 3.3V	17	18	GPIO 24 (GPIO_GEN5)
(SPI_MOSI) GPIO 10	19	20	GND
(SPI_MISO) GPIO 9	21	22	GPIO 25 (GPIO_GEN6)
(SPI_SLCK) GPIO 11	23	24	GPIO 8 (SPI_CE0_N)
GND	25	26	GPIO 7 (SPI_CE1_N)
(only for I2C) ID_SD	27	28	ID_SC (only for I2C)
GPIO 5	29	30	GND
GPIO 6	31	32	GPIO 12
GPIO 13	33	34	GND
GPIO 19	35	36	GPIO 16
GPIO 26	37	38	GPIO 20
GND	39	40	GPIO 21

Interact with the hardware

- **sysfs** interface provided by the Linux kernel. We won't need to do any programming as we can do this from shell commands
- Use **sysfs** to Interface with /sys/class/ files

Interact with the hardware

If your system has a suitable **sysfs** driver loaded, you will see the GPIO hardware exposed in the file system under **/sys/class/gpio**. On a Raspberry Pi it might look something like this:

The basic steps to use a GPIO pin from the **sysfs** interface are the following:

1. Export the pin.
2. Set the pin direction (input or output).
3. If an output pin, set the level to low or high.
4. If an input pin, read the pin's level (low or high).
5. When done, unexport the pin.

Pin assignment for pigpio

It still uses BCM assignment system.

Type 3 - Model A+, B+, Pi Zero, Pi Zero W, Pi2B, Pi3B, Pi4B

- 40 pin expansion header (J8).
- Hardware revision numbers of 16 or greater.
- User GPIO 2-27 (0 and 1 are reserved).

	GPIO	pin	pin	GPIO	
3V3	-	1	2	-	5V
SDA	2	3	4	-	5V
SCL	3	5	6	-	Ground
	4	7	8	14	TXD
Ground	-	9	10	15	RXD
ce1	17	11	12	18	ce0
	27	13	14	-	Ground
	22	15	16	23	
3V3	-	17	18	24	
MOSI	10	19	20	-	Ground
MISO	9	21	22	25	
SCLK	11	23	24	8	CE0
Ground	-	25	26	7	CE1
ID_SD	0	27	28	1	ID_SC
	5	29	30	-	Ground
	6	31	32	12	
	13	33	34	-	Ground
miso	19	35	36	16	ce2
	26	37	38	20	mosi
Ground	-	39	40	21	sclk

Raspberry Pi	Pin code	Pin code	Raspberry Pi
+ 3.3V	1	2	+ 5V
(SDA1) GPIO 2	3	4	+ 5V
(SCL1) GPIO 3	5	6	GND
(GPIO_GCLK) GPIO 4	7	8th	GPIO 14 (TXD0)
GND	9	10	GPIO 15 (RXD0)
(GPIO_GEN0) GPIO 17	11	12	GPIO 18 (GPIO_GEN1)
(GPIO_GEN2) GPIO 27	13	14	GND
(GPIO_GEN3) GPIO 22	15	16	GPIO 23 (GPIO_GEN4)
+ 3.3V	17	18	GPIO 24 (GPIO_GEN5)
(SPI_MOSI) GPIO 10	19	20	GND
(SPI_MISO) GPIO 9	21	22	GPIO 25 (GPIO_GEN6)
(SPI_SLCK) GPIO 11	23	24	GPIO 8 (SPI_CE0_N)
GND	25	26	GPIO 7 (SPI_CE1_N)
(only for I2C) ID_SD	27	28	ID_SC (only for I2C)
GPIO 5	29	30	GND
GPIO 6	31	32	GPIO 12
GPIO 13	33	34	GND
GPIO 19	35	36	GPIO 16
GPIO 26	37	38	GPIO 20
GND	39	40	GPIO 21

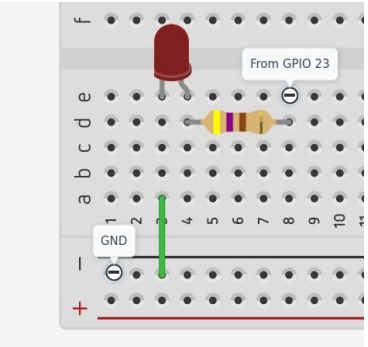
Workshop #1 : Control LED with **sysfs**

Example, Control GPIO Pin 23

```
pi@raspberrypi:/sys/class/gpio $ echo 23 >/sys/class/gpio/export
pi@raspberrypi:/sys/class/gpio $ ls
export  gpio23  gpiochip0  gpiochip504  unexport
pi@raspberrypi:/sys/class/gpio $ ls gpio23/
active_low  device  direction  edge  power  subsystem  uevent  value
pi@raspberrypi:/sys/class/gpio $ echo out >/sys/class/gpio/gpio23/direction
pi@raspberrypi:/sys/class/gpio $ echo 1 >/sys/class/gpio/gpio23/value
pi@raspberrypi:/sys/class/gpio $ echo 0 >/sys/class/gpio/gpio23/value
pi@raspberrypi:/sys/class/gpio $
```

Export target pin, gpio23 folder will appear.

Set direction mode (Input/Output)



Write 1 or 0 to the output (for output pin)

Use pigpiod instead

- **Pigpiod** is **daemon**-service for control GPIO of raspberry pi (Pre-install on Raspbian OS)
- **Pigpio** library is used to interact with **Pigpiod**.
  (https://abyz.me.uk/rpi/pigpio/index.html)


```
pi@raspberrypi:~ $ ps -ef | grep pigpio
pi        1913  1326  0 05:05 pts/0    00:00:00 grep --color=auto pigpio
pi@raspberrypi:~ $ sudo systemctl start pigpiod.service
pi@raspberrypi:~ $ ps -ef | grep pigpio
root      1935     1 25 05:06 ?        00:00:00 /usr/bin/pigpiod -l
pi        1940  1326  0 05:06 pts/0    00:00:00 grep --color=auto pigpio
pi@raspberrypi:~ $
```

pigs m/mode

M/MODES g m - Set GPIO mode

This command sets GPIO g to mode m, typically input (read) or output (write).

Example

```
$ pigs m 4 r # Input (read)
$ pigs m 4 w # Output (write)
$ pigs m 4 0 # ALT 0
$ pigs m 4 5 # ALT 5
```

Mode	Input	Output	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
Code	R	W	0	1	2	3	4	5

MG/MODEG g m - Set GPIO mode

This command returns the current mode of GPIO g.

Example

```
$ pigs mg 4
1
```

Value	0	1	2	3	4	5	6	7
Mode	Input	Output	ALT5	ALT4	ALT0	ALT1	ALT2	ALT3

pigs r/read & pigs w/write

R/READ g - Read GPIO level

This reads the current level of GPIO g

Example

```
$ pigs r 17 # Get level of GPIO 17.
0

$ pigs r 4 # Get level of GPIO 4.
1
```

W/WRITE g L - Write GPIO level

This command sets GPIO g to level L. The level may be 0 (low, off, clear) or 1 (high, on, set).

```
$ pigs w 23 0
$ pigs w 23 1

$ pigs w 23 2
-5
ERROR: level not 0-1
```

Workshop #2 : Control LED with **pigs**

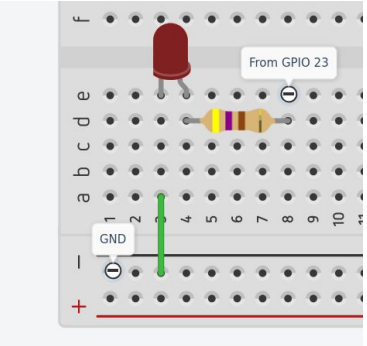
- Use GPIO 23 as LED pin. Connect GPIO 23 to the LED circuit.
- Set GPIO 23 direction as the **output**.

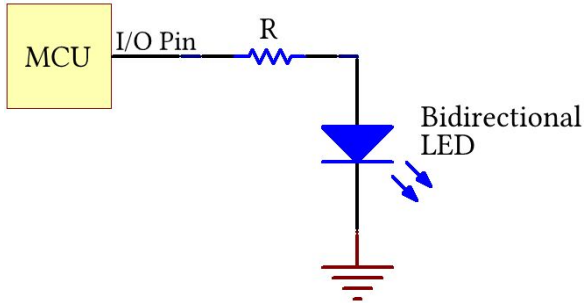
  $ pigs m 23 w # set as output

- Turn on and turn off the LED with **pigs w** command.

  $ pigs w 23 1 # turn on

  $ pigs w 23 0 # turn off





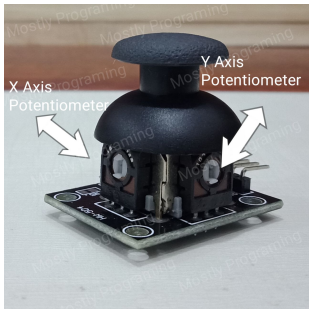
Workshop #3 : Read PS2 Joystick with **pigs**

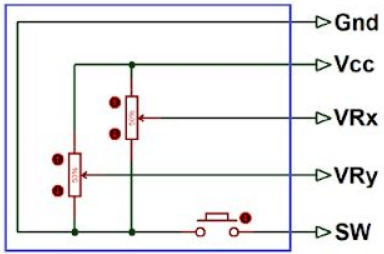
- Use GPIO 24, 25 as X, Y input for the joystick and connect the circuit.
- Set GPIO 24, 25 as Input mode

  $ pigs m 24 r # set as input

- Read the logic value of X and Y with pigs commands. Try to move joystick and observe the changing of value.

  $ pigs r 24 # read input





To GND

To 3.3V

To GPIO 24

To GPIO 25

How to automate GPIO with pigs

- We can create **shell script** (.sh file) or **bash script** and run it.
- If need more complexity of program. Use **C/C++** Programming/**Python** Programming with pigpio lib.

To compile, link, and run a C program

```
gcc -Wall -pthread -o foobar foobar.c -lpigpio -lrt
sudo ./foobar
```

Blink LED with shell script

```
#!/bin/sh

# Setup GPIO for LED as OUTPUT pin
export ledpin=23
pigs modes $ledpin w

# Blinking LED
export period=0.5
export numseq=4
for i in `seq $numseq`
do
        echo "turn on LED @ GPIO pin $ledpin"
        pigs write $ledpin 1
        sleep $period
        echo "turn off LED @ GPIO pin $ledpin"
        pigs write $ledpin 0
        sleep $period
done
```

C Programming on pigpio library

- Include <pigpio.h> in your source files.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4
5 #include <pigpio.h>
6
7 /*
8 # servo_demo.c
9 # 2016-10-08
10 # Public Domain
```

- Assuming your source is in prog.c use the following command to build and run the executable. (Build with GCC)

  gcc -Wall -pthread prog.c -lpigpio -lrt -o prog

  sudo ./prog

C Programming on pigpio library

pigs cmd

BASIC		
M/MODES g m	Set GPIO mode	gpioSetMode
MG/MODEG g	Get GPIO mode	gpioGetMode
PUD g p	Set GPIO pull up/down	gpioSetPullUpDown
R/READ g	Read GPIO level	gpioRead
W/WRITE g L	Write GPIO level	gpioWrite
PWM (overrides servo commands on same GPIO)		
P/PWM u v	Set GPIO PWM value	gpioPWM
PFS u v	Set GPIO PWM frequency	gpioSetPWMfrequency
PRS u v	Set GPIO PWM range	gpioSetPWMrange
GDC u	Get GPIO PWM dutycycle	gpioGetPWMdutycycle
PFG u	Get GPIO PWM frequency	gpioGetPWMfrequency

C functions

Makefile? Why you should use it.

What if your program has a lot of dependencies.

Example

$ gcc afe_worker.c ../smartmeter_lib/cs5484_wiringpi.c ../smartmeter_lib/rtc.c ../smartmeter_lib/relay_led.c \
../smartmeter_lib/ct_model.c -l wiringPi -l hiredis -o $(TARGET)

With Makefile, you can just run
$ make

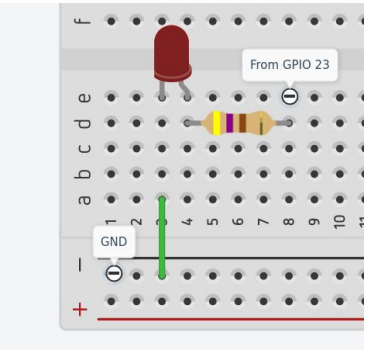
Makefile? Why you should use it.

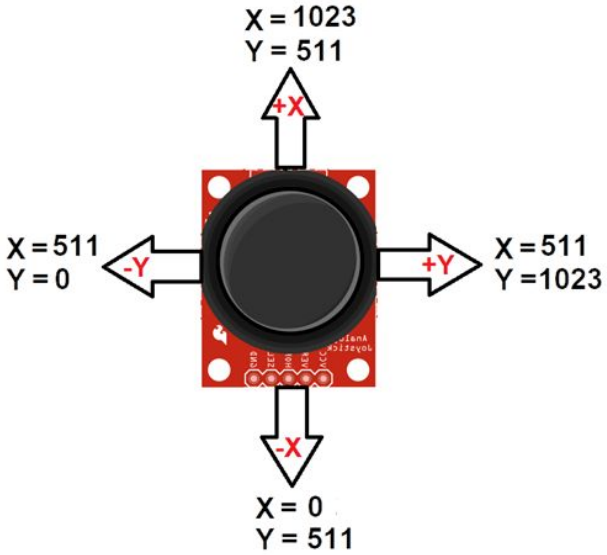
Makefile example

```
#make file - afe worker for smartmeter project

CC=gcc  #compiler
#target file name (output bin)
TARGET=afe_worker
TARGET_ASYNC=afe_worker_async

all:
	$(CC) afe_worker.c ../smartmeter_lib/cs5484_wiringpi.c ../smartmeter_lib/rtc.c ../smartmeter_lib/relay_led.c \
		../smartmeter_lib/ct_model.c -l wiringPi -l hiredis -o $(TARGET)
	$(CC) afe_worker_async.c ../smartmeter_lib/cs5484_wiringpi.c ../smartmeter_lib/rtc.c ../smartmeter_lib/relay_led.c \
		../smartmeter_lib/ct_model.c -l wiringPi -l hiredis -l event -l pthread -o $(TARGET_ASYNC)

debug:
	$(CC) afe_worker.c ../smartmeter_lib/cs5484_wiringpi.c ../smartmeter_lib/rtc.c ../smartmeter_lib/relay_led.c \
		../smartmeter_lib/ct_model.c -g -l wiringPi -l hiredis -o $(TARGET)_debug
	$(CC) afe_worker_async.c ../smartmeter_lib/cs5484_wiringpi.c ../smartmeter_lib/rtc.c ../smartmeter_lib/relay_led.c \
		../smartmeter_lib/ct_model.c -g -l wiringPi -l hiredis -l event -l pthread -o $(TARGET_ASYNC)_debug

clean:
	rm $(TARGET) $(TARGET_ASYNC)
	rm $(TARGET)_debug $(TARGET_ASYNC)_debug
```

Workshop #4 : Blinky LED with C program

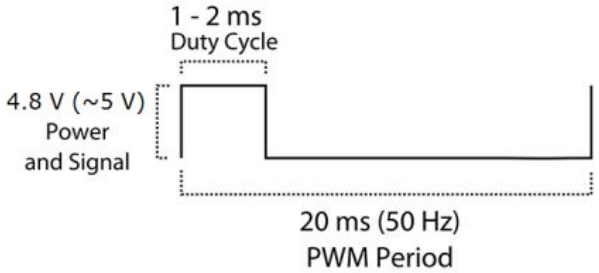
- Using C programming
- LED will blink with period **1 second** when **X-axis** is moved down.
- LED will blink with period **0.2 second** when **Y-axis** is moved left.
- GPIO 24 for X-axis, GPIO 25 for Y-axis
- GPIO 23 for LED
- The example code is available.





Servo motor

- Position-Control Motor
- Controlled with **PWM** input signal (50 Hz signal).



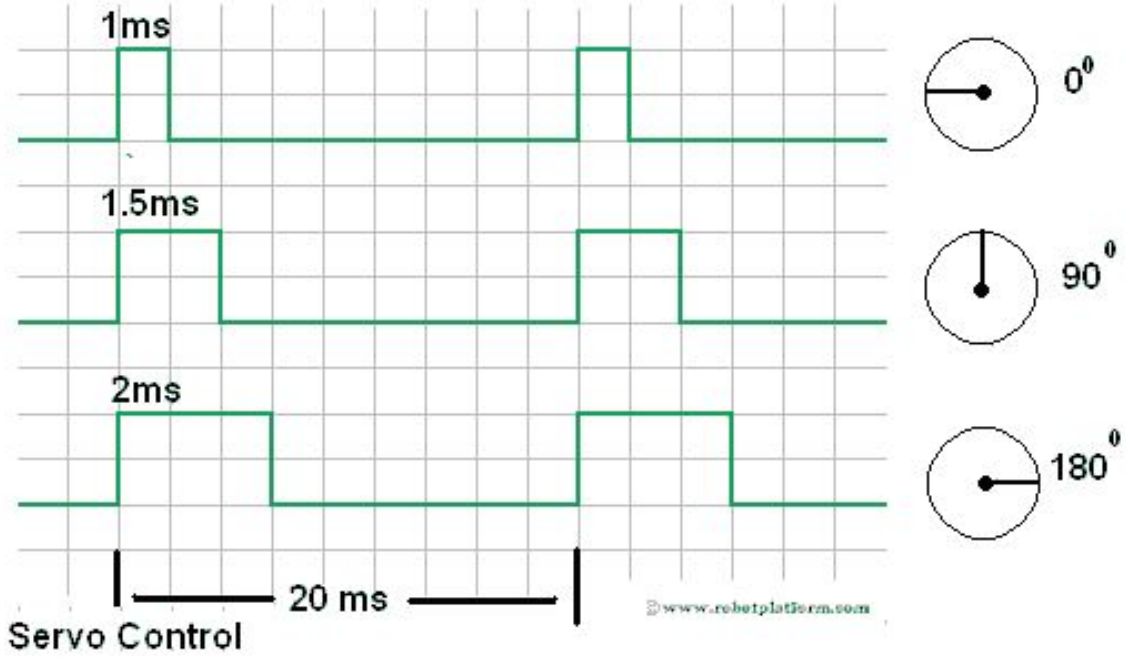
SG90 Servo Motor

Servo Motor Pinout (Wires)

Servo Motor Wire Configuration

Wire Number	Wire Colour	Description
1	Brown	Ground wire connected to the ground of system
2	Red	Powers the motor typically +5V is used
3	Orange	PWM signal is given in through this wire to drive the motor

Servo motor



Workshop #5 : Generate PWM signal for servo with pigpio

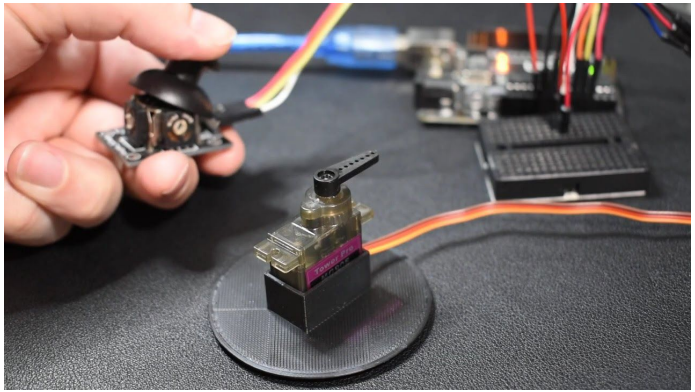
- PWM in raspberry pi has 8-bit solution, (value 0-255 as duty cycle=0-100%)
- Connect orange wire of servo to the pin **GPIO 23** of the raspberry pi. Then run these.

```
$ pigs m 23 w          # set pin as output
$ pigs pfs 23 50       # set PWM frequency = 50 Hz
$ pigs pwm 23 64       # generate PWM signal with duty cycle = 26/256*100% = 10%
$ pigs pwm 23 5        # generate PWM signal with duty cycle = 5/256*100 = 2%
```

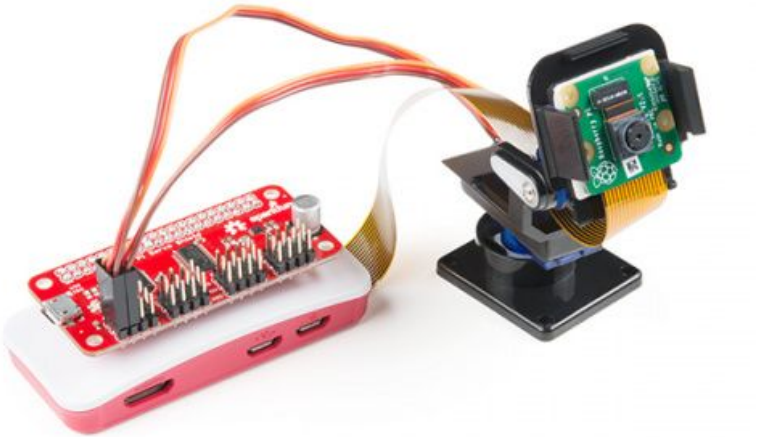
- Experiment with the other duty-cycle value and see the result.

Workshop #6 : Control servo with joystick program

- Using C programming
- Servo will move 180 degree when **Y-axis** is moved left. Else It will stay at 0 degree.
- GPIO 23 for servo motor input (orange wire)
- GPIO 24 for joystick's Y-axis
- The example code is available.



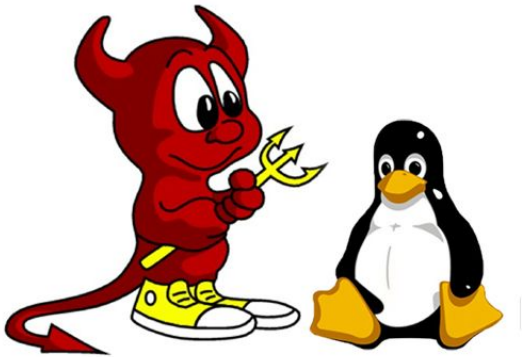
Application for this idea



How to run your program whenever Pi is reboot?

- Your program will not gonna start automatically. You need to set up.
- There are many techniques. But the common one is run your program as a **daemon-service.**
- Run in every startup, in background.

```
1 [Unit]
2 Description=Iot Project Startup Service
3
4 [Service]
5 WorkingDirectory=/usr/local/sbin
6 ExecStart=iot-startup.sh
7 Restart=always
8
9 [Install]
10 WantedBy=multi-user.target
11
```



Q&A