物聯網與微處理機系統設計
Internet of Things and Microprocessor System Design

Lecture 04 – PWM & UART

Lecturer: 陳彥安 Chen, Yan-Ann

YZU CSE

Outline

- PWM
  - LED Control
  - Buzzer
- UART
  - Terminal Program
  - Python Program
- Lab

Outline

- PWM
  - LED Control
  - Buzzer
- UART
  - Terminal Program
  - Python Program
- Lab

Blink

```python
import RPi.GPIO as GPIO
import time

LED_PIN = 12
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_PIN, GPIO.OUT)
try:
    while True:
        print("LED in on.")
        GPIO.output(LED_PIN, GPIO.HIGH)
        time.sleep(1)
        print("LED is off.")
        GPIO.output(LED_PIN, GPIO.LOW)
        time.sleep(1)
except KeyboardInterrupt:
    print("Exception: KeyboardInterrupt")
finally:
    GPIO.cleanup()
```

Load GPIO library

LED is on pin 12 by physical pin numbering (z-shape)

the try clause (the statement(s) between the try and except keywords) is executed.

a user-generated interruption is signaled (ctrl + c)

A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.

Speed-Up Blink (1/2)

Case 1

```
while True:
    # print("LED in on.")
    GPIO.output(LED_PIN, GPIO.HIGH)
    time.sleep(0.005)
    # print("LED is off.")
    GPIO.output(LED_PIN, GPIO.LOW)
    time.sleep(0.005)
```

Case 2

```
while True:
    # print("LED in on.")
    GPIO.output(LED_PIN, GPIO.HIGH)
    time.sleep(0.005)
    # print("LED is off.")
    # GPIO.output(LED_PIN, GPIO.LOW)
    # time.sleep(0.005)
```

Case 3

```
while True:
    # print("LED in on.")
    GPIO.output(LED_PIN, GPIO.HIGH)
    time.sleep(0.001)
    # print("LED is off.")
    GPIO.output(LED_PIN, GPIO.LOW)
    time.sleep(0.009)
```

Case 4

```
while True:
    # print("LED in on.")
    GPIO.output(LED_PIN, GPIO.HIGH)
    time.sleep(0.009)
    # print("LED is off.")
    GPIO.output(LED_PIN, GPIO.LOW)
    time.sleep(0.001)
```

Speed-Up Blink (2/2)

Case 1 50% Brightness

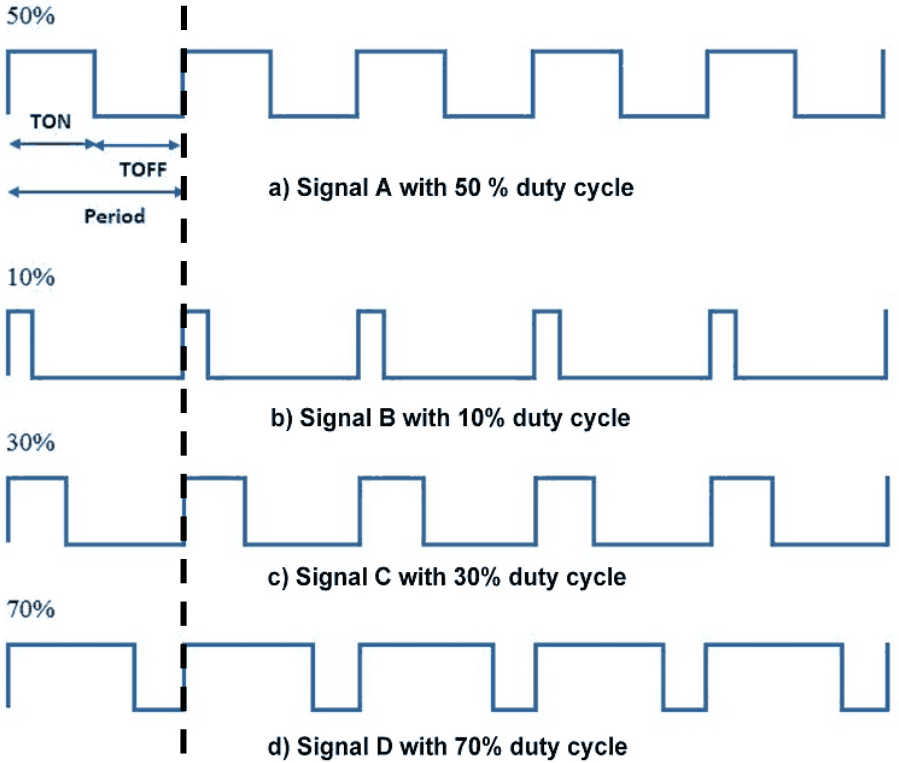
Case 3 10% Brightness

Case 2 100% Brightness

Case 4 90% Brightness

PWM

- Pulse width modulation (PWM) is a modulation technique that generates variable-width pulses to represent the amplitude of an analog input signal.
- For a PWM signal, it consists of frequency and duty cycle.
- Control the delivered average power.



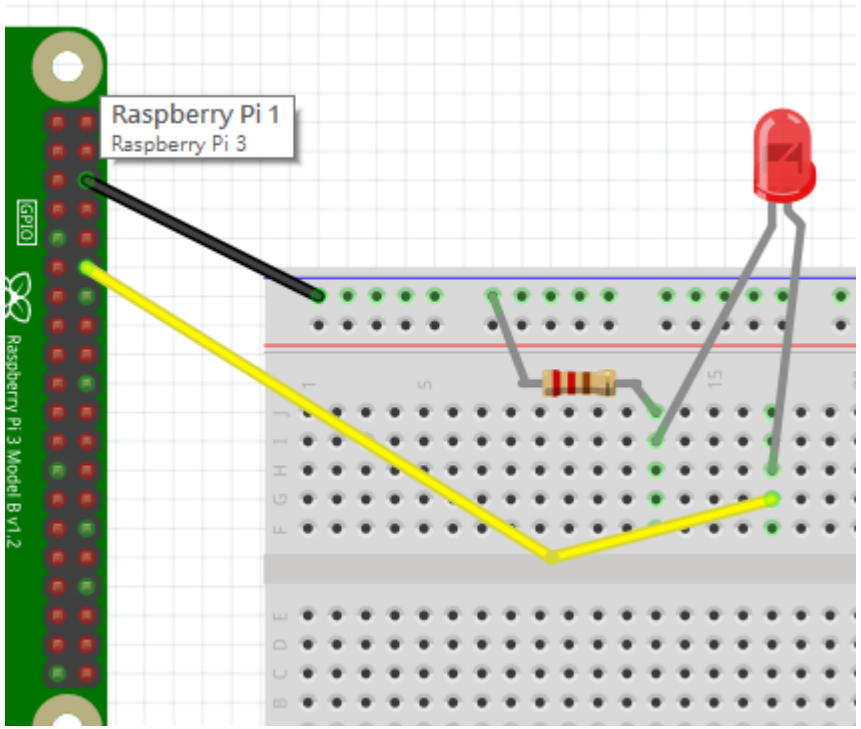
PWM Applications

- Servos motor
  - Control the angle by providing different PWM signals

- Light control
  - Control the delivered power for brightening or dimming an LED.
- Signal modulation
- Frequency generation

Circuit

- PIN 12 - LED(+)
- PIN 6 - LED(-)



RPi.GPIO PWM

To create a PWM instance:

```
p = GPIO.PWM(channel, frequency)
```

To start PWM:

```
p.start(dc)   # where dc is the duty cycle (0.0 <= dc <= 100.0)
```

To change the frequency:

```
p.ChangeFrequency(freq)   # where freq is the new frequency in Hz
```

To change the duty cycle:

```
p.ChangeDutyCycle(dc)  # where 0.0 <= dc <= 100.0
```

To stop PWM:

```
p.stop()
```

pwm_led_control.py

- Control the brightness of LED according to user's input.

```
import RPi.GPIO as GPIO
import time

LED_PIN = 12
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_PIN, GPIO.OUT)

pwm = GPIO.PWM(LED_PIN, 100)
pwm.start(0)
try:
    while True:
        brightness = input("Set brightness (0 ~ 100):")
        if not brightness.isdigit() or int(brightness) > 100 or int(brightness) < 0:
            print("Please enter an integer between 0 ~ 100.")
            continue
        pwm.ChangeDutyCycle(int(brightness))
except KeyboardInterrupt:
    pass
pwm.stop()
GPIO.cleanup()
```

$ python3 pwm_led_control.py

```
Set brightness (0 ~ 100):10
Set brightness (0 ~ 100):20
Set brightness (0 ~ 100):30
Set brightness (0 ~ 100):50
Set brightness (0 ~ 100):100
Set brightness (0 ~ 100):110
Please enter an integer between 0 ~ 100.
Set brightness (0 ~ 100):
```

pwm_breath.py

```
import RPi.GPIO as GPIO
import time

LED_PIN = 12
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_PIN, GPIO.OUT)

pwm = GPIO.PWM(LED_PIN, 100)
pwm.start(0)
try:
    while True:
        for i in range(101):
            pwm.ChangeDutyCycle(i)
            time.sleep(0.05)
        pwm.ChangeDutyCycle(0)
        time.sleep(1)
except KeyboardInterrupt:
    pass
pwm.stop()
GPIO.cleanup()
```

100: frequency

0: duty cycle

i: duty cycle

$ python3 pwm_breath.py

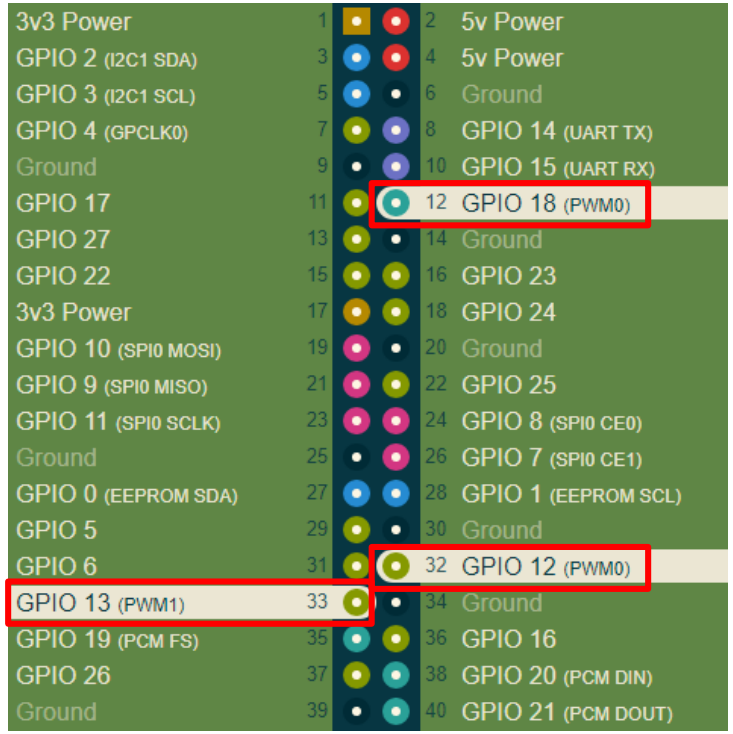
RPi GPIO Speed

Language	Library	Tested / version	Square wave
Shell	/proc/mem access	2015-02-14	2.8 kHz
Shell / gpio utility	WiringPi gpio utility	2015-02-15 / 2.25	40 Hz
Python	RPi.GPIO	2015-02-15 / 0.5.10	70 kHz
Python	wiringpi2 bindings	2015-02-15 / latest github	28 kHz
Ruby	wiringpi bindings	2015-02-15 / latest gem (1.1.0)	21 kHz
C	Native library	2015-02-15 / latest RaspPi wiki code	22 MHz
C	BCM 2835	2015-02-15 / 1.38	5.4 MHz
C	wiringPi	2015-02-15 / 2.25	4.1 – 4.6 MHz
Perl	BCM 2835	2015-02-15 / 1.9	48 kHz

Ref: https://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/

Hardware PWM

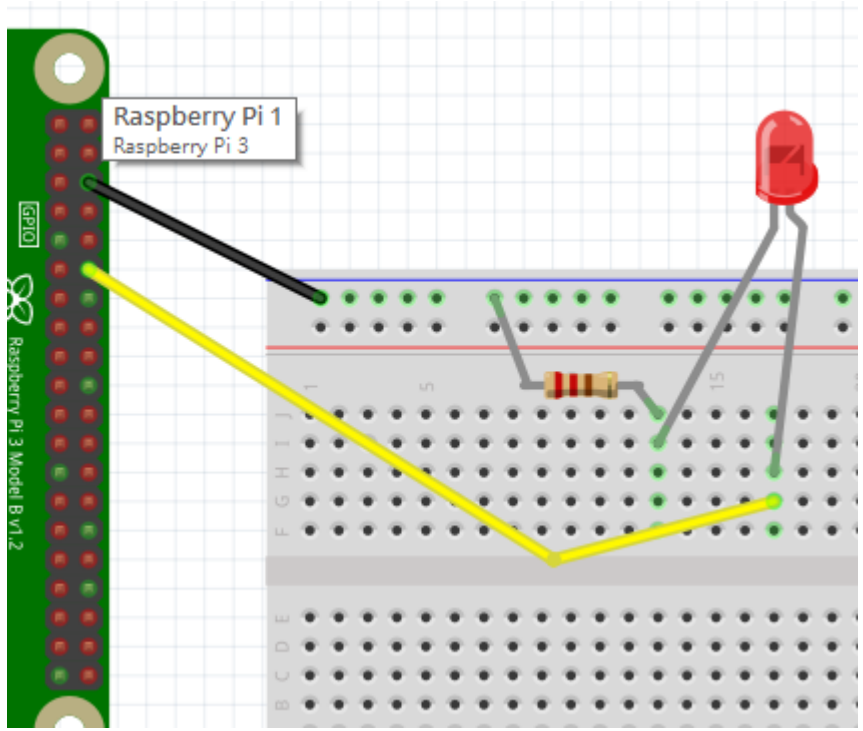
- Generating a PWM signal by HW is more accurate.
  - Software PWM will be affected by OS scheduling.



- RPi.GPIO only supports software PWM.
- pigpio supports hardware PWM.
  - http://abyz.me.uk/rpi/pigpio/python.html

Circuit

- PIN 12 - LED(+)
- PIN 6 - LED(-)



pwm_led_hw.py

```
import pigpio
import time
PWM_PIN = 18 # BCM Number
pi = pigpio.pi()
pi.set_PWM_frequency(PWM_PIN, 100)
pi.set_PWM_range(PWM_PIN, 255)
pi.set_PWM_dutycycle(PWM_PIN, 0)
try:
    while True:
        for i in range(101):
            pi.set_PWM_dutycycle(PWM_PIN, 255*i/100)
            time.sleep(0.05)
except KeyboardInterrupt:
    pass
pi.set_PWM_dutycycle(PWM_PIN, 0)
pi.stop()
```

$ sudo pigpiod
$ python3 pwm_led_hw.py
$ sudo killall pigpiod

CPU Utilization (1/2)

- Set to a fixed brightness by software PWM.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1273	pi	20	0	10400	2980	2468	R	0.7	0.1	0:02.80	top
1315	pi	20	0	22680	7112	4808	S	0.3	0.2	0:00.14	python3
1	root	20	0	32692	8136	6568	S	0.0	0.2	0:04.32	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu+
9	root	20	0	0	0	0	S	0.0	0.0	0:00.09	ksoftirqd+
10	root	20	0	0	0	0	I	0.0	0.0	0:00.33	rcu_sched
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration+
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
14	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration+
15	root	20	0	0	0	0	S	0.0	0.0	0:00.08	ksoftirqd+
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/2
19	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration+
20	root	20	0	0	0	0	S	0.0	0.0	0:00.11	ksoftirqd+

CPU Utilization (2/2)

- Set to a fixed brightness by hardware PWM.

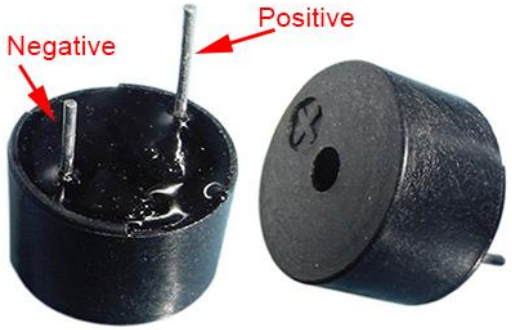
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1330	root	20	0	13568	1768	1584	S	7.3	0.0	0:47.22	pigpiod
1273	pi	20	0	10400	2980	2468	R	1.0	0.1	0:09.54	top
1	root	20	0	32692	8136	6568	S	0.0	0.2	0:04.32	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu+
9	root	20	0	0	0	0	S	0.0	0.0	0:00.10	ksoftirqd+
10	root	20	0	0	0	0	I	0.0	0.0	0:00.47	rcu_sched
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration+
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
14	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration+
15	root	20	0	0	0	0	S	0.0	0.0	0:00.08	ksoftirqd+
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/2
19	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration+
20	root	20	0	0	0	0	S	0.0	0.0	0:00.11	ksoftirqd+

Outline

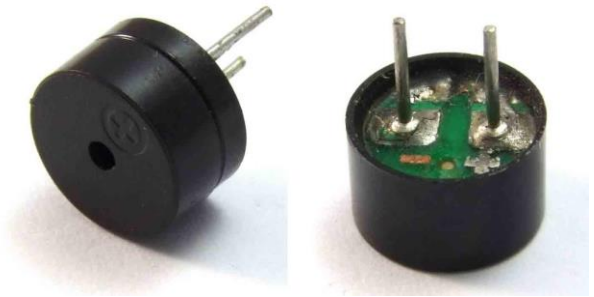
- PWM
  - LED Control
  - Buzzer
- UART
  - Terminal Program
  - Python Program
- Lab

Buzzer

- Active buzzer
  - Generate a tone using internal oscillator.

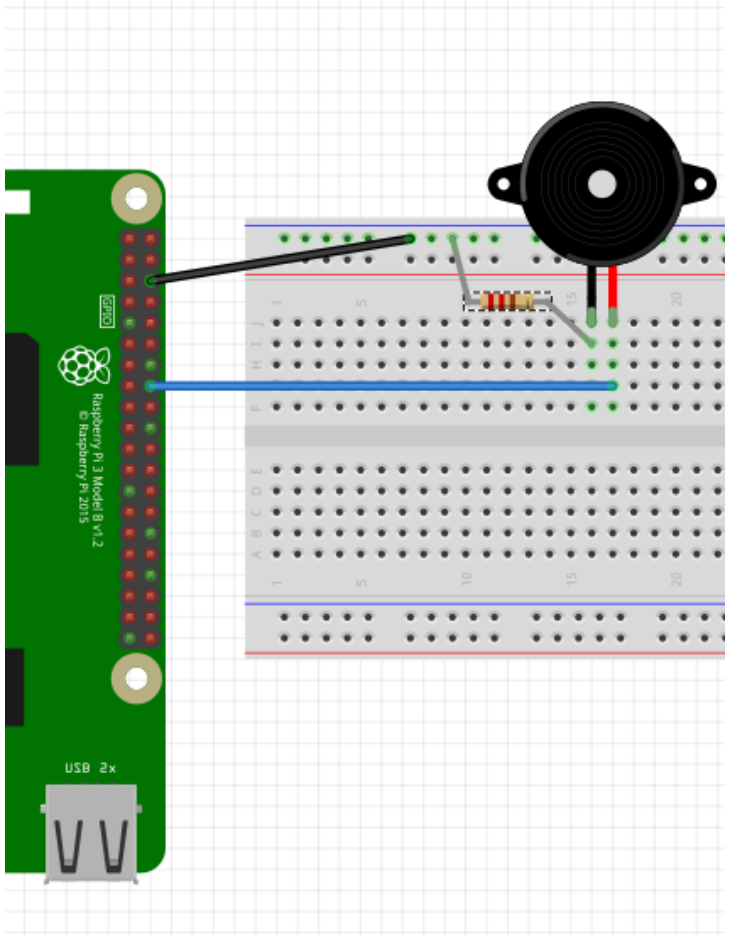


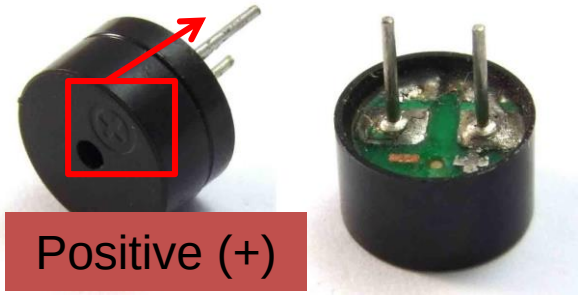
- Passive buzzer
  - Generate the tone by providing an external oscillating electronic signal.
  - We use this passive buzzer here.



Circuit

- PIN 16 - Buzzer(+)
- PIN 6 - 220 Ohm resistor





Pitch

	1	2	3	4	5	6	7	8	9	10	11	12
音名	**C**	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**	**A**	**A#**	**B**
0	16	17	18	19	21	22	23	25	26	28	29	31
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	93	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	493
5	523	554	587	622	659	698	740	784	831	880	932	988
6	1046	1109	1175	1245	1319	1397	1480	1568	1661	1760	1864	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

buzzer.py

```python
import RPi.GPIO as GPIO
import time

BUZZ_PIN = 16
pitches = [262, 294, 330, 349, 392, 440, 493, 523]
# pitches = [262, 294, 330, 349, 392, 440, 493, 523, 587, 659, 698, 784, 880, 932, 988]
GPIO.setmode(GPIO.BOARD)
GPIO.setup(BUZZ_PIN, GPIO.OUT)

pwm = GPIO.PWM(BUZZ_PIN, pitches[0])
pwm.start(0)

def play(pitch, intv):
    pwm.ChangeFrequency(pitch)
    time.sleep(intv)
try:
    while True:
        pwm.ChangeDutyCycle(50)
        for pitch in pitches:
            play(pitch, 1)
        pwm.ChangeDutyCycle(0)
        time.sleep(5)
except KeyboardInterrupt:
    pass
pwm.stop()
GPIO.cleanup()
```

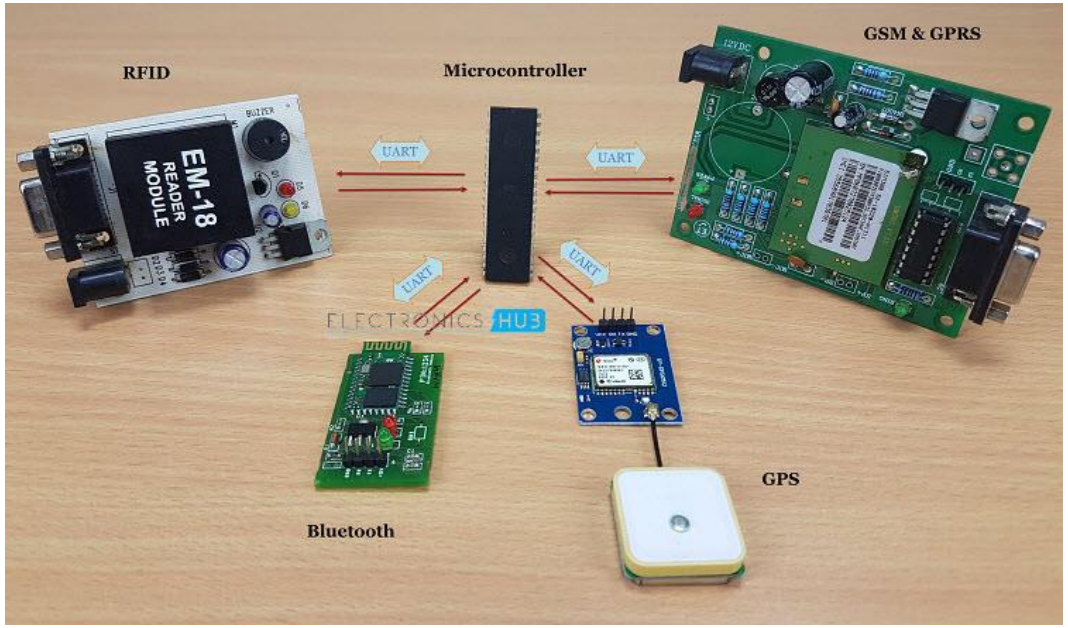
$ python3 buzzer.py

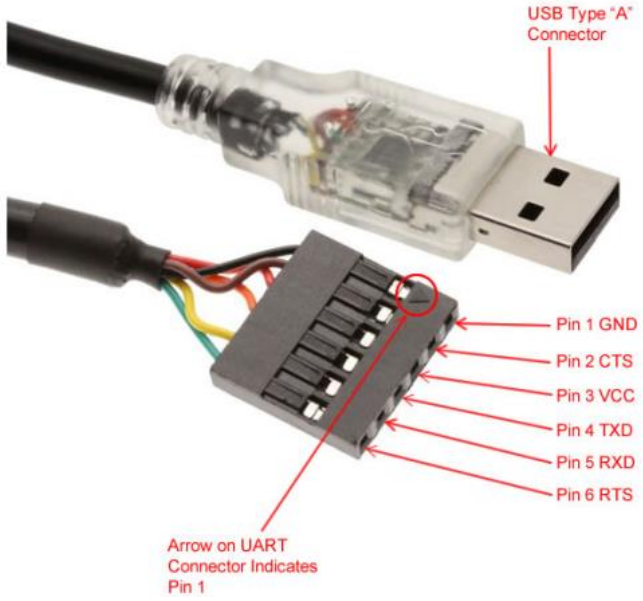
Outline

- PWM
  - LED Control
  - Buzzer
- UART
  - Terminal Program
  - Python Program
- Lab

UART

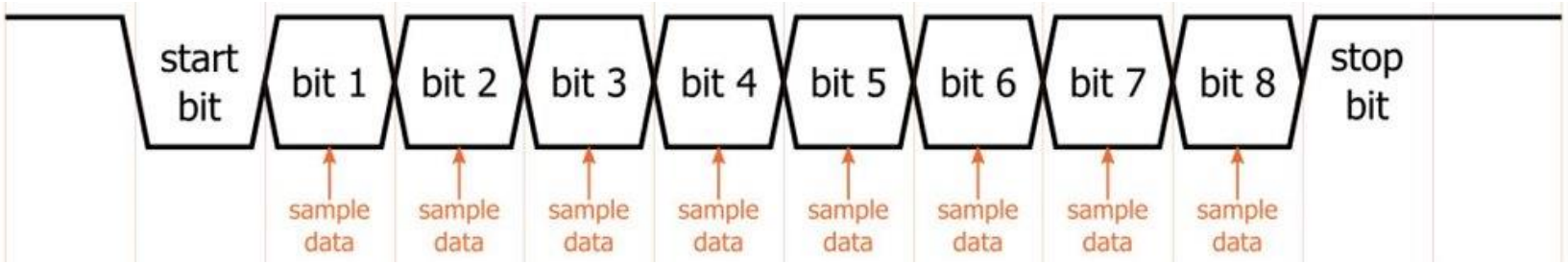
- Universal Asynchronous Receiver/Transmitter
  - Translate data between parallel and serial forms
  - The sender takes bytes of data and transmits the individual bits in a sequential fashion.
  - The receiver reassembles the bits into complete bytes.



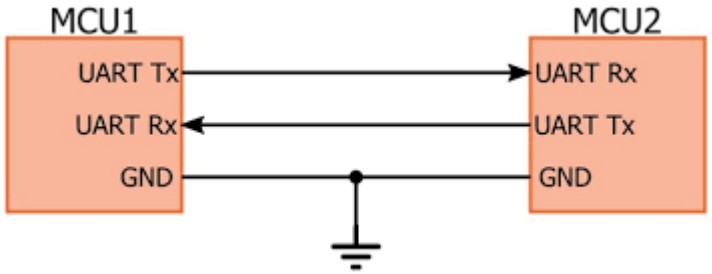


Serial Form

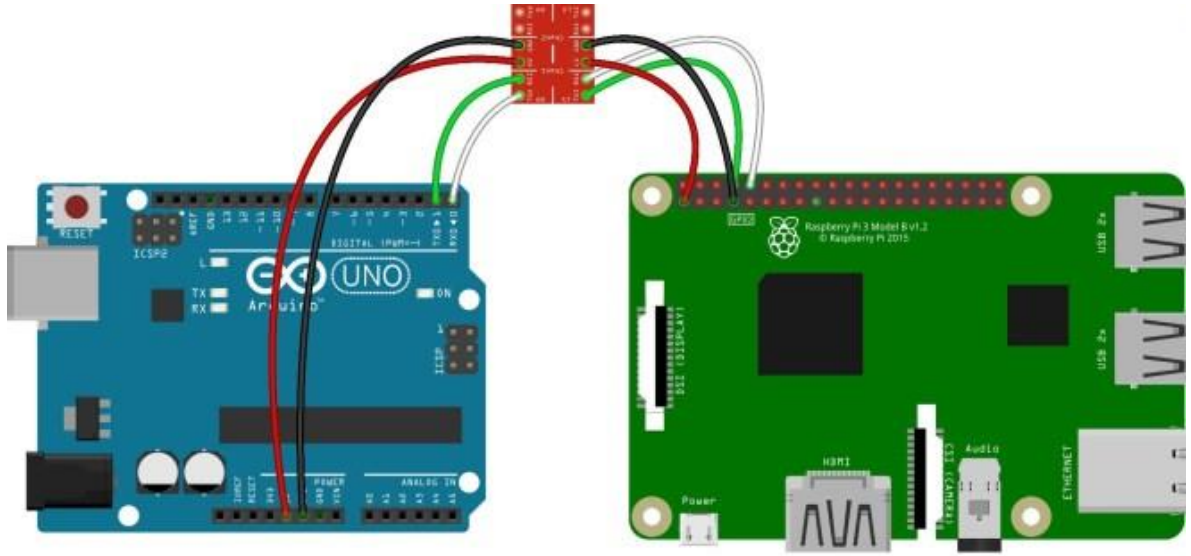

Start
Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
Stop


start bit
bit 1
bit 2
bit 3
bit 4
bit 5
bit 6
bit 7
bit 8
stop bit
sample data
sample data
sample data
sample data
sample data
sample data
sample data
sample data

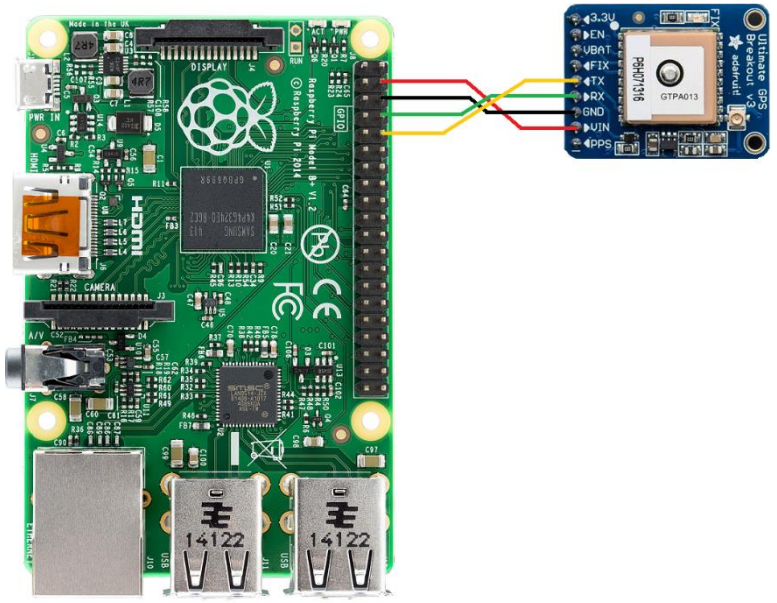
Inter-Device Comm.



ref: https://electropeak.com/learn/raspberry-pi-serial-communication-uart-w-arduino-pc/



ref: https://www.teachmemicro.com/raspberry-pi-serial-uart-tutorial/



Raspberry Pi Photo By Lucasbosch - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=34179985

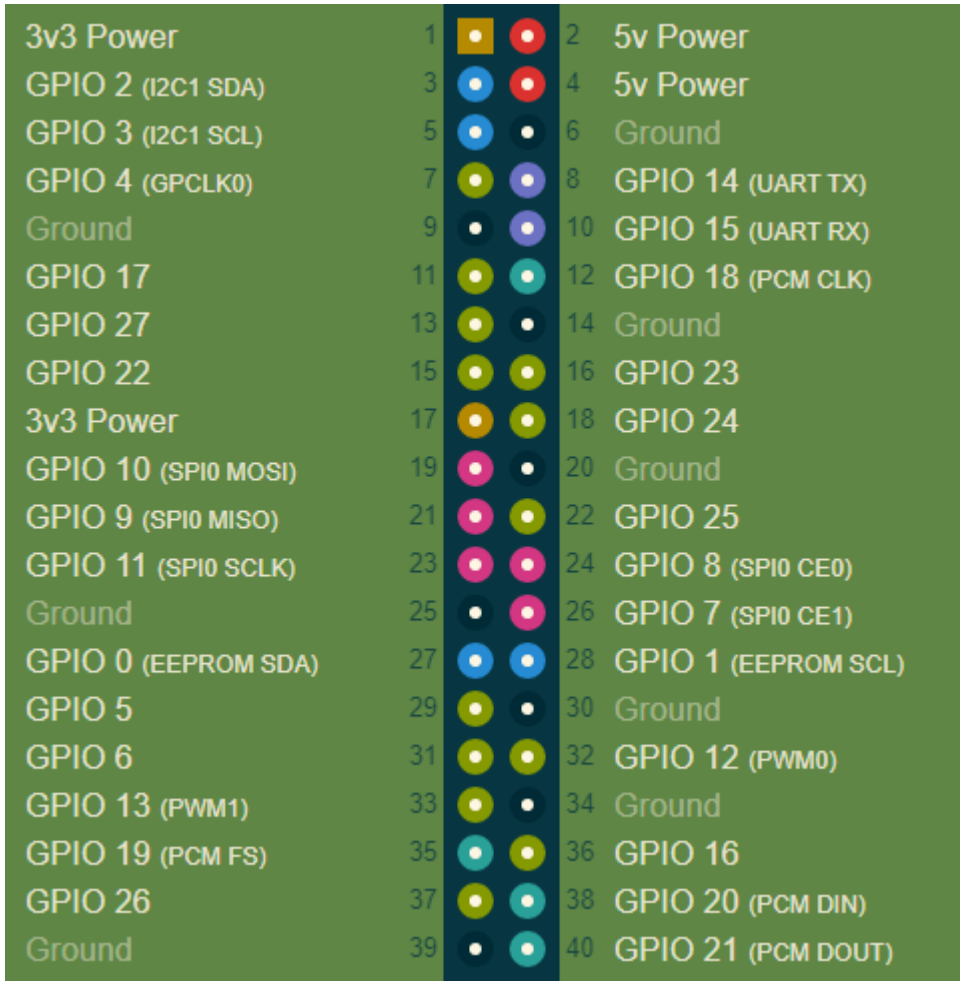
UART on RPi 4

Pi 4 - six UARTS

The Raspberry Pi 4 has four additional PL011s, which are disabled by default. The full list of UARTs on the Pi 4 is:

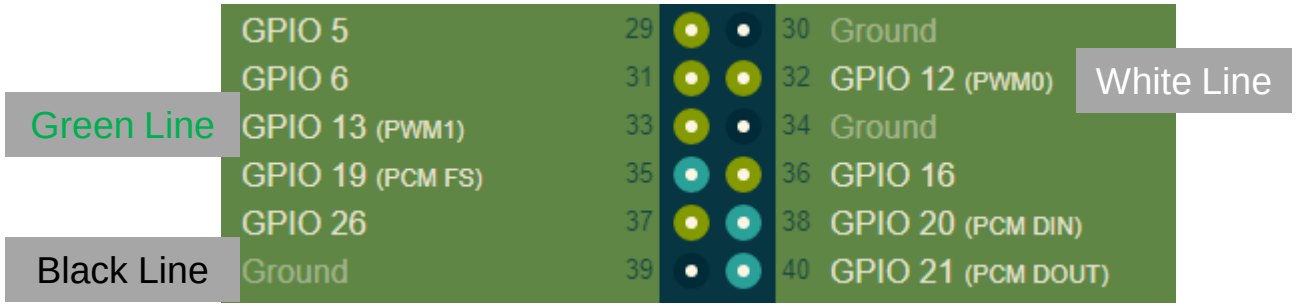
Name	Type
UART0	PL011
UART1	mini UART
UART2	PL011
UART3	PL011
UART4	PL011
UART5	PL011

```
        TXD RXD CTS RTS     Board Pins
uart0   14  15              8   10
uart1   14  15              8   10
uart2   0   1   2   3       27  28  (I2C)
uart3   4   5   6   7       7   29
uart4   8   9   10  11      24  23  (SPI0)
uart5   12  13  14  15      32  33  (gpio-fan)
```



RPi-PC Comm.

- Connect TTL cable to RPi



- Putty@PC <-> minicom@RPi

UART5

- Enable UART5

$ sudo nano /boot/config.txt

- Add the following line into config.txt for enabling UART5.

dtoverlay=uart5

```
[all]
#dtoverlay=vc4-fkms-v3d

dtoverlay=miniuart-bt
core_freq=250
enable_uart=1

dtoverlay=uart5
```

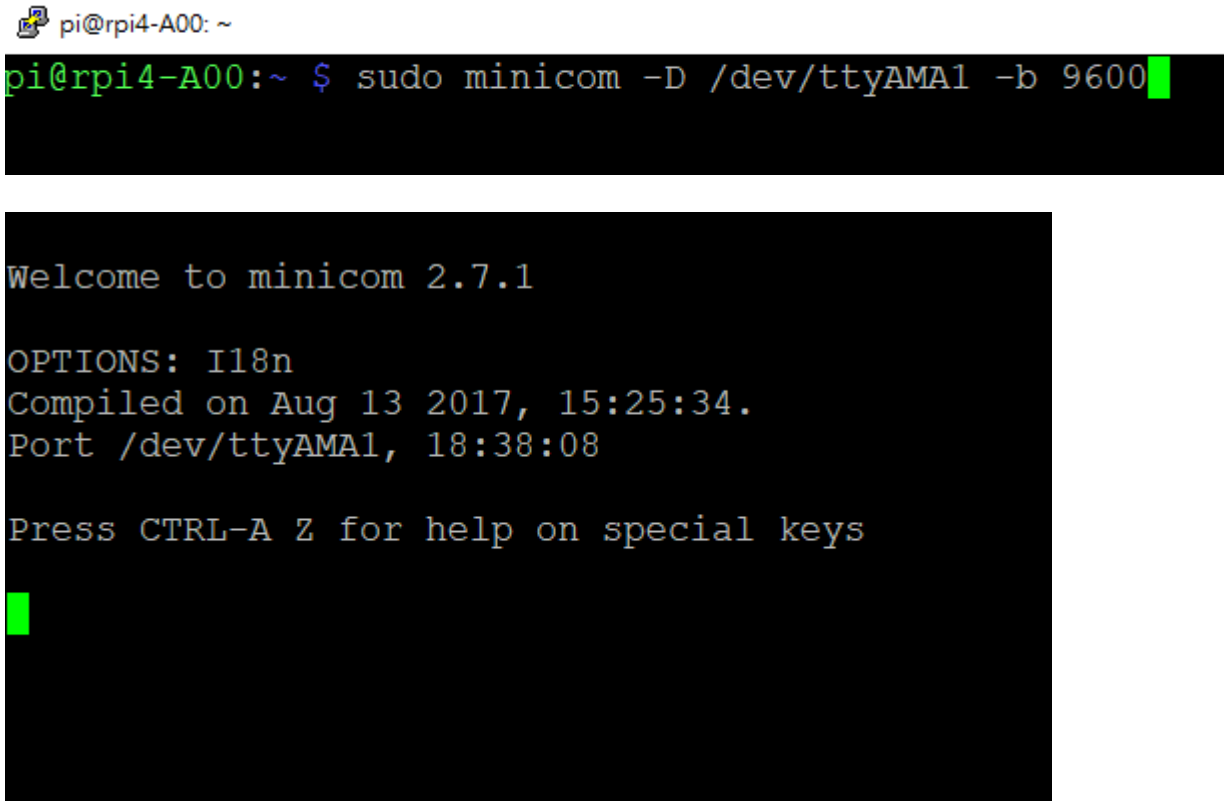
- Restart the system

$ sudo reboot

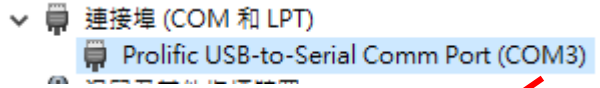
Minicom@RPi

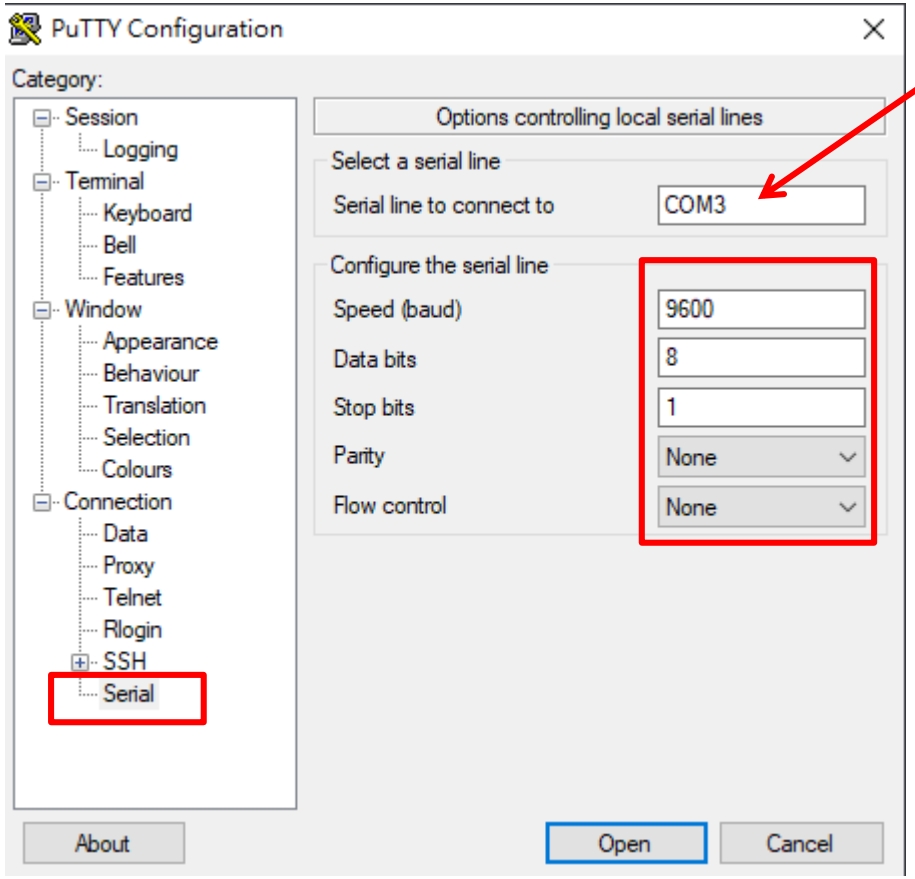
$ sudo apt-get --yes install minicom

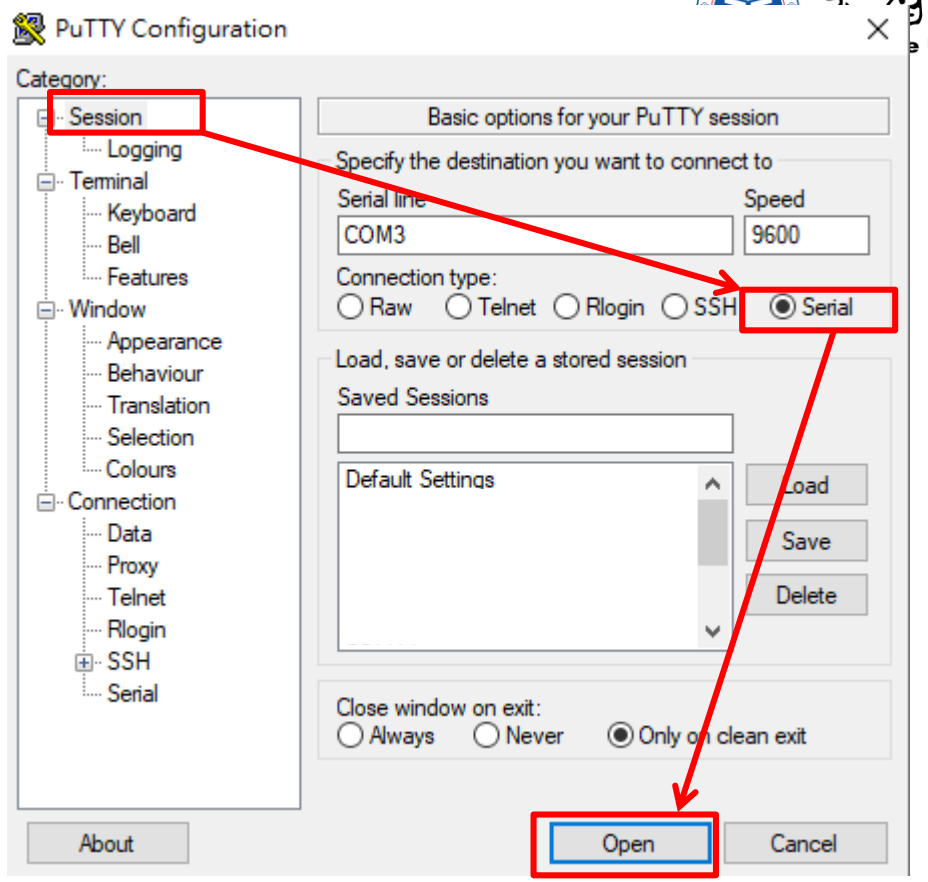
$ sudo minicom -D /dev/ttyAMA1 -b 9600



COM@PC


連接埠 (COM 和 LPT)
Prolific USB-to-Serial Comm Port (COM3)

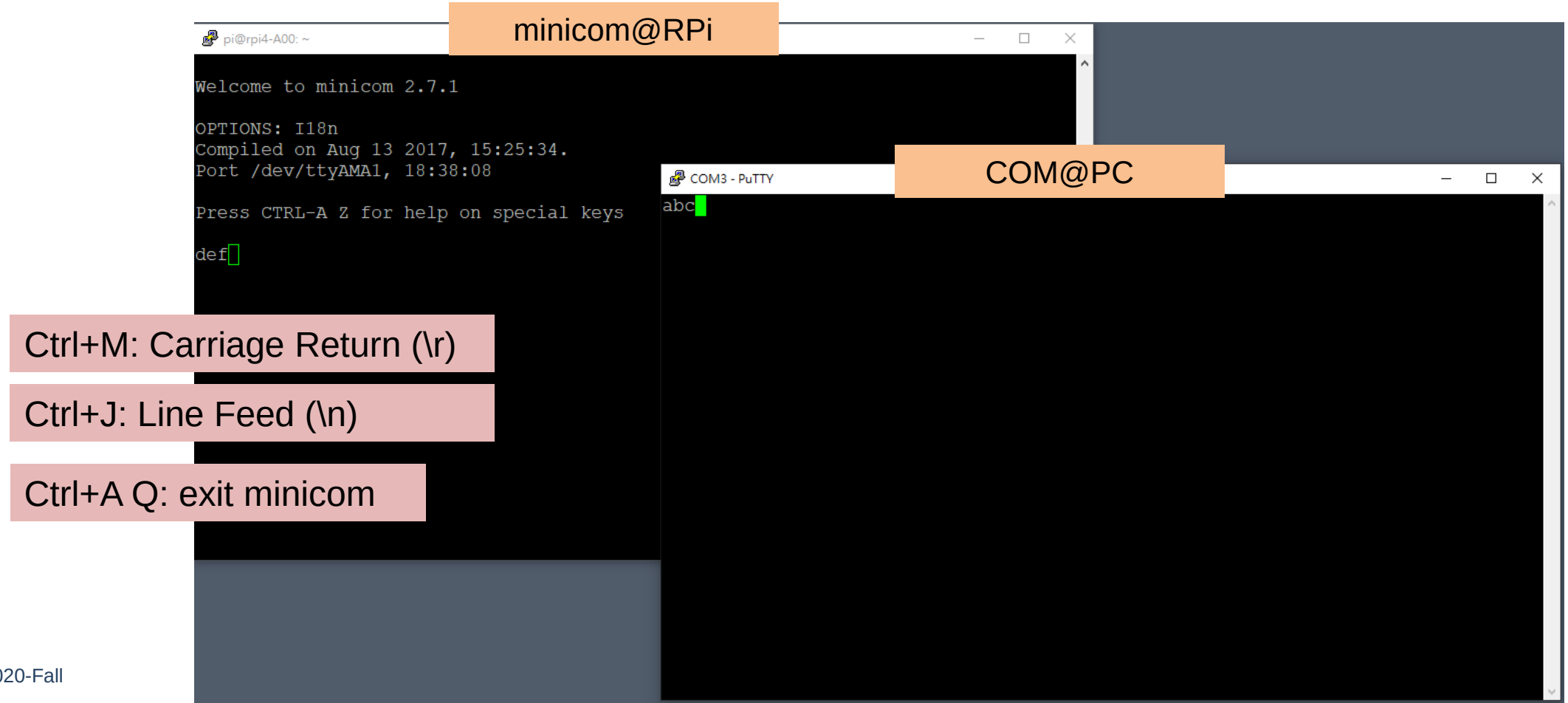

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Options controlling local serial lines
Select a serial line
Serial line to connect to
COM3
Configure the serial line
Speed (baud)
9600
Data bits
8
Stop bits
1
Parity
None
Flow control
None
About
Open
Cancel


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line
Speed
COM3
9600
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel


COM3 - PuTTY

Test

- Type some characters on COM@PC and they will be shown on minicom@RPi.
- Type some characters on minicom@RPi and they will be shown on COM@PC.



Outline

- PWM
  - LED Control
  - Buzzer
- UART
  - Terminal Program
  - Python Program
- Lab

PC Integration (1/2)

- Exit minicom on RPi.
- Keep the COM connection on PC.

$ python3 rpi_serial_echo.py

rpi_serial_echo.py

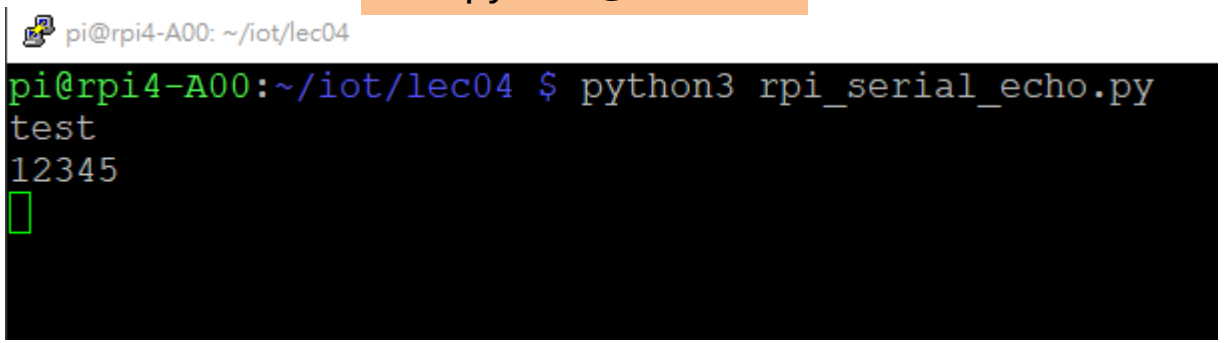
```
import time
import serial

ser = serial.Serial('/dev/ttyAMA1', baudrate=9600,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    bytesize=serial.EIGHTBITS
                    )
try:
    ser.write(b'Hello World\r\n')
    ser.write(b'Serial Communication Using Raspberry Pi\r\n')
    while True:
        data = ser.readline()
        print(data.decode("utf-8").strip())
        ser.write(data)
        ser.flushInput()
        time.sleep(0.1)
except KeyboardInterrupt:
    pass
finally:
    ser.close()
```

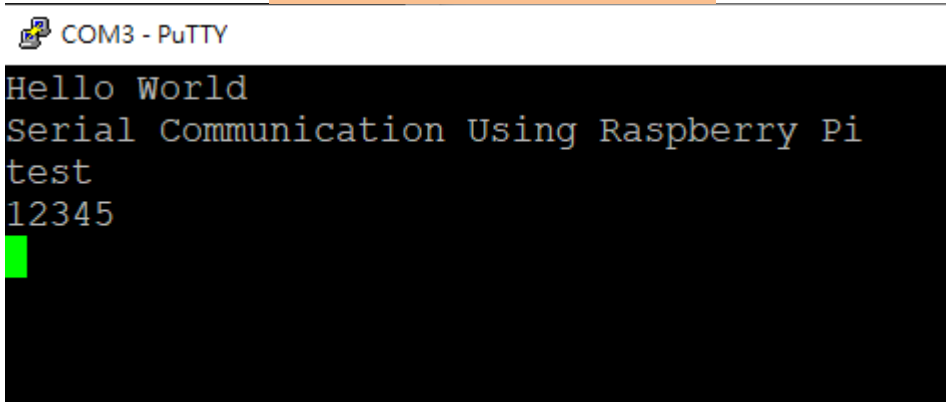
PC Integration (2/2)

- You will see “Hello …” in COM@PC.
- When you send a message in COM@PC, python@RPi will receive and echo it back.
  - Sending a line in python@RPi “test” Ctrl+M Ctrl+J
  - Sending a line in python@RPi “12345” Ctrl+M Ctrl+J

python@RPi



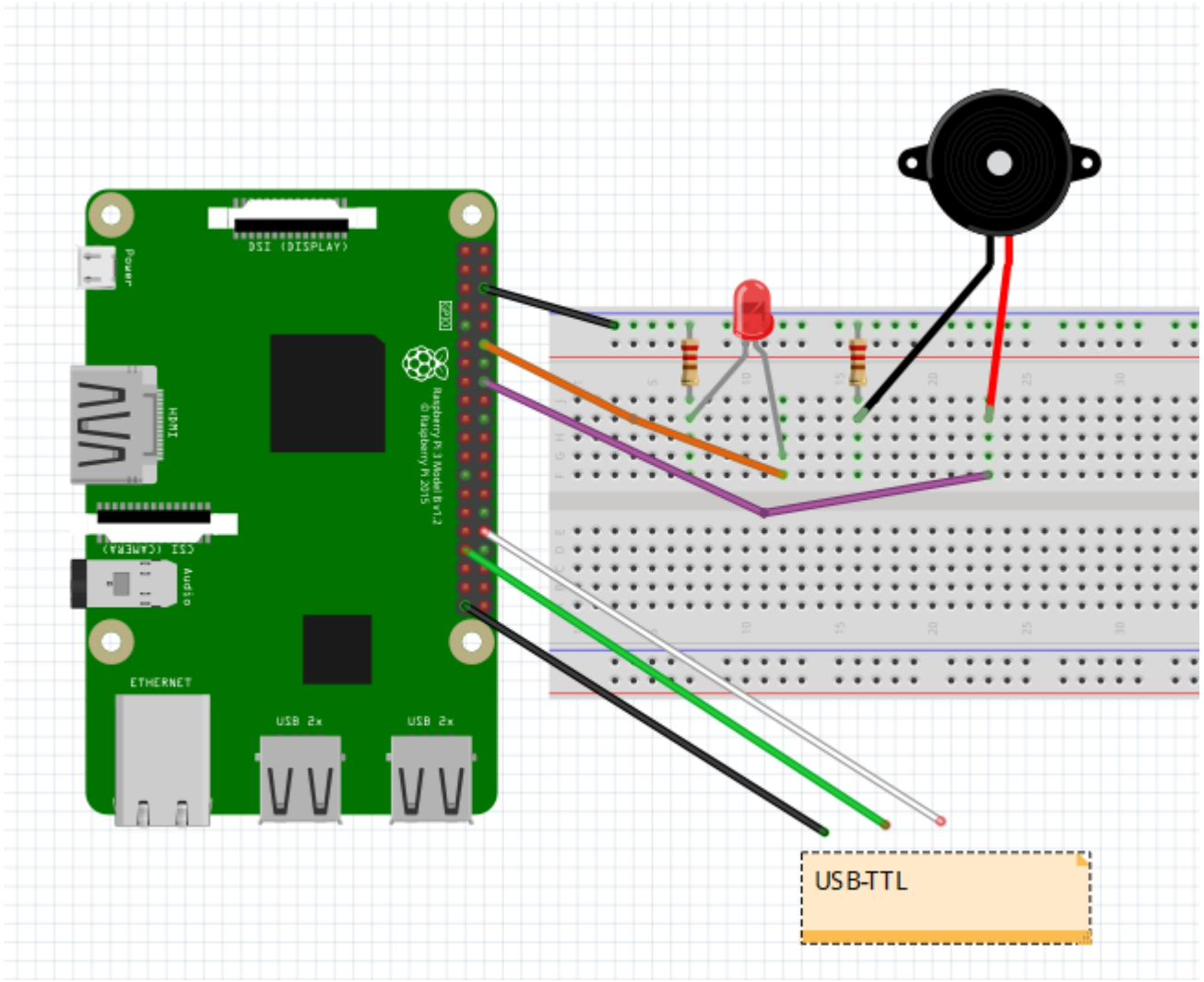
COM@PC



Outline

- PWM
  - LED Control
  - Buzzer
- UART
  - Terminal Program
  - Python Program
- Lab

Circuit


DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
USB-TTL

CS348A Lab

- Control the tone of buzzer and brightness of LED by the serial input from PC.
- The LED is turned off at the beginning.
- “play x1,x2,x3…”: Play each specified tone for 2 seconds in sequence. (x: c,d,e,f,g,a,b)
  - Ex: “play d,e,c” means that tune the frequency to 294 for 2 seconds, to 330 for 2 seconds, and to 262 for 2 seconds.

	1	3	5	6	8	10	12
音名	C	D	E	F	G	A	B
4	262	294	330	349	392	440	493

- “set xxx”: control the LED brightness to xxx% (xxx is 0 ~ 100.)
  - Ex: “set 60” means that set 60% brightness to the LED.

CS348B Lab

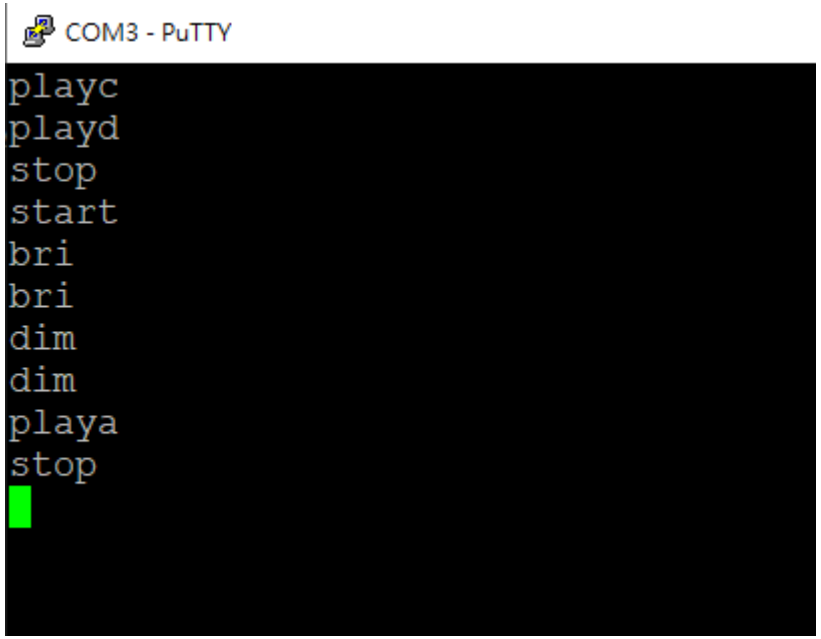
- Control the tone of buzzer and brightness of LED by the serial input from PC.
- The LED is turned off at the beginning.
- “playx”: Set the tone of buzzer to the corresponding frequency. (x: c,d,e,f,g,a,b)
  - Ex: “playd” to tune the frequency to 294.

	1	3	5	6	8	10	12
音名	C	D	E	F	G	A	B
4	262	294	330	349	392	440	493

- “stop”: Stop the sound.
- “bri”: Increase 10% brightness.
- “dim”: Decrease 10% brightness.

Note

- Input example
  - Add Ctrl+M Ctrl+J at the end when inputting each command.



Sample Files

$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec04/01_pwm_led_control.py

$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec04/02_pwm_breath.py

$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec04/03_pwm_led_hw.py

$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec04/04_buzzer.py

$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec04/05_serial_echo.py