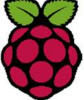


ปฏิบัติการบน RaspberryPi ครั้งที่ 1:

ทำความคุ้นเคยกับสภาพแวดล้อมของ Raspberry Pi OS

ซีงเกิลบอร์ด Raspberry Pi เป็นซีงเกิลบอร์ดที่ใช้ชิพ SoC Broadcom BCM2837 ซึ่งซีฟียภายในเป็น ARM Cortex A53 (ARMv8 instruction set) จำนวน 4 คอร์ ทำงานที่ 1.2 GHz

	Raspberry Pi 4 Model B	Raspberry Pi3 Model B	Raspberry Pi Zero	Raspberry Pi 2 Model B	Raspberry Pi Model B+
Introduction Date	24/6/2019	29/2/2016	25/11/2015	2/2/20185	14/7/2014
SoC	BCM2711	BCM2837	BCM2835	BCM2836	BCM2835
CPU	Quad core A72 @ 1.5GHz (64bit)	Quad Cortex A53 @ 1.2GHz	ARM11 @ 1GHz	Quad Cortex A7 @ 900MHz	ARM11 @ 700MHz
Instruction set	ARMv8	ARMv8-A	ARMv6	ARMv7-A	ARMV6
GPU	500MHz VideoCore VI	400Mz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV
RAM	2GB	1GB SDRAM	512MB SDRAM	1GB SDRAM	512MB SDRAM
Storage	miroSD	microSD	microSD	microSD	microSD
Ethernet	Gigabit	10/100	none	10/100	10/100
Wireless	802.11ac/Bluetooth5.0	802.11n/Bluetooth4.0	none	none	none
Video Output	dual microHDMI	HDMI	HDMI	HDMI/Composite	HDMI/Composite
Audio Output	HDMI/headphone	HDMI/headphone	HDMI	HDMI/headphone	HDMI/headphone
GPIO	40	40	40	40	40


ระบบปฏิบัติการที่ใช้ เป็นลินุกซ์ ซึ่งมีหลายดิสทริบิวชันที่ port มาลงโดยเฉพาะ ตัวที่จะใช้ในปฏิบัติการจะเป็น Raspberry Pi OS ซึ่งพอร์ตมาจากกลุ่ม debian ดิสทริบิวชันอื่นที่นิยมใช้ซึ่งมาจากกลุ่มเดียวกันก็อย่างเช่น Ubuntu ด้วยเหตุที่โครงสร้างโดยทั่วไปใกล้เคียงกับ Ubuntu ชุดคำสั่งต่างๆ ที่ใช้ผ่าน terminal จึงใกล้เคียงกันมาก

ในปฏิบัติการชุดนี้ เราจะใช้ Code::Blocks เพื่อพัฒนาซอฟต์แวร์ต่างๆ โดยเขียนด้วยภาษา C ที่น่าสนใจก็คือ แม้ว่าสถาปัตยกรรมทางฮาร์ดแวร์ของ Raspberry Pi นั้นแตกต่างจากพีซีอยู่มากพอสมควร แต่ด้วยไลบรารีต่างๆ ที่ GCC มีให้ ทำให้เราสามารถโอนโปรแกรมตัวอย่างสำหรับลินุกซ์ทั้งหมดที่ได้เรียนมาก่อนหน้านี้ ไปรันบน Raspberry Pi ได้ทั้งหมด รวมทั้งตัวอย่างที่มีการใช้ atomic instruction set ซึ่งไลบรารีของ GCC บน pi นั้นมีการปรับให้แปลไปเป็นชุดคำสั่งของ ARM แทน

ลำดับการติดตั้งซอฟต์แวร์บน pi

- 1) ดาวน์โหลด SD card image ของตัว Raspberry Pi OS จากเว็บไซต์ <https://www.raspberrypi.org/downloads/>
- 2) ดาวน์โหลด Balena Etcher จาก <https://www.balena.io/etcher/> และติดตั้งบนพีซี
- 3) ใช้ Etcher จากข้อ 2) เพื่อเขียน image ของ Raspberry Pi OS ลงใน microSD card ที่มีขนาดไม่น้อยกว่า 8GB (เนื่องจากตัว image ตัวเต็มของ Raspberry Pi OS มีขนาดใหญ่กว่า 4GB) อนึ่ง ตัว Etcher จะมีกลไกการขยายไฟล์ zip เพื่อเป็นอิมเมจให้อัตโนมัติ ดังนั้นไม่ต้องแตกไฟล์ zip ที่ได้จากข้อ 1) แต่ประการใด และควรให้ Etcher verify ด้วยว่าสามารถเขียนได้ถูกต้องหรือไม่
- 4) เสียบการ์ด microSD ที่เขียนเสร็จแล้วลงใน pi จากนั้นเสียบสายจอผ่านช่อง HDMI เสียบคีย์บอร์ดและเมาส์ที่ช่อง USB จากนั้นจึงเสียบอะแดปเตอร์จ่ายไฟให้ที่ช่อง power ที่เป็น microUSB pi จะบูตในทันที
- 5) ทำการอัปเดตระบบปฏิบัติการและเช็คการใช้งานระบบในส่วนต่างๆ

การใช้ `sudo` นำหน้าคำสั่งต่างๆ เพื่อยกระดับสิทธิ์
การใช้คำสั่งในระดับ `root` เพราะคำสั่งเหล่านี้
อนุญาตให้ `root` ทำงานได้เท่านั้น

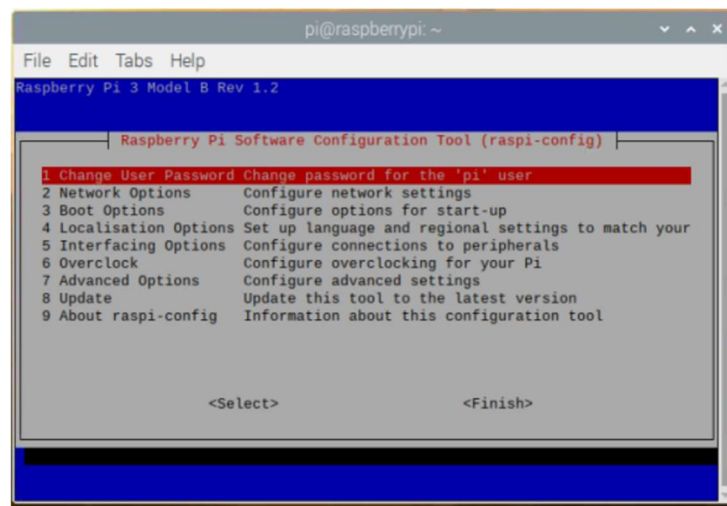
- เชื่อมต่อ pi ทางไวไฟหรือ ethernet (การเชื่อมต่อผ่านไวไฟ คลิกที่ไอคอนไวไฟ และคลิกเลือก access point ที่พบ และใส่ passphrase ตามที่ร้องขอ)
- รันเทอร์มินัลโดยการคลิกที่เมนู เลือก Accessories/Terminal หรือคลิกที่ไอคอน  ที่ทาสก์บาร์ด้านบน
- ภายในเทอร์มินัล สั่ง

`sudo apt-get update` (สั่งให้อัปเดตรายการแพ็คเกจที่มีรองรับทั้งหมดจากเว็บ)

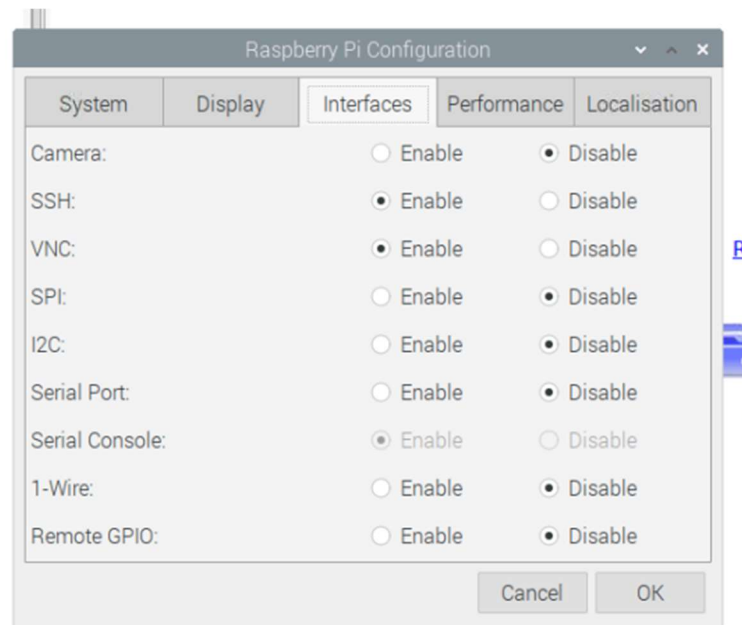
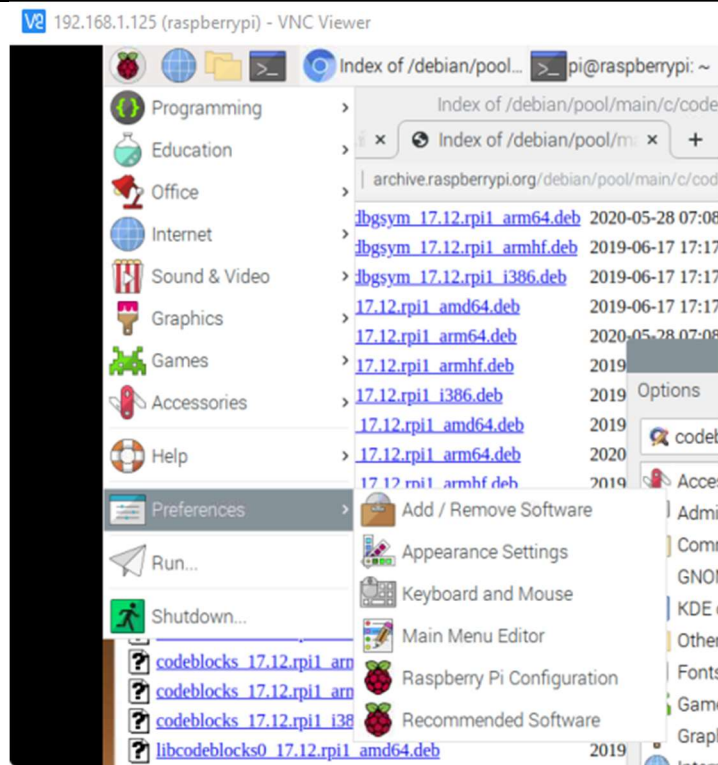
`sudo apt-get upgrade` (สั่งให้ตรวจแพ็คเกจทั้งหมดที่ติดตั้ง และอัปเดตแพ็คเกจทั้งหมดให้เป็นเวอร์ชันล่าสุด)

หมายเหตุ ระบบเครือข่ายขององค์กรอาจจะมีการบล็อกพอร์ตบางพอร์ต หรือมีความเร็วในการเชื่อมต่อที่ช้า ทำให้เกิดปัญหาในการเชื่อมต่อขณะดาวน์โหลดไฟล์ได้ (แม้ว่าอาจจะสามารถเข้าถึง URL ปลายทางได้ก็ตาม) ดังนั้นหากพบว่ามีปัญหา ให้เปลี่ยนไปใช้ระบบเครือข่ายอื่นแทน)

- จากนั้นใช้คำสั่ง `sudo raspi-config` เพื่อเช็คระบบให้เป็นไปตามที่เราต้องการ



หรืออาจใช้เมนู Preferences/ Raspberry Pi Configuration เพื่อเข้าสู่หน้าต่างการปรับแต่งก็ได้เช่นกัน

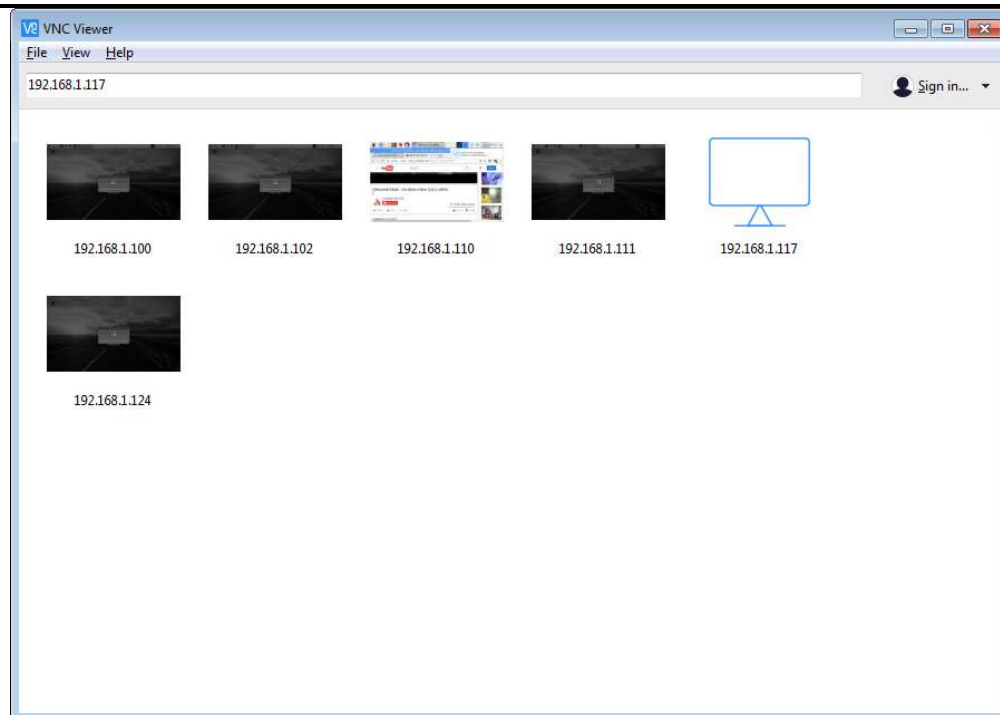


ส่วนการปรับแต่งใน raspi-config ที่น่าสนใจ

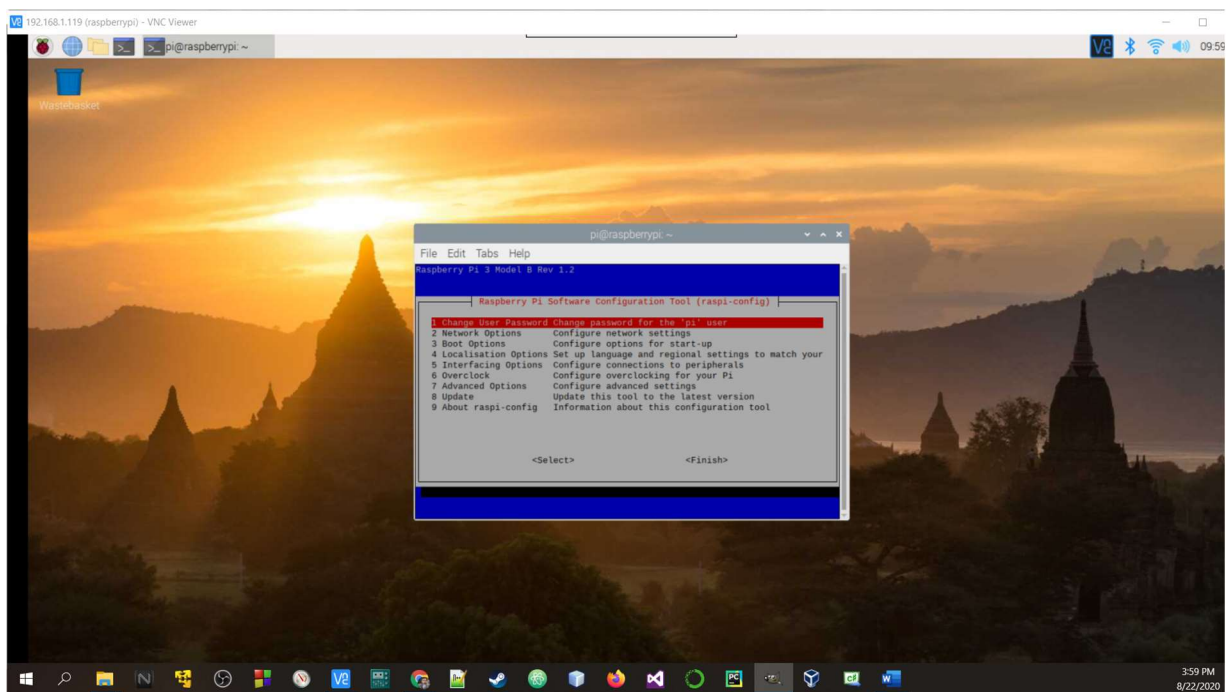
Advanced Options/Expand Filesystem ในกรณีที่เรานำ microSD ที่มีความจุใหญ่ ตอนเราเขียน image ของ Raspberry Pi OS มาลง จะมีพื้นที่เหลือที่ไม่ได้ใช้งาน เราใช้ฟังก์ชันนี้เพื่อขยายขนาดของพาร์ติชันบน microSD ให้กินพื้นที่ทั้งหมดได้ (ตามปกติแล้วเวลาเราครั้งแรก Raspberry Pi OS จะขยายพื้นที่พาร์ติชันให้โดยอัตโนมัติอยู่แล้ว ดังนั้นจึงไม่จำเป็นต้องสั่งเพิ่มเติมอีก)

Interfacing Options/SSH การสั่งเปิดปิดเซิร์ฟเวอร์ SSH ซึ่งจะทำให้เราสามารถเข้า WinSCP และ Putty จากภายนอกเข้ามาได้ผ่านทางระบบเครือข่าย อนึ่ง เนื่องจากบัญชีผู้ใช้งานเริ่มต้นคือ pi และพาสเวิร์ดคือ raspberry หากเราเปิดใช้เซิร์ฟเวอร์นี้ก็ควรที่จะเปลี่ยนรหัสผ่านเสียด้วยก็จะปลอดภัยมากขึ้น (หากไม่เปลี่ยน เวลาเราเชื่อมต่อ Raspberry Pi OS จะเตือนให้เราทราบทุกครั้ง)

Interfacing Options/VNC การสั่งเปิดปิดเซิร์ฟเวอร์ VNC ที่จะทำให้เราสามารถเข้าโปรแกรม RealVNC เชื่อมต่อเข้ามาได้



หน้าต่าง VNC Viewer ที่บันทึกไอพีของเครื่องลูกข่ายที่เคยใช้งานมาก่อนหน้า



ตัวอย่างหน้าต่าง VNC Viewer ที่เชื่อมต่อกับ Raspberry pi

Interfacing Options/I2C และ SPI เป็นส่วนการเปิดปิดวงจรการทำงานในส่วน I2C และ SPI ของ pi ซึ่งในตั้งต้นจะปิดอยู่ ดังนั้นหากต้องต่อวงจรผ่านอินเทอร์เฟซทั้งสอง ต้องมาเปิดบริการในส่วนนี้ด้วย

ตัวอย่างการเซ็ตวันเวลาให้เป็นปัจจุบัน

```
sudo date -s "Sat Nov 21 20:14:11 2020"
```

การโปรแกรมและใช้งาน raspberry pi

เพื่อความสะดวกและต่อเนื่องในคลาสนี้ ให้นักศึกษาติดตั้งตัว Code::Blocks ลงใน Raspberry Pi OS เพิ่มเติม โดยใช้คำสั่ง

```
sudo apt-get install codeblocks
```

เมื่อติดตั้งเสร็จ ตัว Code::Blocks สามารถเรียกใช้ได้ผ่านทางเมนู Programming/Code::Blocks IDE

การใช้งาน Code::Blocks บน pi นั้นเหมือนกับการใช้งานบนระบบปฏิบัติการอื่นๆ และแพลตฟอร์มอื่นๆ

เมื่อนักศึกษาจะเลิกใช้งาน pi ให้นักศึกษาทำการ shutdown ระบบปฏิบัติการให้เรียบร้อย (รอนจนกว่าไฟแสดงการทำงานของบอร์ดเป็นสีแดงนิ่งๆ) แล้วจึงค่อยถอดสายไฟแฉะเตอร์ออกกะครีบ การถอดสายไฟเลี้ยงออกทันทีอาจจะทำให้ข้อมูลบางส่วนเสียหายได้ (ทำนองเดียวกันกับการใช้งานลินุกซ์และวินโดวส์โดยทั่วไป)

การใช้งาน raspberry pi ในส่วน GPIO

ตัว Raspberry Pi OS ที่ติดตั้งนี้ มีการลงไลบรารีช่วยในการติดต่อ I/O ต่างๆ ของ Pi มาแล้ว ตัวไลบรารีที่น่าสนใจอย่างเช่น wiringPi และ pigpio รองรับการเขียนโปรแกรมควบคุมด้วยภาษา C และภาษา Python โดยสามารถเริ่มเขียนโปรแกรมในทั้งสองภาษาได้โดยทันที

ในชุดบทเรียนนี้ เราจะหันมาใช้ pigpio เพื่อทำงานเป็นหลัก ทั้งนี้เนื่องจาก wiringPi นั้นได้หยุดการพัฒนาเพื่อเผยแพร่แล้ว โดยเวอร์ชันสุดท้ายที่มีให้ใช้และรองรับอย่างเป็นทางการ เป็นเวอร์ชันที่รองรับ Raspberry Pi 3

สำหรับการดูสถานะของพอร์ตต่างๆ บน pi เราสามารถเรียกใช้คำสั่ง gpio readall เพื่อดูสถานะปัจจุบันของพอร์ตต่างๆ ใน GPIO

หมายเหตุ `gpio` เป็นโปรแกรมที่ให้มาพร้อมกับ `wiringPi` ในกรณีที่ใช้บอร์ดรุ่นที่ไม่สามารถใช้คำสั่งนี้ได้ ให้คอมไพล์โปรแกรมตัวอย่างที่ให้ไว้ในส่วนปฏิบัติการตัวอย่างนี้ เพื่อนำไปใช้งานแทน

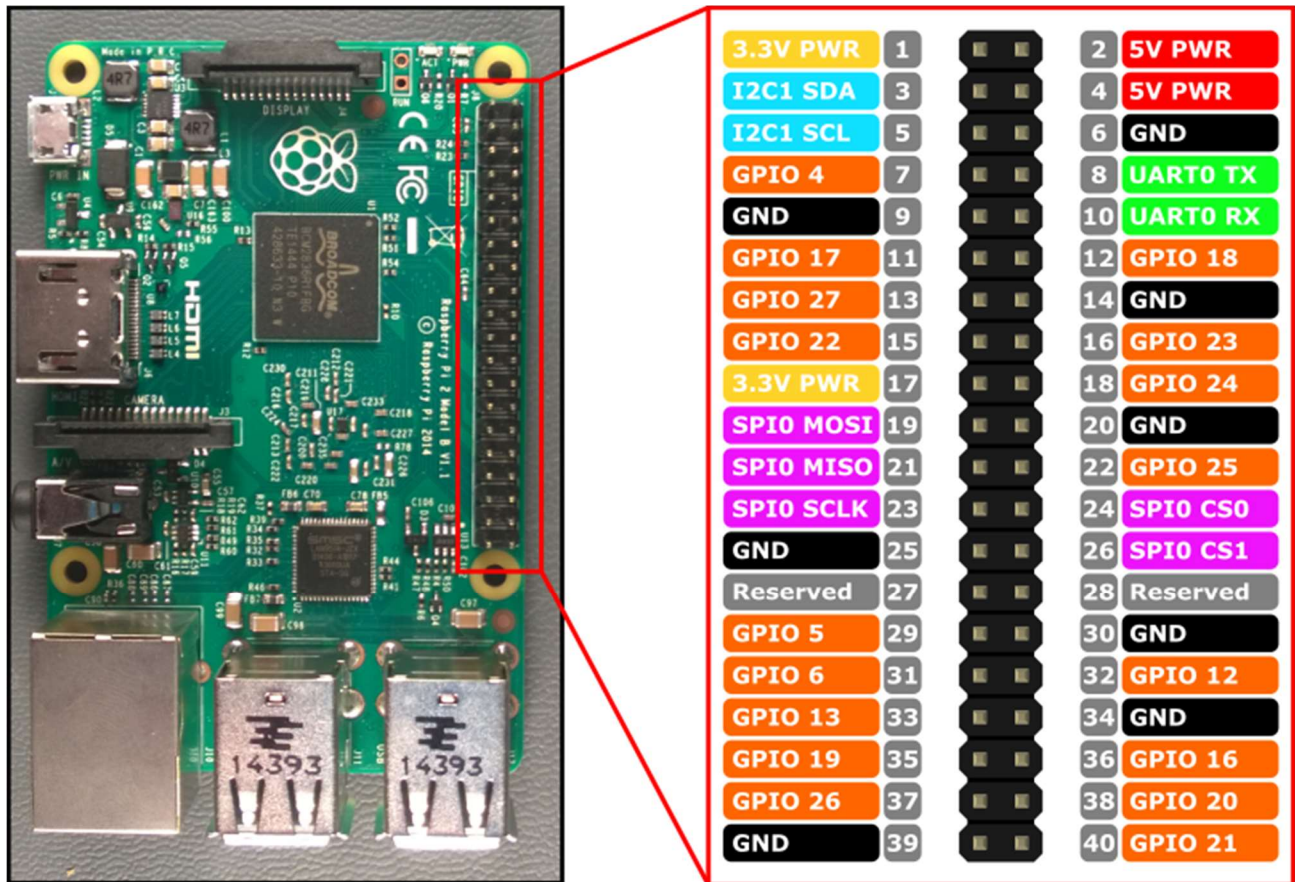
```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ gpio readall
+-----Pi 3-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+
| 2 | 8 | 3.3v | | | 1 | 2 | | | 5v | | |
| 3 | 9 | SDA.1 | ALTO | 1 | 3 | 4 | | | 5v | | |
| 4 | 7 | SCL.1 | ALTO | 1 | 5 | 6 | | | 0v | | |
| 4 | 7 | GPIO. 7 | IN | 1 | 7 | 8 | 1 | ALT5 | TxD | 15 | 14 |
| | | | | | 9 | 10 | 1 | ALT5 | RxD | 16 | 15 |
| 17 | 0 | GPIO. 0 | IN | 0 | 11 | 12 | 0 | IN | GPIO. 1 | 1 | 18 |
| 27 | 2 | GPIO. 2 | OUT | 0 | 13 | 14 | | | 0v | | |
| 22 | 3 | GPIO. 3 | OUT | 1 | 15 | 16 | 1 | OUT | GPIO. 4 | 4 | 23 |
| | | | | | 17 | 18 | 0 | OUT | GPIO. 5 | 5 | 24 |
| 10 | 12 | MOSI | IN | 0 | 19 | 20 | | | 0v | | |
| 9 | 13 | MISO | IN | 0 | 21 | 22 | 0 | IN | GPIO. 6 | 6 | 25 |
| 11 | 14 | SCLK | IN | 0 | 23 | 24 | 1 | IN | CE0 | 10 | 8 |
| | | | | | 25 | 26 | 1 | IN | CE1 | 11 | 7 |
| 0 | 30 | SDA.0 | IN | 1 | 27 | 28 | 1 | IN | SCL.0 | 31 | 1 |
| 5 | 21 | GPIO.21 | OUT | 1 | 29 | 30 | | | 0v | | |
| 6 | 22 | GPIO.22 | OUT | 1 | 31 | 32 | 1 | IN | GPIO.26 | 26 | 12 |
| 13 | 23 | GPIO.23 | IN | 0 | 33 | 34 | | | 0v | | |
| 19 | 24 | GPIO.24 | IN | 0 | 35 | 36 | 1 | IN | GPIO.27 | 27 | 16 |
| 26 | 25 | GPIO.25 | IN | 1 | 37 | 38 | 1 | IN | GPIO.28 | 28 | 20 |
| | | | | | 39 | 40 | 1 | IN | GPIO.29 | 29 | 21 |
+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+
pi@raspberrypi:~ $

```


จากตารางข้างต้น เนื่องจากเราใช้ pigpio ดังนั้นตัวเลขพอร์ตที่จะใช้ จะใช้ตัวเลขที่ปรากฏในช่อง BCM เป็นหลัก

บอร์ด Raspberry pi และ pin GPIO นั้นมีรายละเอียดดังรูปนี้ ขอให้นักศึกษาใช้ความระมัดระวังในการเชื่อมต่อวงจร เนื่องจากตัวชิพ SoC นั้นไม่มีกลไกป้องกัน การป้อนสัญญาณผิด สามารถทำให้บอร์ดเสียหายได้

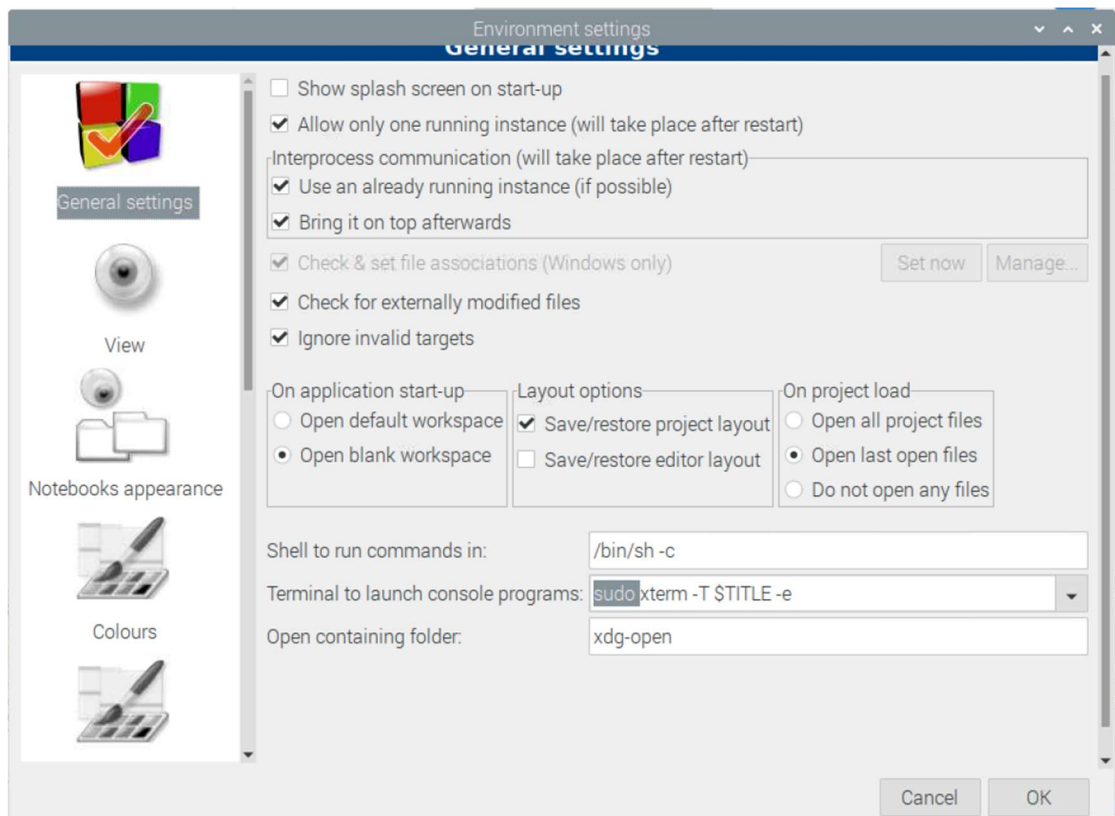
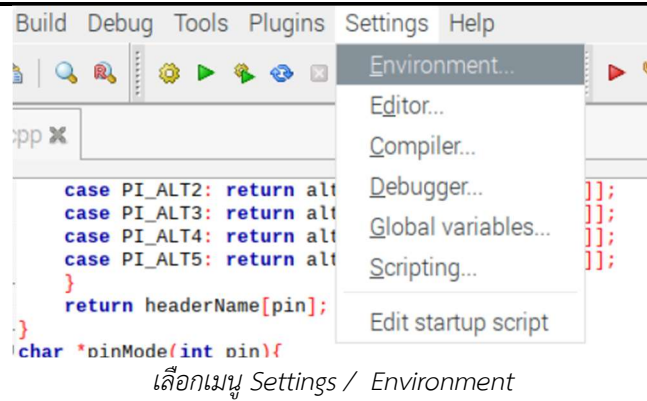


พอร์ตต่างๆ ที่ปรากฏในรูปข้างบนนี้ ยกเว้นพอร์ตจ่ายไฟ 5v พอร์ตที่เหลือทั้งหมดมีระดับสัญญาณเป็น 3.3โวลต์ ดังนั้นการเชื่อมต่อ GPIO กับวงจรภายนอกเช่น วงจรอัลตราโซนิกหรือเซอร์โว จำเป็นต้องต่อวงจรระดับสัญญาณ (Logic shifter) คั่นไว้ด้วย

รวมฟังก์ชันพื้นฐานสำหรับ pigpio

การใช้งานไลบรารี pigpio นั้นมีอยู่สามแบบด้วยกัน แบบแรกคือการเรียกใช้ฟังก์ชันที่ตัวไลบรารีผูกไว้กับโปรแกรมที่เราพัฒนาขึ้นเลย ซึ่งในปฏิบัติการทั้งหมดเราจะเลือกใช้กลไกนี้ ส่วนแบบที่สองเป็นการเรียกใช้ฟังก์ชันผ่านไดรเวอร์ ซึ่งจะต้องรันตัวไดรเวอร์ (daemon) เสียก่อนด้วยคำสั่ง `sudo pigpiod` ข้อดีของการใช้งานแบบไดรเวอร์นั้น ตัวบอร์ดที่ถูกควบคุม ซึ่งต้องลงตัวไดรเวอร์ pigpiod กับบอร์ดหรือคอมพิวเตอร์ที่เราเขียนโปรแกรมสั่งงานนั้นไม่จำเป็นต้องเป็นเครื่องเดียวกัน โดยเราสามารถระบุหมายเลขไอพีและพอร์ตที่เชื่อมต่อได้ และแบบที่สาม เป็นการเรียกใช้ผ่านคำสั่ง `pigs` ในเชลล์ของระบบปฏิบัติการ ซึ่งจะต้องรันตัว pigpiod ก่อนด้วยเช่นกัน

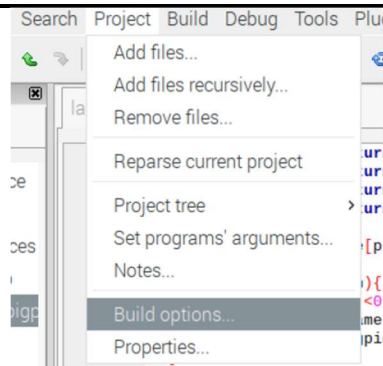
สำหรับการใช้งานไลบรารีของ pigpio แบบรวมไลบรารีไว้กับโปรแกรมของเราด้วยเลยนั้น มีข้อเสียอยู่ตรงที่เราจะต้องรันโปรแกรมในระดับ root (ต้องใช้ `sudo` กับตัวโปรแกรมของเราเพื่อยกระดับสิทธิการใช้งาน) ในขณะที่การทำงานในอีกสองแบบนี้ ตัวโปรแกรมที่รันไม่จำเป็นต้องมีสิทธิในระดับ root (แต่ตัว pigpiod ต้องรันในระดับ root) ดังนั้น ก่อนที่เราจะเริ่มเขียนโปรแกรมแรก เราจะเซตคำสั่งรันโปรแกรมภายใน Code::blocks เพื่อให้รันในระดับ root ได้โดยเลือกเมนู Settings/Environment แล้วแก้ไขเพิ่มเติม `sudo` ในจุดที่สั่งรันโปรแกรมผ่าน xterm ดังรูป



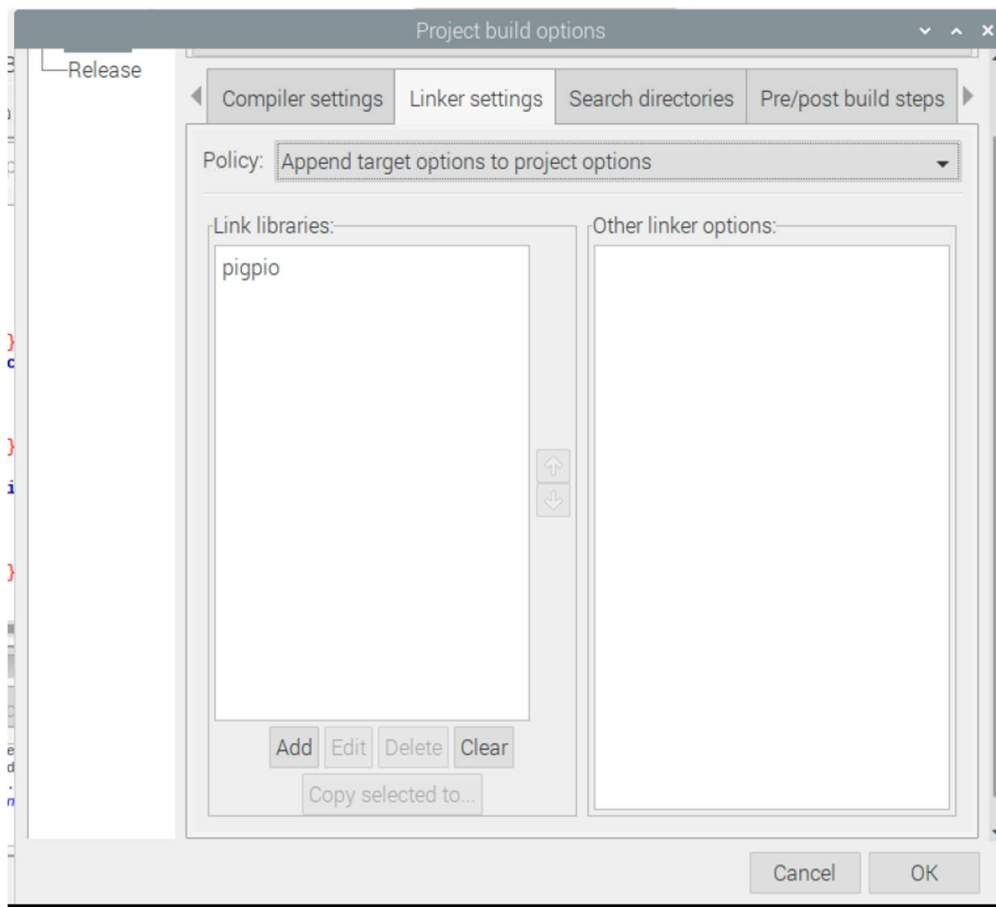
ที่แท็บ General Settings ตรงช่อง Terminal to launch... ให้เพิ่มคำ sudo (และเว้นวรรค) นำหน้า xterm ซึ่งคือตัวเทอร์มินัลที่เราจะใช้รันโปรแกรม

ในการเรียกใช้งานไลบรารีของ pigpio นักศึกษาจะต้องเรียกใช้ไลบรารีนี้ใน Code::blocks เพื่อรวมไว้ในโปรเจกต์ และ #include ตัว pigpio.h ด้วย

```
#include <pigpio.h>
```



เลือกเมนู Project / Build options...



จากนั้น เลือกแท็บ Linker settings แล้วคลิก Add เพื่อเพิ่ม pigpio เข้าไป และกด OK เพื่อยืนยันการเปลี่ยนแปลง

สำหรับฟังก์ชันพื้นฐานของ pigpio ที่ควรรู้จักในเบื้องต้น มีดังนี้

```
int gpioInitialise(void);
```

ในการเริ่มต้นใช้งาน pigpio จะต้องเรียกใช้ฟังก์ชัน gpioInitialise() เพื่อเช็คตัวไลบรารีให้พร้อมรับการทำงานต่างๆ ที่จะมีต่อมา ตัวอย่างการใช้งานดังเช่น

```
if(gpioInitialise() < 0){
    return 1;
}
```

```
void gpioTerminate(void);
```


ก่อนจบโปรแกรม ควรสั่งหยุดการทำงานไลบรารี pigpio ด้วยฟังก์ชัน gpioTerminate() เพื่อคืนทรัพยากรต่างๆ ที่ไลบรารีนำมาใช้งาน ให้กับระบบ

ตัวอย่างการใช้งาน เรียกใช้งานในลักษณะข้อความสั่งดังนี้

```
gpioTerminate();
```

```
void gpioSetMode(int pin,int mode);
```

กำหนดให้ขา GPIO ที่กำหนดทำหน้าที่ตามต้องการ

pin หมายเลขขา

mode ในเบื้องต้นเราจะใช้โหมดดังนี้คือ PI_INPUT เพื่อให้เป็นอินพุต และ PI_OUTPUT เพื่อให้เป็นเอาต์พุต

```
void gpioSetPullUpDown(int pin,int pud);
```

กำหนดให้ขา GPIO ที่กำหนด ต่อกับวงจร pull up หรือ pull down กับ R เป็นการภายใน ซึ่งจะทำให้เราสามารถเช็คสถานะขา output ให้เป็น 0 หรือ 1 ในขณะที่เราปล่อยขาลอยได้

pin หมายเลขขา

pud มีให้เลือกคือ PI_PUD_OFF (ไม่ต่อวงจร) PI_PUD_UP (ต่อขึ้น vcc) PI_PUD_DOWN (ต่อลง gnd)

```
void gpioWrite(int pin,int value);
```

ส่งสัญญาณออกขาเอาต์พุต

pin หมายเลขขา

value เป็น 0 หรือ 1

```
int gpioRead(int pin);
```

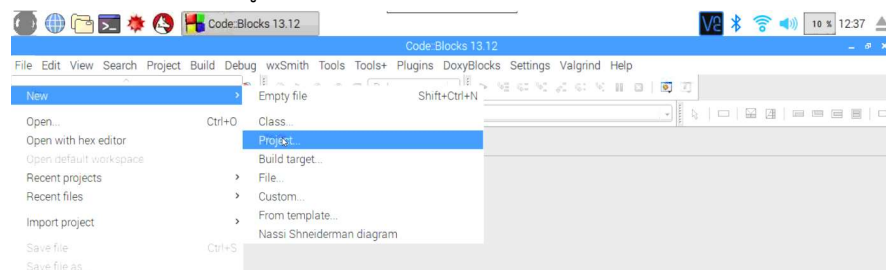
อ่านสัญญาณจากขาอินพุต

pin หมายเลขขา

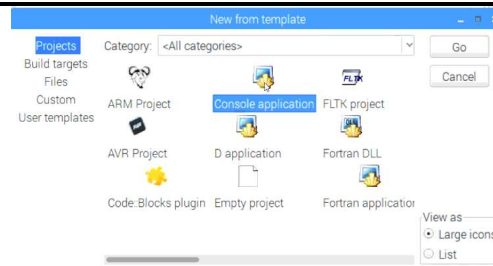
ค่ากลับคืนเป็น 0 หรือ 1

การใช้โปรแกรม Code::Blocks ร่วมกับ gcc ในการพัฒนาโปรแกรมบน raspberry pi

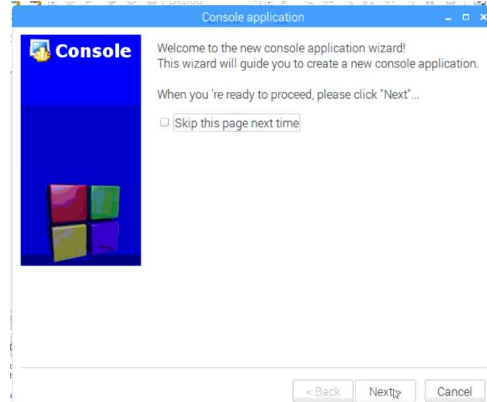
เริ่มต้นด้วยการเปิดโปรเจกต์ใหม่โดยเลือกเมนู File/New/Project... เพื่อสร้างโปรเจกต์ใหม่



เลือก Console application



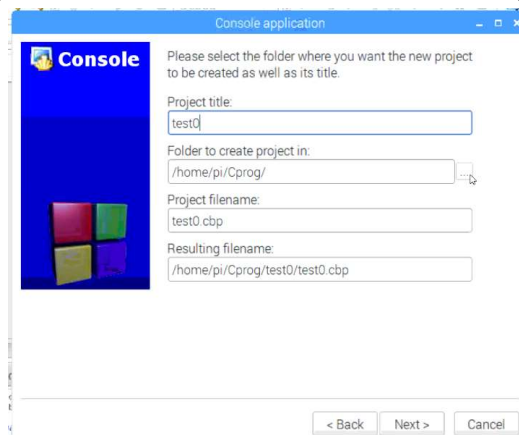
กดเลือก Skip this page next time แล้วกด Next (เพื่อในภายหลังจะได้ไม่ต้องแสดงหน้าจออธิบายหน้านี้อีก)



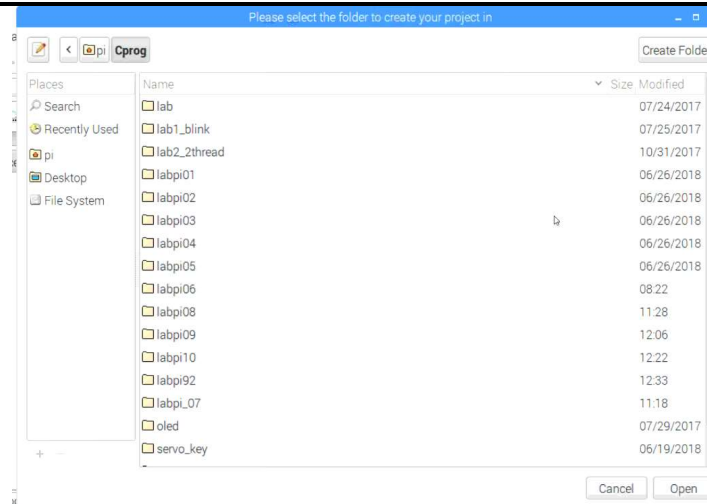
เลือกกฎการแปลว่าต้องการใช้ภาษาของภาษา C หรือ C++ ในที่นี้เนื่องจากนักศึกษาอาจจะคุ้นเคยกับการใช้ iostream หรือกฎปลั๊กย่อยอื่นๆ ของ C++ เราก็จะเลือก C++ แทน แม้ว่าโปรแกรมอาจจะเขียนโดยใช้กลไกของภาษาซีเป็นหลักก็ตาม



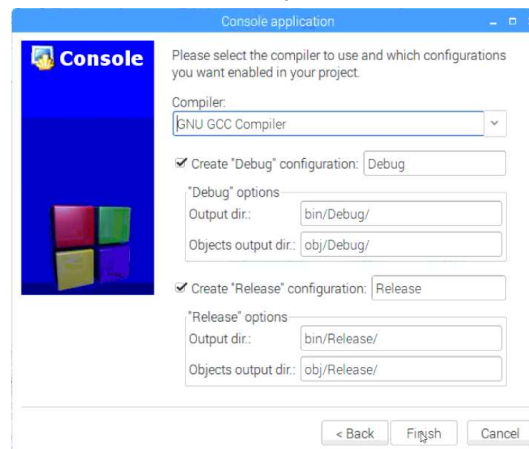
พิมพ์ชื่อโปรเจกต์ ซึ่งจะกลายเป็นชื่อโปรแกรมที่ใช้รันต่อไป ทั้งนี้ในช่อง Folder to create project in: จะยังคงว่างอยู่หากเพิ่งใช้งาน Code::blocks เป็นครั้งแรก ให้คลิกปุ่ม ... ที่อยู่ด้านขวามือเพื่อค้นหาไปยังโฟลเดอร์ที่ต้องการจะบรรจุไฟล์โปรเจกต์ทั้งหมดไว้



เมื่อใช้งานตัว Code::blocks เป็นครั้งแรก (หรือหากต้องการเปลี่ยนโฟลเดอร์หลักที่ใช้งาน) เมื่อคลิกปุ่ม ... เข้ามาก็จะสามารถเลือกโฟลเดอร์หรือสร้างโฟลเดอร์ใหม่ได้ตามต้องการ (จะต้องเลือกโฟลเดอร์ให้ปรากฏขึ้น เพื่อให้โปรแกรมทำงานต่อไปได้อย่างถูกต้อง)

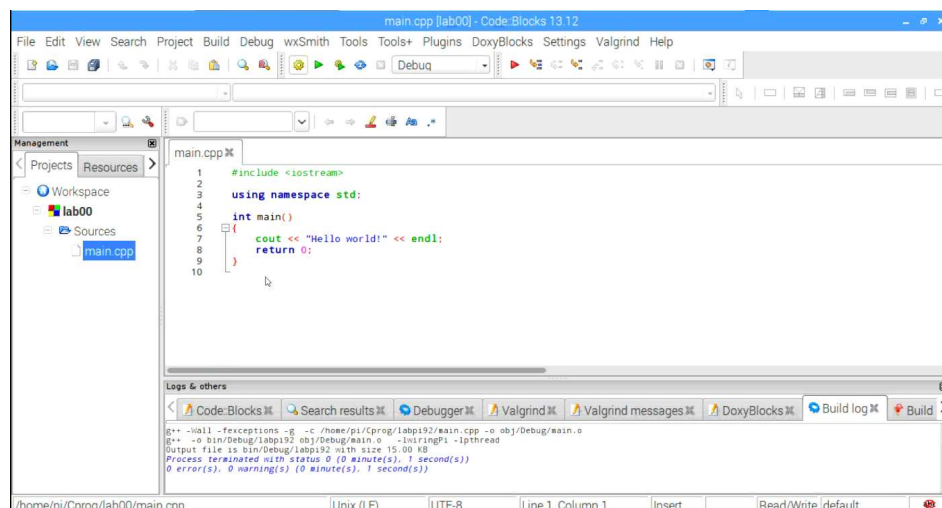


ตัวคอมไพเลอร์ที่เราจะใช้คือ GCC ซึ่งเป็นตัวเลือกตั้งต้นที่ปรากฏอยู่แล้ว ดังนั้นให้คลิก Finish เพื่อจบกระบวนการสร้างโปรเจกต์ใหม่

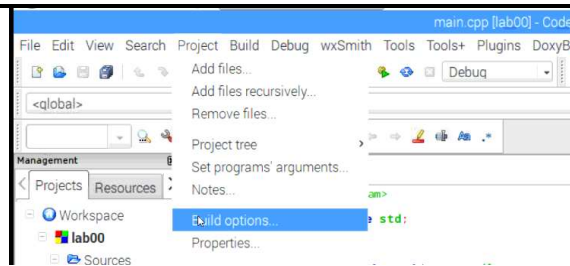


ที่หน้าต่าง/แท็บ Projects คลิกขยาย + ที่ชื่อโปรเจกต์ และเข้าไปยัง Sources และเข้าไปยังไฟล์ต้นฉบับตั้งต้นที่ชื่อ main.cpp (หากนักศึกษาต้องการเลือกไฟล์ต้นฉบับที่เตรียมไว้ล่วงหน้าก่อนแล้ว สามารถทำได้โดยคลิกเลือก main.cpp แล้วกดปุ่ม DELETE จากนั้นคลิกขวาที่ชื่อโปรเจกต์แล้วเลือก Add Files... เพื่อค้นหาไฟล์ที่ต้องการเพิ่มหรือแทนที่ได้ตามต้องการ

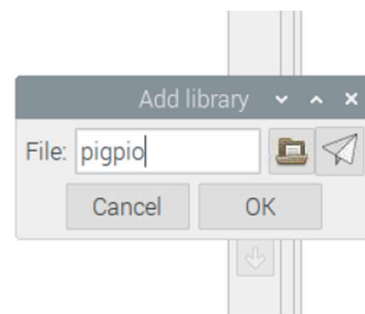
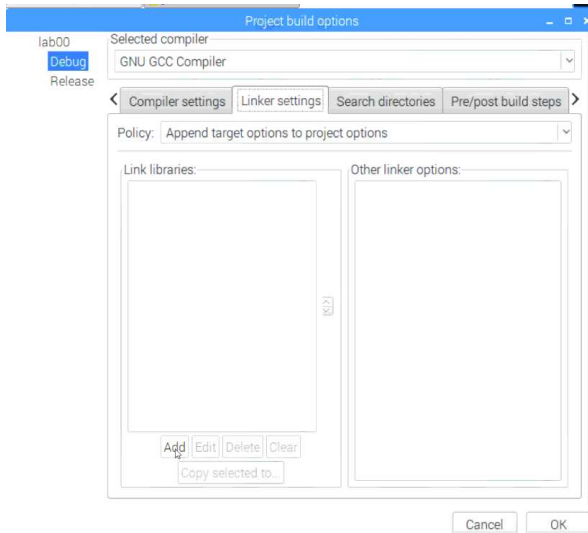
สังเกตว่า Code::Blocks เตรียมโปรแกรมตัวอย่างไว้ให้ด้วยเพื่อต้องการทดลองแปลโปรแกรม






ในชุดปฏิบัติการ raspberry pi นี้ นักศึกษาจำเป็นต้องใช้ไลบรารี pigpio สำหรับการติดต่อกับอุปกรณ์รอบข้าง และต้องการไลบรารี pthread ในการเขียนโปรแกรมแบบหลายเธรด ดังนั้นเราสามารถแทรกเพิ่มไลบรารีลงในโปรเจกต์ได้โดยการเลือกเมนู Project/Build Options...



จากนั้นเลือกแท็บ Linker settings และคลิก Add เพื่อเพิ่มไลบรารีที่ต้องการลงไป เมื่อเพิ่มจนครบที่ต้องการแล้วจากนั้นคลิก Ok



การแปลและรันโปรแกรมอย่างง่าย สามารถเลือกทำได้ในสามลักษณะ คลิก  เพื่อแปลอย่างเดียว คลิก  เพื่อรันโปรแกรมที่แปลไว้ก่อนหน้า หรือคลิก  เพื่อแปลและรัน

ปฏิบัติการ: เขียนโปรแกรมแรกบน pi

ให้นักศึกษาทดลองหีบเอาโปรแกรมที่เคยเรียนมาก่อนหน้านี้ หรือใช้โปรแกรมตัวอย่างที่ Code::Blocks สร้างให้ เพื่อลองแปลและรันดูว่าทำงานได้หรือไม่

ปฏิบัติการ: ทดลองใช้โปรแกรมตัวอย่างเพื่อแสดงข้อมูลพอร์ตทั้งหมดของ raspberry Pi ด้วย pigpio

ให้นักศึกษาใช้โปรแกรมตัวอย่างต่อไปนี้ โดยจะต้องรันโปรแกรมในระดับ root ตามที่ได้อธิบายไปแล้วก่อนหน้านี้ (การเซตที่เมนู Setting / Environment... จะเป็นการเซตถาวรให้กับทุกโปรแกรมที่จะเขียนขึ้นมาหลังจากนี้) และอย่าลืมเพิ่มไลบรารี pigpio ไว้ใน Project / Build options... / Linker settings ด้วย

```
#include <pigpio.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
char modeName[9][8] = {"IN", "OUT", "ALT5", "ALT4", "ALT0", "ALT1", "ALT2", "ALT3", " "};
char headerName[40][8] = {"3.3v", "5v", "2", "5v", "3", "GND", "4", "14", "GND", "15",
                          "17", "18", "27", "GND", "22", "23", "3.3v", "24", "10", "GND",
                          "9", "25", "11", "8", "GND", "7", "0", "1", "5", "GND",
                          "6", "12", "13", "GND", "19", "16", "26", "20", "GND", "21"};
```

```
int headerMap[40] = {-1, -1, 2, -1, 3, -1, 4, 14, -1, 15,
                    17, 18, 27, -1, 22, 23, -1, 24, 10, -1,
                    9, 25, 11, 8, -1, 7, 0, 1, 5, -1,
                    6, 12, 13, -1, 19, 16, 26, 20, -1, 21};
```

```

#define gpioPins 40

char altFunc[6][28][16]={{"SDA0","SCL0","SDA1","SCL1","GPCLK0","GPCLK1","GPCLK2","SPI0_CE1_N",
    "SPI0_CE0_N","SPI0_MISO","SPI0_MOSI","SPI0_SCLK","PWM0","PWM1",
    "TXD0","RXD0","FL0","FL1","PCM_CLK","PCM_FS",
    "PCM_DIN","PCM_DOUT","SD0_CLK","SD0_XMD","SD0_DAT0","SD0_DAT1","SD0_DA2","SD0_DA3"},
    {"SA5","SA4","SA3","SA2","SA1","SA0","SOE_N","SWE_N","SD0","SD1",
    "SD2","SD3","SD4","SD5","SD6","SD7","SD8","SD9","SD10","SD11",
    "SD12","SD13","SD14","SD15","SD16","SD17","TE0","TE1"},
    {"PCLK","DE","LCD_VSYNC","LCD_HSYNC","DPI_D0","DPI_D1","DPI_D2","DPI_D3",
    "DPI_D4","DPI_D5","DPI_D6","DPI_D7","DPI_D8","DPI_D9","DPI_D10",
    "DPI_D11","DPI_D12","DPI_D13","DPI_D14","DPI_D15",
    "DPI_D16","DPI_D17","DPI_D18","DPI_D19","DPI_D20","DPI_D21","DPI_D22","DPI_D23"},
    {"SPI3_CEO_N","SPI3_MISO","SPI3_MOSI","SPI3_SCLK","SPI4_CEO_N",
    "SPI4_MISO","SPI4_MOSI","SPI4_SCLK","_","_","_",
    "_","_","SPI5_CE0_N","SPI5_MISO","SPI5_MOSI","SPI5_SCLK",
    "CTS0","RTS0","SPI6_CE0_N","SPI6_MISO",
    "SPI6_MOSI","SPI6_SCLK","SD1_CLK","SD1_CMD","SD1_DAT0","SD1_DAT1","SD1_DAT2","SD1_DAT3"},
    {"TXD2","RXD2","CTS2","RTS2","TXD3","RXD3","CTS3","RTS3","TXD4","RXD4",
    "CTS4","RTS4","TXD5","RXD5","CTS5","RTS5","SPI1_CE2_N",
    "SPI1_CE1_N","SPI1_CE0_N","SPI1_MISO",
    "SPI1_MOSI","SPI1_SCLK","ARM_TRST","ARM_RTCX","ARM_TDO","ARM_TCK","ARM_TDI","ARM_TMS"},
    {"SDA6","SCL6","SDA3","SCL3","SDA3","SCL3","SDA4","SCL4","SDA4","SCL4",
    "SDA5","SCL5","SDA5","SCL5","TXD1","RXD1","CTS1","RTS1","PWM0","PWM1",
    "GPCLK0","GPCLK1","SDA6","SCL6","SPI3_CE1_N","SPI4_CE1_N","SPI5_CE1_N","SPI6_CE1_N"}};

char pullFunc[28][8]={"High","High","High","High","High","High","High","High","High","Low",
    "Low","Low","Low","Low","Low","Low","Low","Low","Low","Low",
    "Low","Low","Low","Low","Low","Low","Low","Low",};

char piModel[32][6]={"A","B","A+","B+","2B","-","CM1","-",
    "3B","Zero","CM3","-","ZeroW","3B+","3A+","-",
    "CM3+","4b",};

char *pinName(int pin);
char *pinMode(int pin);
int pinValue(int pin);

int main(){
    int hwRevision,h,i;

    if(gpioInitialise()<0){
        printf("usage: sudo ./pigpio\n");
        exit(-1);
    }

    hwRevision = gpioHardwareRevision()>>4;
    hwRevision %=32;

    printf("+-----+-----+-----+-----+ PI %-5s +-----+-----+-----+-----+\n",piModel[hwRevision]);
    printf("| BCM |   Name   | Mode | V | Board   | V | Mode | Name           | BCM |\n");
    printf("+-----+-----+-----+-----+-----+-----+-----+-----+-----+\n");
    for(i=0;i<40;i+=2){
//        odd pin
        h = i;
        if(headerMap[h]<0){
            printf("|           | %10s |           | |",pinName(h));
        }else{
            printf("| G%.2d | %10s | %4s | %.1d |",headerMap[h],pinName(h),pinMode(h),pinValue(h));
        }
        printf(" %2d || %2d ",i+1,i+2);
//        even pin
        h = i+1;
        if(headerMap[h]<0){
            printf("|           | %10s |           | \n",pinName(h));
        }else{
            printf("| %.1d | %4s | %10s | G%.2d | \n",pinValue(h),pinMode(h),pinName(h),headerMap[h]);
        }
    }
    printf("+-----+-----+-----+-----+-----+-----+-----+-----+-----+\n");
    printf("| BCM |   Name   | Mode | V | Board   | V | Mode | Name           | BCM |\n");

```



```

printf("+-----+-----+-----+---+ PI %-5s +---+-----+-----+-----+\n", piModel[hwRevision]);
gpioTerminate();
return 0;
}

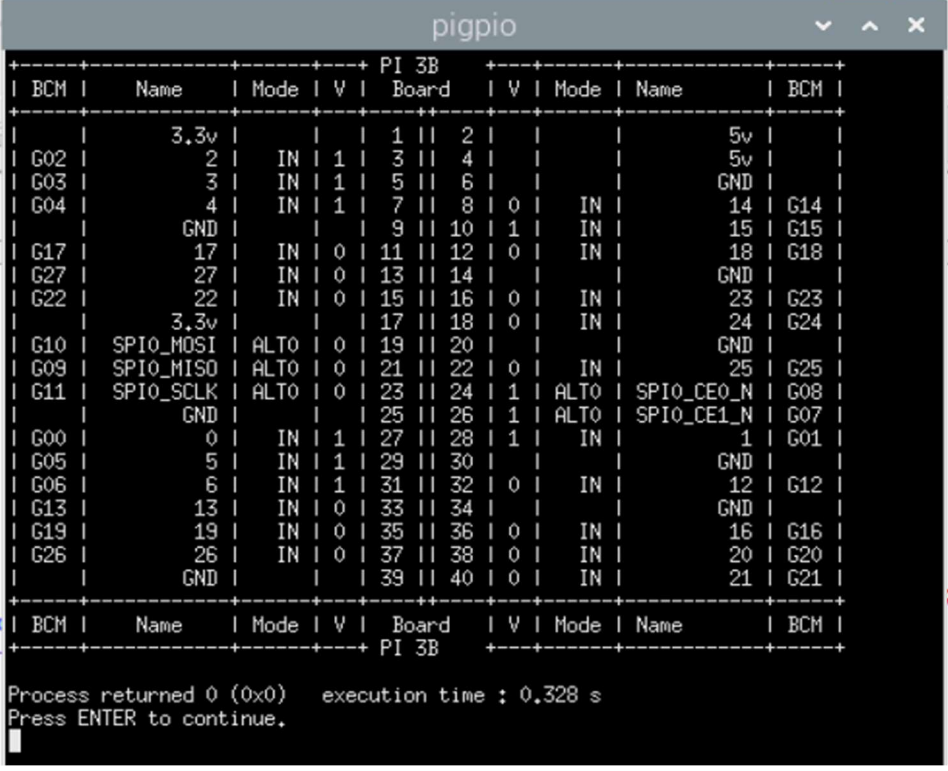
char *pinName(int pin){
    if(headerMap[pin]<0) return headerName[pin];
    switch(gpioGetMode(headerMap[pin])){
        case PI_ALT0: return altFunc[0][headerMap[pin]];
        case PI_ALT1: return altFunc[1][headerMap[pin]];
        case PI_ALT2: return altFunc[2][headerMap[pin]];
        case PI_ALT3: return altFunc[3][headerMap[pin]];
        case PI_ALT4: return altFunc[4][headerMap[pin]];
        case PI_ALT5: return altFunc[5][headerMap[pin]];
    }
    return headerName[pin];
}

char *pinMode(int pin){
    if(headerMap[pin]<0)
        return modeName[8];
    return modeName[gpioGetMode(headerMap[pin])];
}

int pinValue(int pin){
    if(headerMap[pin]<0)
        return -1;
    return gpioRead(headerMap[pin]);
}

```

จากโปรแกรมตัวอย่างข้างต้น (ในที่นี้กำหนดชื่อโปรเจ็คเป็น pigpio) เมื่อสั่งทำงานจะได้ผลในลักษณะดังนี้



The screenshot shows a terminal window titled 'pigpio' displaying a table of GPIO pin configurations. The table is divided into two sections by a dashed line. The top section lists pins 2 through 21, and the bottom section lists pins 22 through 40. Each row contains columns for BCM pin number, Name, Mode, V, Board, V, Mode, Name, and BCM pin number. The table shows various pin configurations, including 3.3v, 5v, GND, and specific functions like SPI0_MOSI, SPI0_MISO, SPI0_SCLK, SPI0_CE0_N, and SPI0_CE1_N. The table is followed by the text 'Process returned 0 (0x0) execution time : 0.328 s' and 'Press ENTER to continue.'

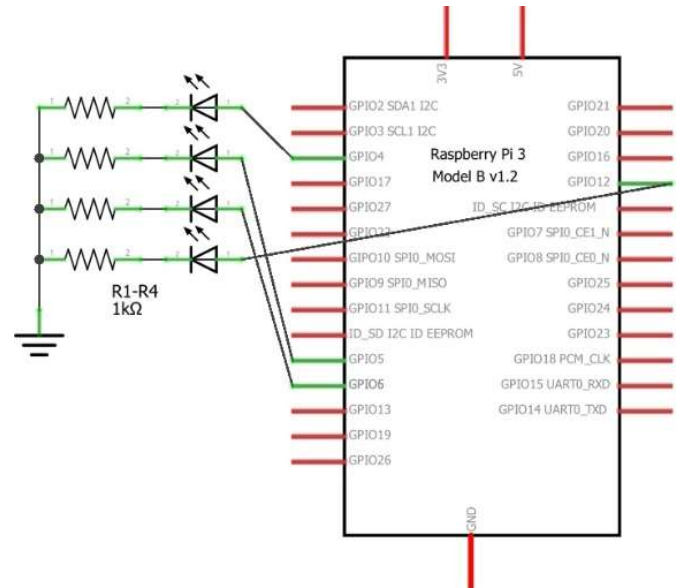
BCM	Name	Mode	V	Board	V	Mode	Name	BCM
	3.3v			1	2		5v	
G02	2	IN	1	3	4		5v	
G03	3	IN	1	5	6		GND	
G04	4	IN	1	7	8	0	IN	G14
	GND			9	10	1	IN	G15
G17	17	IN	0	11	12	0	IN	G18
G27	27	IN	0	13	14		GND	
G22	22	IN	0	15	16	0	IN	G23
	3.3v			17	18	0	IN	G24
G10	SPI0_MOSI	ALTO	0	19	20		GND	
G09	SPI0_MISO	ALTO	0	21	22	0	IN	G25
G11	SPI0_SCLK	ALTO	0	23	24	1	ALTO	SPI0_CE0_N
	GND			25	26	1	ALTO	SPI0_CE1_N
G00	0	IN	1	27	28	1	IN	G01
G05	5	IN	1	29	30		GND	
G06	6	IN	1	31	32	0	IN	G12
G13	13	IN	0	33	34		GND	
G19	19	IN	0	35	36	0	IN	G16
G26	26	IN	0	37	38	0	IN	G20
	GND			39	40	0	IN	G21

Process returned 0 (0x0) execution time : 0.328 s
Press ENTER to continue.

ปฏิบัติการ: เขียนโปรแกรมส่งข้อมูลออกทาง GPIO

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมแอดปเตอร์
- สายจัมป์จากขา GPIO ของบอร์ด
- โปรโตบอร์ด และสายจัมป์อีกตามต้องการ
- LED 4 ดวง เลือกสีตามต้องการ
- R ขนาด 100-1kohm จำนวน 4 ตัว (แนะนำให้ใช้ค่าประมาณ 1kohm)



ให้นักศึกษาต่อวงจร LED ง่ายๆ จากขา GPIO ตามที่นักศึกษาเลือก โดยอาศัยรูปเป็นแนวทาง โดยต่อ LED ไว้ทั้งหมด 4 ดวง จากนั้นทดลองการเขียนโปรแกรมเพื่อเปิดปิด LED ทั้งสี่ดวง ทดสอบดูว่าทำงานได้หรือไม่

** ขา GPIO4 เป็นขาสำหรับเชื่อมต่อกับ 1-wire หากนักศึกษาเปิดการใช้งาน 1-wire ให้เปลี่ยนขา GPIO4 ไปใช้ขาอื่นแทน (ตัวอย่างต่อไปนี้จะเป็นไปตามรูปร่างวงจรข้างต้น หากนักศึกษาต่อ LED เข้ากับ GPIO ตัวอื่น นักศึกษาจะต้องเปลี่ยนค่าตัวแปร `ledGPIO` ให้สอดคล้องกับพอร์ตที่นักศึกษาเลือกใช้ด้วย)

```
#include <pigpio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int ledGPIO[4] = {4,5,6,12};

int main() {
    int i;
    if(gpioInitialise() < 0) exit(-1);
    for(i=0;i<4;i++)
        gpioSetMode(ledGPIO[i],PI_OUTPUT);
    for(i=0;i<4;i++)
        gpioWrite(ledGPIO[i],1);
    usleep(5000000);
    for(i=0;i<4;i++)
        gpioWrite(ledGPIO[i],0);

    gpioTerminate();
    return 0;
}
```

หลังจากทดสอบรันโปรแกรมแล้ว ให้ทดลองใช้คำสั่ง `gpio readall` หรือรันโปรแกรมอ่านค่าพอร์ตที่ให้ไว้ก่อนหน้านี้ ในเทอร์มินัล เพื่อดูสถานะของพอร์ต GPIO ว่าเปลี่ยนไปอย่างไร และทดลองถอดข้อความสั่งในส่วน `for(i=0;i<4;i++) gpioWrite(ledGPIO[i],0);` ออกไป เพื่อให้ LED ยังคงติดค้างเมื่อจบโปรแกรม และใช้โปรแกรมอ่านค่าพอร์ตที่เราสร้างไว้อีกครั้ง เพื่อดูสถานะของพอร์ต GPIO ว่าเป็นอย่างไร

ปฏิบัติการ: ไฟวิ่งอย่างง่าย

จากปฏิบัติการก่อนหน้า คราวนี้ให้เขียนโปรแกรมง่ายๆ เพื่อสร้างเป็นไฟวิ่งโดยให้มีรูปแบบเป็นไปตามที่นักศึกษาต้องการ โดยมีความเร็วในการวิ่งเป็นไปตามที่นักศึกษากำหนดในเบื้องต้นก่อน

HINT การเขียนโปรแกรมไฟวิ่งที่สะดวกที่สุด ให้สร้างอะเรย์สองมิติเพื่อเก็บสถานะปิด-เปิดไฟในแต่ละจังหวะ โดยตัวมิติต่ำสุดเป็นอะเรย์ของไฟวิ่งในหนึ่งจังหวะ และมิติสูงขึ้นไป คือลำดับของจังหวะเช่น

`int runningLED[4][4]={1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1};` โดยในที่นี้มีหลอด LED ทั้งหมดสี่ดวง และไฟวิ่งทั้งหมด 4 จังหวะ (ในที่นี้จะวิ่งกวาดจากดวงแรกไปยังดวงสุดท้าย)

`int ledGPIO[4]={ก,ข,ค,ง};` เลขสี่ตัวในนี้คือเลขประจำพอร์ตของ GPIO (ตามตัวเลข wPi ที่ปรากฏใน `gpio readall`)

ส่วนการทำงานของโปรแกรม จะใช้ `while` loop (หรืออื่นใดที่ทดแทนกัน) เพื่อวนรอบการแสดงผล โดยเพิ่มค่าลำดับขึ้นไปเรื่อยๆ ทีละหนึ่งในแต่ละรอบ เมื่อรันไปจนถึงตัวสุดท้าย ก็จะวนกลับมาเริ่มที่ลำดับ 0 ใหม่ วนเวียนนี้เรื่อยไป

```
while(1){
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
            gpioWrite(ledGPIO[j],runningLED[i][j]);
            usleep(หน่วยเวลาให้เล็กน้อย);
        }
    }
}
```

ภายใน `while` loop จะประกอบไปด้วยการเขียนไปยัง GPIO ทั้งสี่ตัวด้วยสถานะที่กำหนดไว้ในอะเรย์ จากนั้นก็หยุดรอด้วย `usleep()`

การเขียนโปรแกรมแบบหลายเธรดในลินุกซ์

ระบบปฏิบัติการลินุกซ์รองรับการเขียนโปรแกรมแบบหลายเธรดโดยอาศัยไลบรารี POSIX thread หรือที่เรียกสั้นๆ ว่า pthread โดยเมื่อโปรแกรมทำงานในเบื้องต้น จะสร้างโปรเซสหลักขึ้นมาหนึ่งตัวซึ่งมีสถานะเป็นเธรดหลัก จากนั้นเราจะเรียกใช้ฟังก์ชัน `pthread_create()` เพื่อสร้างเธรดใหม่ โดยเราจะผ่านค่าเธรดฟังก์ชันเข้าไปเป็นอาร์กิวเมนต์ เพื่อใช้ในการทำงานของเธรดที่สร้างขึ้น เธรดแต่ละตัวที่ถูกสร้างขึ้น สามารถสร้างเธรดแยกออกไปได้อีก และเมื่อเราต้องการให้เธรดที่สั่งทำงานนั้นจบลง เราจะออกจากเธรดฟังก์ชันด้วยฟังก์ชัน `pthread_exit()` ส่วนเธรดหลัก (หรือเธรดพ่อแม่ที่สร้างเธรดลูกออกไป) จะใช้ฟังก์ชัน `pthread_join()` เพื่อรอให้เธรดลูกจบการทำงาน ก่อนที่เธรดหลักจะทำงานต่อไป การเขียนโปรแกรมแบบหลายเธรด จำเป็นต้องให้เธรดหลักรอเธรดลูกให้จบการทำงานเสียก่อน เธรดหลัก(หรือโปรเซสหลักของโปรแกรม) จึงจะจบการทำงานได้ ไมเช่นนั้น อาจเกิดปัญหาเกิดขึ้นกับการคืนทรัพยากรที่เธรดขอใช้งานเพิ่มเติมให้กับระบบ

ลองพิจารณาโปรแกรมตัวอย่างแรกต่อไปนี้

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

int c[4][4];
int a[4][4]={1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int b[4][4]={10,-10,10,-10},{-10,10,-10,10},{10,-10,10,-10},{-10,10,-10,10}};

void showresult();
void *threadFunction(void *selector);

int main(void){
```

```

pthread_t tid1,tid2;           // Thread ID
pthread_attr_t attr1,attr2;    // Thread attributes
int section1=0,section2=1;

pthread_attr_init(&attr1); // Get default attributes
pthread_attr_init(&attr2); // Get default attributes

// Create 2 threads
pthread_create(&tid1,&attr1,threadFunction,(void *)&section1);
pthread_create(&tid2,&attr2,threadFunction,(void *)&section2);

// Wait until all threads finish
pthread_join(tid1,NULL);
pthread_join(tid2,NULL);

pthread_attr_destroy(&attr1);
pthread_attr_destroy(&attr2);

showresult();
return 0;
}

void showresult(){
    int i,j;

    for(j=0;j<4;j++){
        printf("|");
        for(i=0;i<4;i++){
            printf("%3d ",c[j][i]);
        }
        printf("|\\n");
    }
}

void *threadFunction(void *selector){
    int sel=((int *)selector);
    int start,stop;
    int i,j;

    start=sel*2;
    stop =start+2;

    for(j=start;j<stop;j++){
        for(i=0;i<4;i++){
            c[j][i]=a[j][i]+b[j][i];
            // To Show how thread runs, this will slow thing down...
            sleep(1);
            printf("From thread%d i=%d j=%d\\n",sel,i,j);
            fflush(stdout); // Send text to display immediately
        }
    }
    pthread_exit(NULL);
}

```

มีประเด็นที่น่าสนใจจากโปรแกรมตัวอย่างดังต่อไปนี้

- เธรดฟังก์ชัน จำต้องมีการนิยามโปรโตไทป์และหัวฟังก์ชันในรูป

```
void *ชื่อเธรดฟังก์ชัน(void *ชื่ออาร์กิวเมนต์)
```

เธรดฟังก์ชันจำเป็นต้องมีอาร์กิวเมนต์หนึ่งตัว สำหรับค่าอ้างอิงไปยังตัวแปร หรือตัวแปรโครงสร้างที่ใช้สำหรับรับค่าที่ผ่านมาจากเธรดหลัก (ส่งผ่านอาร์กิวเมนต์ของ pthread_create()) และจะต้องกำหนดไว้เสมอ ไม่ว่าจะได้ใช้งานหรือไม่

- เธรดฟังก์ชัน จะต้องจบด้วยการเรียกใช้ฟังก์ชัน pthread_exit() ซึ่งมีโปรโตไทป์ดังรูป

```
void pthread_exit(void *ret);
```

อาร์กิวเมนต์รับค่าอ้างอิงไปยังตัวแปรที่ใช้ส่งค่ากลับคืนไปยังเธรดพ่อแม่ จากตัวอย่างข้างต้นเรียกใช้งานอย่างข้อความสั่ง pthread_exit(NULL); ซึ่งหมายความว่าไม่มีการส่งค่าใดๆ กลับไปยังเธรดพ่อแม่

- การสร้างเธรด ใช้ฟังก์ชัน pthread_create() ซึ่งมีโปรโตไทป์ดังนี้

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                  void *(*start_routine) (void *), void *arg);
```

และเราต้องนิยามตัวแปรสองตัวเพื่อใช้งานคือ Thread ID และ Thread attribute โดย Thread ID มีแบบชนิดเป็น pthread_t และตัวแอตทริบิวต์มีแบบชนิดเป็น pthread_attr_t และเราต้องกำหนดสภาพเริ่มต้นให้กับตัวแปรแอตทริบิวต์ด้วยฟังก์ชัน pthread_attr_init() เสียก่อน ฟังก์ชัน pthread_attr_init() มีโปรโตไทป์ดังนี้

```
int pthread_attr_init(pthread_attr_t *attr);
```

เมื่อเธรดถูกจบลงแล้ว ควรใช้ฟังก์ชัน pthread_attr_destroy() เพื่อคืนทรัพยากรในการจัดการเธรดให้กับระบบ

```
int pthread_attr_destroy(pthread_attr_t *attr);
```

- เมื่อแตกเธรดถูกไปทำงานแล้ว เธรดหลักหรือเธรดพ่อแม่สามารถทำงานอื่นต่อไปพร้อมกันได้ โดยตัวแปรส่วนกลาง (global variable) ต่างๆ และ I/O ต่างๆ ที่เธรดหลักเปิดไว้ให้ จะสามารถใช้ได้ในระหว่างเธรดลูกด้วยกัน ดังนั้นเราจึงสามารถผ่านข้อมูลต่างๆ ระหว่างเธรดโดยอาศัยการนิยามตัวแปรส่วนกลางไว้

ฉะนั้น เธรดหลัก ควรจะต้องรอให้เธรดลูกทำงานจบเสียก่อน ตนเองจึงจะสามารถจบการทำงานได้ (เพื่อไม่ให้เกิดการคืนทรัพยากรส่วนกลางก่อนที่เธรดลูกจบการทำงาน ซึ่งจะทำให้เกิดความผิดพลาดในการทำงาน) การรอเธรดลูกให้จบการทำงานใช้ฟังก์ชัน pthread_join() ซึ่งมีโปรโตไทป์ดังรูป

```
int pthread_join(pthread_t thread, void **retval);
```

อาร์กิวเมนต์แรกของ pthread_join() รับค่าเธรดไอดี (เป็นค่าที่อยู่ในตัวแปรที่ใช้ส่งค่าอ้างอิงไปให้อาร์กิวเมนต์แรกของ pthread_create()) ส่วนอาร์กิวเมนต์ที่สองเป็นค่าอ้างอิงที่ชี้ไปยังค่าอ้างอิงของค่าที่จะรับค่ากลับ

ลักษณะการใช้งานของการส่งค่ากลับเป็นดังเช่น สมมติว่ามีการนิยามตัวแปร int ret1; เพื่อใช้ส่งค่ากลับคืนจากเธรด การใช้งาน pthread_exit() จะเป็นดังรูป

```
ret1 = 0;
pthread_exit((void *)&ret1);
```

และการใช้งาน pthread_join() จะเป็นดังรูป

```
int *retvaluel;
pthread_join(pid1, (void **)&retvaluel);
```

จากตัวอย่าง เรานิยามตัวชี้ retvaluel เพื่อใช้รับค่าอ้างอิงของตัวแปรที่จะส่งค่ากลับ และเราส่งค่าอ้างอิงของตัวชี้ retvaluel ไปยังฟังก์ชัน pthread_join() เพื่อให้ฟังก์ชันอาศัยค่าอ้างอิงของตัวชี้ที่ได้ เพื่อใช้เก็บค่าอ้างอิงของตัวแปรที่จะส่งค่ากลับจากเธรด (ในตัวอย่างข้างต้นนิยามไว้เป็นตัวแปรแบบชนิด int ซึ่งจะเห็นว่าตัวชี้ปลายทางที่จะรับค่าต้องมีแบบชนิดเดียวกัน การส่งค่ากลับในลักษณะเช่นนี้อาจดูยุ่งยาก แต่ข้อดีก็คือ เราสามารถนิยามตัวแปรเฉพาะที่ไว้ในเธรดฟังก์ชัน เพื่อใช้ในการส่งค่ากลับคืนมายังเธรดพ่อแม่ได้

- จากโปรแกรมตัวอย่าง จะเห็นถึงการนิยามตัวแปร section1 และ section2 เพื่อเก็บค่าที่จะใช้ส่งให้กับเรดฟังกซ์ โดยเราจะส่งค่าอ้างอิงของตัวแปรให้กับฟังก์ชัน pthread_create() ผ่านอาร์กิวเมนต์ตัวสุดท้าย ซึ่งในที่นี้เราต้องแปลงแบบชนิดค่าอ้างอิงเป็น (void *) เพื่อให้ตรงกับแบบชนิดของอาร์กิวเมนต์ที่ต้องการ ส่วนภายในเรดฟังกซ์ เราจะแปลงแบบชนิดค่าอ้างอิงกลับเป็นแบบชนิดเดิมของต้นทาง (ในที่นี้เป็น int *) แล้วจึงใช้ตัวดำเนินการโดยอ้อม (*) เพื่อเข้าถึงค่าในตัวแปรต้นทางต่อไป

โดยการทำให้ลักษณะเช่นนี้ จึงทำให้เราสามารถส่งผ่านค่าต่างๆ ไปยังเรดฟังกซ์ได้ อย่างเช่นในโปรแกรมตัวอย่างข้างต้นนี้ เราใช้ตัวเรดฟังกซ์ร่วมกันระหว่างสองเรด (หมายความว่าทั้งสองเรดใช้โค้ดตัวเดียวกัน) แต่เราแยกแยะข้อมูลที่จะใช้งาน โดยอาศัยค่าที่ผ่านเข้ามาดังกล่าว

ในกรณีที่ไม่ต้องส่งผ่านค่าจากเรดหลักไปยังเรดฟังกซ์ (เช่นการสร้างเรดหลายเรดซึ่งแต่ละเรดมีโค้ดฟังกซ์เป็นของตนเอง) เวลาเรียกใช้ฟังก์ชัน pthread_create() เราจะผ่านค่า NULL ให้เป็นอาร์กิวเมนต์สุดท้าย ตัวอย่างเช่น เราสร้างเรดฟังกซ์ชื่อ threadFunction1() และ threadFunction2() ซึ่งรองรับการทำงานของแต่ละเรดแยกจากกัน และไม่มีการผ่านค่าไปยังเรดฟังกซ์ ข้อความสั่งสร้างเรดทั้งสองจะเป็นดังนี้

```
pthread_create(&tid1,&attr1,threadFunction1,NULL);  
pthread_create(&tid2,&attr2,threadFunction2,NULL);
```

- จะต้องมีการเรียกใช้ไลบรารี pthread ด้วย (ในลักษณะเดียวกันกับการใช้ pigpio ก่อนหน้านี้)

ปฏิบัติการ: เขียนโปรแกรมแบบหลายเรดเพื่อแบ่งหน้าที่การทำงาน

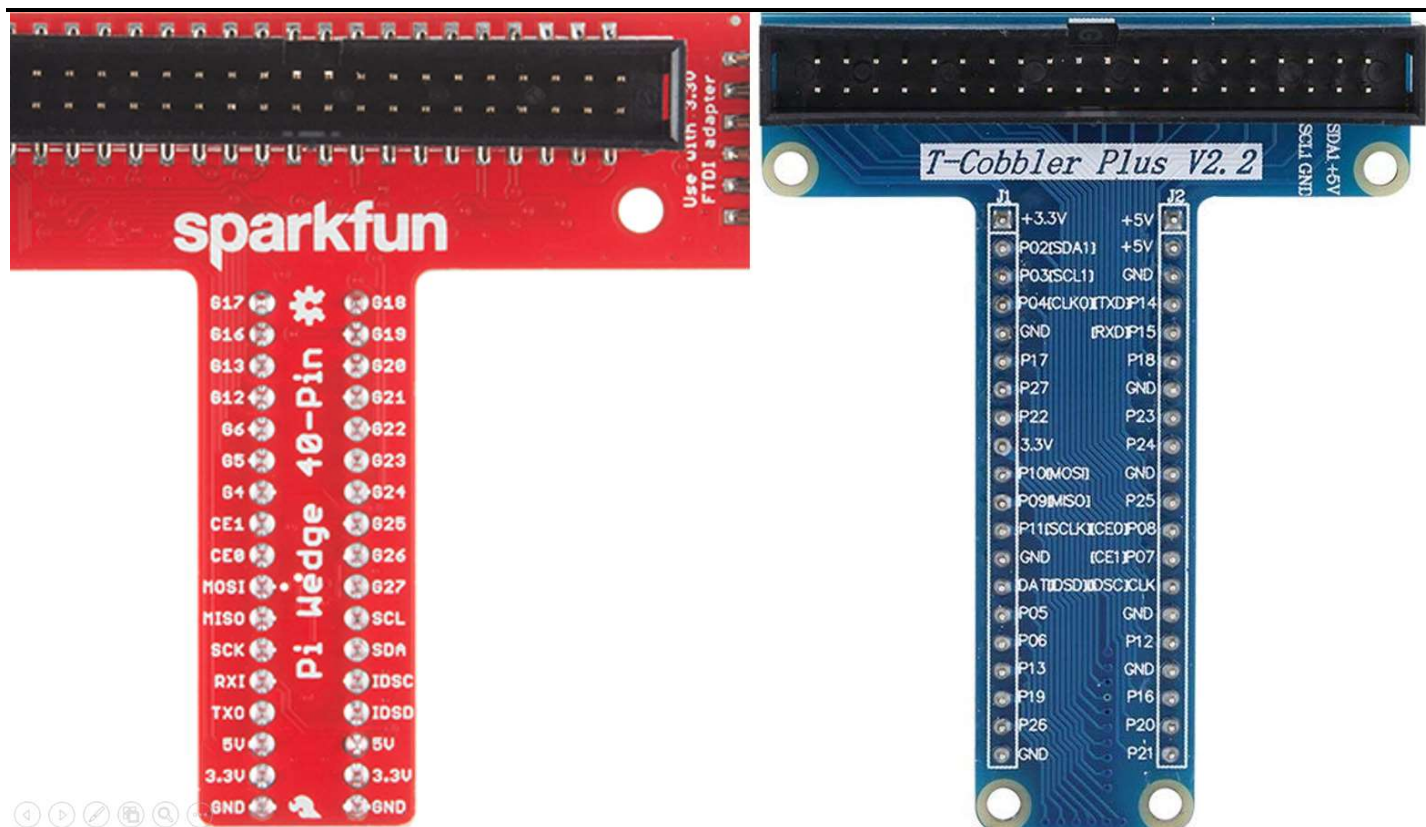
จากโปรแกรมไฟวิ่งก่อนหน้า คราวนี้ให้นักศึกษายกชุดคำสั่งเฉพาะส่วนที่ทำงานด้านไฟวิ่ง ไปใส่ไว้ในเรดหนึ่งแยกออกมาจากโปรแกรมหลัก และปรับโปรแกรมให้ไปสร้างเรดดังกล่าวเมื่อโปรแกรมเริ่มทำงานและเช็คสถานะเริ่มต้นให้ pigpio เรียบร้อยแล้ว

จากนั้น ให้นักศึกษาเขียนเรดแยกออกมาอีกเรดหนึ่ง เรดนี้จะทำหน้าที่รอรับการกดคีย์บอร์ดด้วย scanf() ตามธรรมดา โดยให้นักศึกษารับค่าความเร็วของไฟวิ่งในค่าจาก 1 ถึง 10 (1 จะวิ่งเร็วสุด ส่วน 10 จะวิ่งช้าสุด) นักศึกษาจะต้องสร้างตัวแปรส่วนกลางที่ใช้ร่วมกันระหว่างเรดรับค่าจากคีย์บอร์ดกับเรดที่ควบคุมไฟวิ่ง โดยเรดที่คุมไฟวิ่ง จะอ่านค่าความเร็วนี้และนำไปใช้ประกอบการคำนวณเวลาเพื่อใช้กับ usleep() (ให้นักศึกษาใช้ตัวคูณที่ทำให้ผลไฟวิ่งออกมาด้วยความเร็วที่เหมาะสมตามชอบ) ส่วนเรดรับค่าจากคีย์บอร์ดก็จะทำค่าที่อ่านได้ มาเซตไว้ในตัวแปรส่วนกลางตัวนี้

ผลการทำงานของโปรแกรม จะเห็นว่าตัวไฟวิ่งจะทำงานไปตลอดเวลา และเมื่อนักศึกษารอกค่าใหม่ลงไป ไฟวิ่งก็จะเปลี่ยนความเร็วให้ช้าลงหรือเร็วขึ้นในทันทีโดยไม่มีการหยุดรอแต่ประการใด

เพื่อความสะดวก นักศึกษาอาจจะเขียนโปรแกรมดักคำว่า หากผู้ใช้พิมพ์ค่านอกเหนือไปจาก 1 ถึง 10 ให้โปรแกรมหยุดการทำงาน (โดยการตรวจสอบค่าในตัวแปรที่ใช้รับค่าจากคีย์บอร์ดดังกล่าว และสั่ง break; เพื่อให้หลุดจากวนรอบทั้งใน loop while ของเรดแสดงไฟวิ่ง และเรดรอรับค่า เพื่อให้โปรแกรมจบการทำงานอย่างเรียบร้อย)

หมายเหตุ ในการเขียนโปรแกรมแบบหลายเรด เราอาจพบว่าบ่อยครั้งที่ printf() จะไม่ยอมแสดงข้อความในทันที เนื่องจากกลไกการจัดการสตรีนส่วนมาก จะรอให้สตรีนเต็มในระดับหนึ่งก่อนที่จะส่งข้อมูลออกไปจริงๆ ในที่นี้แนะนำให้ศึกษาใช้ fflush(stdout); หลังการใช้ printf() เพื่อดันให้ข้อความแสดงในทันทีโดยไม่ต้องรอให้สตรีน stdout เต็มเสียก่อน



Raspberry Pi breakout board สำหรับใช้เสียบลงบน breadboard เพื่อความสะดวกในการทดลอง

ขั้วด้านขวามือบนของ breakout board เป็นขั้วฝั่งหมายเลข 1

