ปฏิบัติการบน RaspberryPi ครั้งที่ 5: การควบคุม PWM และใช้งานร่วมกับเซอร์โว

การสร้างสัญญาณ hardware PWM อาศัยไลบรารี pigpio

Raspberry Pi มีวงจร PWM อยู่ 2 ตัวด้วยกัน โดยตัวแรก PWM0 สามารถกำหนดให้สัญญาณออกได้ที่ขา GPIO18 (pin 12) หรือ GPIO12 (pin 32) ส่วน PWM1 สามารถกำหนดให้สัญญาณออกได้ที่ขา GPIO13 (pin 33) หรือ GPIO19 (pin35)

ไลบรารี pigpio มีฟังก์ชันไว้ให้ทำงานที่เกี่ยวข้องกับฮาร์ดแวร์ PWM เพียงฟังก์ชันเดียว โดยใช้เพื่อสั่งการเปิดใช้งาน Hardware PWM และการกำหนดค่า duty cycle ไปพร้อมกันเลย ดังนี้

```
int gpioHardwarePWM(unsigned gpio, unsigned PWMfreq, unsigned PWMduty);
```

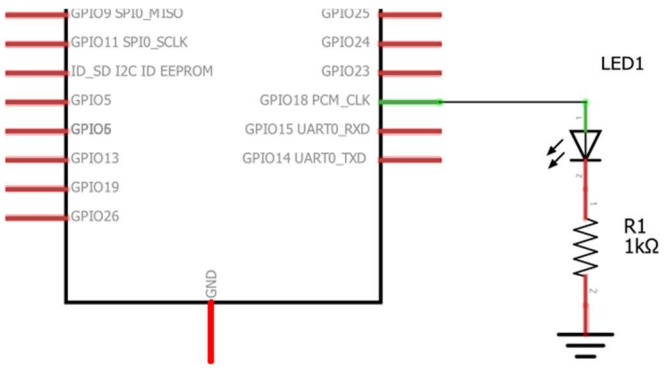
gpio	ขาที่จะใช้เพื่อปล่อยสัญญาณ PWM ออกไป (ใช้ได้เฉพาะขา GPIO18/12/13/19)	
PWMfreq	ความถี่ของสัญญาณ 0 (ปิดการทำงาน) หรือ 1-125000000 (125Mz) ทั้งนี้ค่าความถี่สูงมากๆ อาจจะไม่ทำงาน (ตามที่ไลบรารี pigpio แจ้งไว้ว่าหากสูงเกินกว่า 30MHz อาจจะไม่ทำงาน)	
PWMduty	ค่า duty cycle (สัดส่วนของช่วงสัญญาณพัลส์ 1 ต่อช่วงสถานะที่เป็น 0) หากใช้ 0 จะปิดการทำงาน ช่วงของค่าที่เป็นไปได้อยู่ในช่วง 0 (0% duty cycle) ถึง 1000000 (100% duty cycle)	
ค่ากลับคืน	OK	ทำงานได้อย่างถูกต้อง
	PI_BAD_GPIO	ขา GPIO ดังกล่าวไม่การทำงาน
	PI_NOT_HPWM_GPIO	ขา GPIO ดังกล่าวไม่รองรับฮาร์ดแวร์ GPIO
	PI_BAD_HPWM_DUTY	ช่วงค่า duty cycle ที่ให้ไปอยู่นอกขอบเขตที่กำหนด
	PI_HPWM_ILLEGAL	ค่าที่เซ็ตมีความผิดพลาด

ปฏิบัติการ: การสร้างสัญญาณ hardware PWM เพื่อควบคุมความสว่างของ LED

ในปฏิบัติการแรกนี้ เรานำฮาร์ดแวร์ PWM มาเพื่อใช้ควบคุมความสว่างของ LED โดยเราจะกำหนดค่าความถี่ของพัลส์เป็น 50 Hz (ครั้งต่อวินาที) และจะทดลองกวาดค่า duty cycle ไปตั้งแต่ 0 ถึง 100 เปอร์เซ็นต์

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด
- โปรโตบอร์ด และสายจัมพ์อีกตามต้องการ
- LED หนึ่งดวง และ R 1kohm 1 ตัว



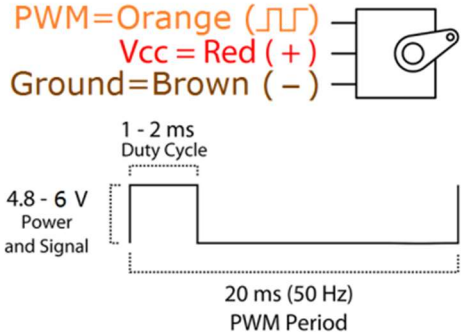
ตัวอย่างโปรแกรม

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

int PWM_pin=18;

void gpio_stop(int sig);

int main(){
    int i;

    printf("LED PWM (0%%-100%% duty cycle)\n");

    if(gpioInitialise() < 0){
         return -1;
    }
    signal(SIGINT,gpio_stop);

    while(1){
        for(i=0;i<100;i++){
                gpioHardwarePWM(PWM_pin,50,i*10000);
                usleep(10000);
        }
        for(i=99;i>0;i--){
                gpioHardwarePWM(PWM_pin,50,i*10000);
                usleep(10000);
        }
    }
    return 0;
}

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    gpioTerminate();
    exit(0);
}
```

ปฏิบัติการ: การสร้างสัญญาณ hardware PWM เพื่อควบคุมมุมของแกนเซอร์โว

กลไกการทำงานภายในของเซอร์โวที่เรานำมาทดลองนั้น ควบคุมมุมกวาดโดยสัญญาณ PWM ที่เราป้อนเข้าไป ภายในตัวเซอร์โวจะมีวงจรคอยวัดมุม และจะทำการขับตัวมอเตอร์เซอร์โวให้บิดไปตามมุมที่วงจรกำหนด ตัวเซอร์โวสามารถบิดแกนไปยังมุมที่ต้องการโดยอาศัยพัลส์เพียงลูกเดียวก็พอ แต่ถ้าเราส่งพัลส์ไปอย่างต่อเนื่อง จะมีประโยชน์ในกรณีที่หากแรงกระทำต่อแขนที่มีมากพอที่จะบิดแกนเซอร์โวให้เปลี่ยนไปจากมุมที่กำหนด พัลส์ที่ตามมา จะช่วยสั่งให้เซอร์โวคอยบิดแกนกลับไปยังมุมที่กำหนดตลอดเวลาได้

ตัวอย่างเซอร์โวที่ใช้ในปฏิบัติการนี้เป็นรุ่น 9g ซึ่งมีสเป็คในการควบคุมดังนี้



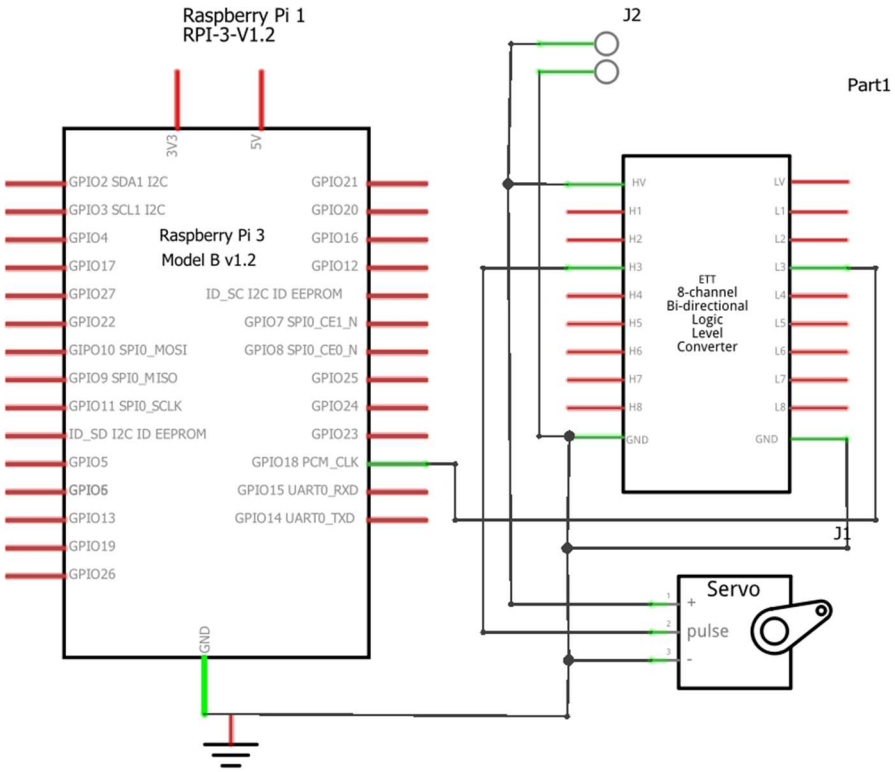
จากรูป เราจะเห็นว่าพัลส์ที่จะต้องส่งออกไปเพื่อควบคุมเซอร์โวนั้น จะมีช่วงสัญญาณ 1 ซึ่งจะมีคาบเวลาอยู่ในช่วง 1 ถึง 2 มิลลิวินาที ส่วนคาบเวลารวมของพัลส์ทั้งช่วงบวกและลบ จะต้องไม่ต่ำกว่า 20 มิลลิวินาที (ในความเป็นจริง เราสามารถทิ้งช่วงสัญญาณ 0 นี้นานกว่านี้ได้) โดยค่ามุม

ต่ำสุด (แกนเซอร์โวกวาดไปทางซ้ายสุด) จะมีค่าสัญญาณช่วง 1 ที่ประมาณ 1 มิลลิวินาที และค่ามุมสูงสุด (แกนเซอร์โวกวาดไปทางขวาสุด) จะมีค่าสัญญาณช่วง 1 ที่ประมาณ 2 มิลลิวินาที ในความเป็นจริง เซอร์โวแต่ละรุ่น อาจจะมีค่าช่วงเวลาที่กล่าวมานี้แตกต่างกันออกไป ดังนั้นนักศึกษาจึงควรเริ่มต้นด้วยการเขียนโปรแกรมที่สามารถรองรับการสร้างพัลส์ที่มีคาบเวลาที่กว้างกว่านี้ แล้วอาศัยการทดลองปรับแต่งเพื่อหาตำแหน่งซ้ายสุดและขวาสุด (ตำแหน่ง -90 องศา และ +90 องศา) และใช้ค่าดังกล่าวในการทำงานจริงต่อไป

จากข้อกำหนดข้างต้นของเซอร์โวที่นำมาใช้งานนั้น จะทำให้ได้ค่า duty cycle อยู่ในช่วง 5% (1ms/20ms) ถึง 10% (2ms/20ms) เมื่อคำนวณกับค่าที่ต้องป้อนให้กับอาร์กิวเมนต์ของ gpioHardwarePWM ก็จะได้ค่าในช่วง 50000 ถึง 100000 เมื่อปรับวงจรและโปรแกรมตัวอย่างจากข้างต้น ก็จะได้ดังนี้

<u>อุปกรณ์ที่ต้องการ</u>

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด
- โปรโตบอร์ด และสายจัมพ์อีกตามต้องการ
- Logic Shifter 1 ตัว
- ไมโครเซอร์โวรุ่น 9g 1 ตัว



<u>หมายเหตุ</u> ตัวเซอร์โวเองอาจจะมีความคาดเคลื่อนไปจากค่าที่คำนวณได้ ดังนั้นในทางปฏิบัติ ควรจะทดลองหาค่าช่วงของ pwmClock ที่ครอบคลุมครบช่วงที่เซอร์โวรับได้ แต่จะต้องไม่ส่งผลทำให้แกนเซอร์โวขยับเกินขอบที่กำหนด (ในตัวอย่างโปรแกรมนี้ นักศึกษาจะพบว่าแขนของเซอร์โวจะไม่กวาดไปครบ 180องศา ตามสเป็กของเซอร์โวที่ให้ไว้)

หากพบว่าเซอร์โวมีอาการเสียงดังคราง ตลอดเวลาคล้ายกับเกียร์ขบภายใน ให้ปลดสัญญาณ PWM ออกโดยทันทีเพื่อป้องกันเกียร์รูด (เซอร์โวเสียหาย)

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

int PWM_pin=18;

void gpio_stop(int sig);

int main(){
    int i;

    printf("Servo PWM (5%%-10%% duty cycle)\n");

    if(gpioInitialise() < 0){
        return -1;
    }
    signal(SIGINT,gpio_stop);

    while(1){
        for(i=50;i<100;i++){
                gpioHardwarePWM(PWM_pin,50,i*1000);
```

ระวัง!!!!

เนื่องจากการทดลองก่อนหน้า เป็นการสร้างสัญญาณ PWM ด้วยวงจรภายใน Pi ซึ่งยังคงทำงานต่อไปแม้ว่าหยุดโปรแกรมแล้ว และเนื่องจากค่า duty cycle ของการทดลองครั้งที่แล้วจะอยู่นอกย่านของการทดลองนี้ ดังนั้น **นักศึกษาอย่าเพิ่งต่อสาย PWM จากเซอร์โว ให้รันโปรแกรมปฏิบัติการนี้เสียก่อน จึงค่อยต่อสาย PWM จากเซอร์โวเข้า Logic shifter**

```
                usleep(20000);
        }
        for(i=99;i>50;i--){
                gpioHardwarePWM(PWM_pin,50,i*1000);
                usleep(20000);
        }
    }
    return 0;
}

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    gpioTerminate();
    exit(0);
}
```

การสร้างสัญญาณ Hardware timed PWM อาศัยไลบรารี pigpio

ไลบรารี pigpio มีกลไกการสร้างสัญญาณ PWM โดยเปลี่ยนระดับสัญญาณขา GPIO ไปมาเพื่อเลียนแบบการทำงานของฮาร์ดแวร์ PWM แต่มีการควบคุมคาบเวลาโดยใช้กลไกทางฮาร์ดแวร์อื่นๆ ที่มีบนชิพ SoC ของ Raspberry Pi ทำให้เราสามารถสร้างสัญญาณ PWM ออกจากขา GPIO ได้จากขาอื่นๆ ที่ไม่รองรับฮาร์ดแวร์ PWM โดยไลบรารี pigpio รองรับการทำ Hardware timed PWM ได้ในขา GPIO0-GPIO31

ฟังก์ชันจัดการ Hardware timed PWM มีดังต่อไปนี้

```
int gpioPWM(unsigned user_gpio, unsigned dutycycle);
```

สั่งเริ่มการทำงาน PWM ที่ขา gpio ที่กำหนด

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)	
dutycycle	ช่วงของ duty cycle จาก 0เปอร์เซ็นต์ถึง 100 เปอร์เซ็นต์ โดยค่าสูงสุดกำหนดโดย gpioSetPWMrange() ค่าเริ่มต้นมีค่าเท่ากับ 255	
ค่ากลับคืน	0	ทำงานได้ปกติ
	PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน
	PI_BAD_DUTYCYCLE	ค่า duty cycle นอกขอบเขตที่รองรับ

```
int gpioGetPWMdutycycle(unsigned user_gpio)
```

สั่งเริ่มการทำงาน PWM ที่ขา gpio ที่กำหนด

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)

ค่ากลับคืน กรณีปกติจะเป็นค่าที่เซ็ตไว้โดย gpioPWM() (ในกรณีที่มีการใช้งาน Hardware Clock ค่ากลับคืนจะมีค่า 500000 และในกรณีที่อ่านค่าขาที่เซ็ตไว้โดย HardwarePWM() จะได้เป็นค่าสัดส่วนจากค่าสูงสุด 1000000)

กรณีทำงานผิดพลาด

PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน
PI_NOT_PWM_GPIO	ขา GPIO ดังกล่าวไม่ได้เซ็ตไว้ในโหมดการทำงาน PWM

```
int gpioSetPWMrange(unsigned user_gpio, unsigned range)
```

กำหนดช่วงค่าที่ใช้เป็นค่า duty cycle สูงสุดของ PWM

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)	
range	ค่าที่แทนค่า duty cycle ที่ 100เปอร์เซ็นต์ ซึ่งมีค่าได้ตั้งแต่ 25 ถึง 40000	
ค่ากลับคืน	0	ทำงานได้ปกติ
	PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน
	PI_BAD_DUTYCYCLE	ค่า duty cycle นอกขอบเขตที่รองรับ

```
int gpioGetPWMrange(unsigned user_gpio)
```

อ่านค่าที่ใช้เป็นค่า duty cycle สูงสุดของ gpioPWM

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)
ค่ากลับคืน	เป็นค่าที่ได้จากการเซ็ตโดย gpioSetPWMrange() สำหรับกรณีที่เป็นขาที่กำลังทำงานอยู่ในโหมด Hardware PWM (ที่เซ็ตด้วยฟังก์ชัน gpioHardwarePWM() จะมีค่ากลับคืนเป็น 1000000
*ในกรณีที่ผิดพลาด PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน

```
int gpioGetPWMrealRange(unsigned user_gpio)
```

อ่านค่า duty cycle ปัจจุบันของ gpioPWM

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)
ค่ากลับคืน	เป็นค่าที่ได้จากการเซ็ตโดย gpioPWM () สำหรับกรณีที่เป็นขาที่กำลังทำงานอยู่ในโหมด Hardware PWM (ที่เซ็ตด้วยฟังก์ชัน gpioHardwarePWM() จะมีค่ากลับคืนเป็นค่าประมาณของ 250000000/PWM frequency
*ในกรณีที่ผิดพลาด PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน

```
int gpioSetPWMfrequency(unsigned user_gpio, unsigned frequency)
```

กำหนดค่าความถี่ของ PWM ที่จะใช้งาน

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)	
frequency	ค่าความถี่ เป็นค่าใดๆ ที่มีค่ามากกว่าศูนย์ โดยมีค่าความเป็นไปได้ดังนี้	
	0	กำหนดให้มีความถี่ต่ำสุดเท่าที่ไลบรารีรองรับ
	100000	กำหนดให้มีความถี่สูงสุดเท่าที่ไลบรารีรองรับ
	ค่าอื่นในในช่วง	กำหนดให้มีความถี่ตามที่กำหนด(หรือใกล้เคียงที่สุด)
ค่ากลับคืน	เป็นค่าที่ได้จากการเซ็ตโดย gpioPWM () สำหรับกรณีที่เป็นขาที่กำลังทำงานอยู่ในโหมด Hardware PWM (ที่เซ็ตด้วยฟังก์ชัน gpioHardwarePWM() จะมีค่ากลับคืนเป็นค่าประมาณของ 250000000/PWM frequency	
*ในกรณีที่ผิดพลาด PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน	

```
int gpioGetPWMfrequency(unsigned user_gpio)
```

อ่านค่าความถี่ของ PWM ที่ใช้งาน

user_gpio ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)

ค่ากลับคืน ค่าความถี่ที่เซ็ตไว้โดย gpioSetPWMfrequency()

*ในกรณีที่ผิดพลาด PI_BAD_USER_GPIO ขา GPIO ดังกล่าวไม่รองรับการทำงาน หรือไม่ได้เซ็ตไว้ในโหมด PWM

ปฏิบัติการ: การสร้างสัญญาณ Hardware timed PWM อาศัยไลบรารี pigpio

ตัวอย่างต่อไปนี้ เป็นการปรับเปลี่ยนมาจากตัวอย่างก่อนหน้า โดยหันมาใช้ Hardware Timed PWM ที่ pigpio สร้างไว้ให้ โดยใช้กับพอร์ตเดิม (GPIO18) และกำหนดช่วงค่าที่เป็นไปได้จาก 0 ถึง 20000 ดังนั้นในกรณีนี้ ค่าที่เหมาะสมสำหรับการขับเซอร์โว 9g ก็จะอยู่ในช่วงตั้งแต่ 1000 ถึง 2000

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

int PWM_pin=18;

void gpio_stop(int sig);

int main(){
    int i;

    printf("Servo PWM using Hardware Timed PWM(5%%-10%% duty cycle)\n");

    if(gpioInitialise() < 0){
         return -1;
    }
    signal(SIGINT,gpio_stop);

    gpioSetPWMfrequency(PWM_pin,50);
    gpioSetPWMrange(PWM_pin,20000);

    while(1){
        for(i=1000;i<2000;i+=10){
                gpioPWM(PWM_pin,i);
                usleep(10000);
        }
        for(i=2000;i>1000;i-=10){
                gpioPWM(PWM_pin,i);
                usleep(10000);
        }
    }
    return 0;
}

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    gpioTerminate();
    exit(0);
}
```

การสร้างสัญญาณ PWM สำหรับเซอร์โวโดยเฉพาะอาศัยไลบรารี pigpio

นอกจากการใช้ชุดฟังก์ชัน Hardware PWM หรือ Hardware Timed PWM ในการควบคุมเซอร์โวแล้ว pigpio ยังมีฟังก์ชันที่ใช้สำหรับการจัดการกับเซอร์โวโดยเฉพาะ โดยมีการกำหนดค่าความถี่ PWM ไว้ที่ 50Hz ซึ่งเป็นค่าที่นิยมใช้กับการควบคุมเซอร์โวโดยทั่วไป โดยมีฟังก์ชันให้ใช้งานดังนี้

```
int gpioServo(unsigned user_gpio, unsigned pulsewidth)
```

กำหนดค่า duty cycle ที่จะป้อนให้กับเซอร์โว

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)	
pulsewidth	คาบเวลาของช่วงสัญญาณพัลส์ที่เป็น 1 มีค่าได้ตั้งแต่ 500 ถึง 2500 หน่วยเป็นไมโครวินาที อนึ่ง การป้อนค่าที่นอกเหนือไปจากที่เซอร์โวรองรับ **จะทำให้เซอร์โวเสียหายได้** ตัวอย่างของเซอร์โว 9g ที่ใช้ในปฏิบัติการจะอยู่ในช่วง 1000 ถึง 2000 (1ms ถึง 2ms)	
ค่ากลับคืน	OK	สั่งการได้อย่างถูกต้อง
*ในกรณีที่ผิดพลาด	PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน
	PI_BAD_PULSEWIDTH	คาบเวลาที่ให้มีค่านอกเหนือจากที่รองรับ

```
int gpioGetServoPulsewidth(unsigned user_gpio)
```

อ่านค่า duty cycle ที่กำหนดโดย gpioServo()

user_gpio	ขา GPIO ที่ต้องการ (ได้ตั้งแต่ 0 ถึง 31)	
ค่ากลับคืน	ค่า pulsewidth ที่กำหนดให้ไว้ก่อนหน้าด้วย gpioServo()	
*ในกรณีที่ผิดพลาด	PI_BAD_USER_GPIO	ขา GPIO ดังกล่าวไม่รองรับการทำงาน
	PI_NOT_SERVO_GPIO	ขา GPIO ดังกล่าวไม่ได้ถูกเซ็ตให้ทำงานด้วย gpioServo()

ปฏิบัติการ: การสร้างสัญญาณ PWM สำหรับเซอร์โวโดยเฉพาะอาศัยไลบรารี pigpio

จากตัวอย่างก่อนหน้าที่สร้าง Hardware Time PWM ในตัวอย่างนี้เราเปลี่ยนมาใช้ฟังก์ชัน gpioServo() เพื่อควบคุมเซอร์โว ซึ่งจะได้โปรแกรมดังนี้

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

int PWM_pin=18;

void gpio_stop(int sig);

int main(){
    int i;

    printf("Servo PWM using Servo function(5%%-10%% duty cycle)\n");

    if(gpioInitialise() < 0){
        return -1;
    }
    signal(SIGINT,gpio_stop);
```

```
    while(1){
        for(i=1000;i<2000;i+=10){
                gpioServo(PWM_pin,i);
                usleep(10000);
        }
        for(i=2000;i>1000;i-=10){
                gpioServo(PWM_pin,i);
                usleep(10000);
        }
    }
    return 0;
}

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    gpioTerminate();
    exit(0);
}
```

ปฏิบัติการ: การสร้างสัญญาณ PWM โดยใช้ซอฟต์แวร์ควบคุม

นอกเหนือจากการสร้างสัญญาณ PWM โดยใช้ไลบรารีของ pigpio ที่มีฟังก์ชันจัดเตรียมไว้ให้เฉพาะก่อนหน้านี้ เรายังสามารถจำลองกลไกการสร้างสัญญาณ PWM โดยอาศัยซอฟต์แวร์ได้โดยการแตกเธรดใหม่ออกไปเพื่อสร้างสัญญาณ PWM ตามคาบเวลาที่กำหนด ทั้งนี้เนื่องจากระบบปฏิบัติการ Raspberry Pi OS ที่เราใช้อยู่นั้นไม่ใช่ระบบปฏิบัติการแบบเวลาจริง การกลับเข้ามาทำงานของเธรดหลังจากที่เราสั่งหยุดรอชั่วคราวอาจจะไม่มีความเที่ยงนัก ส่งผลทำให้คาบสัญญาณ PWM อาจจะมีความคลาดเคลื่อนไปมาบ้าง สำหรับการควบคุมกลไกที่ไม่ต้องการอาศัยความเที่ยงการใช้ซอฟต์แวร์ PWM ก็สามารถนำไปใช้งานได้ โดยเฉพาะในกรณีที่เรามีการเชื่อมต่อบอร์ดแยกที่เป็น GPIO ธรรมดา (ที่ไม่ใช่บอร์ดแยกฮาร์ดแวร์ PWM) ก็สามารถนำวิธีการนี้ไปใช้งานได้เช่นกัน

โปรแกรมต่อไปนี้ เราจะสร้างเธรดขึ้นมาหนึ่งเธรดสำหรับการสร้างพัลส์ PWM ตามข้อกำหนดของเซอร์โวที่ใช้งาน โดยกำหนดตัวแปรส่วนกลาง pwm เพื่อใช้รับค่า duty cycle ซึ่งมีค่าได้ตั้งแต่ 0 ถึง 100 โดยวงจรยังคงใช้จากตัวอย่างโปรแกรมก่อนหน้านี้

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด
- โปรโตบอร์ด และสายจัมพ์อีกตามต้องการ
- Logic Shifter 1 ตัว
- ไมโครเซอร์โวรุ่น 9g 1 ตัว

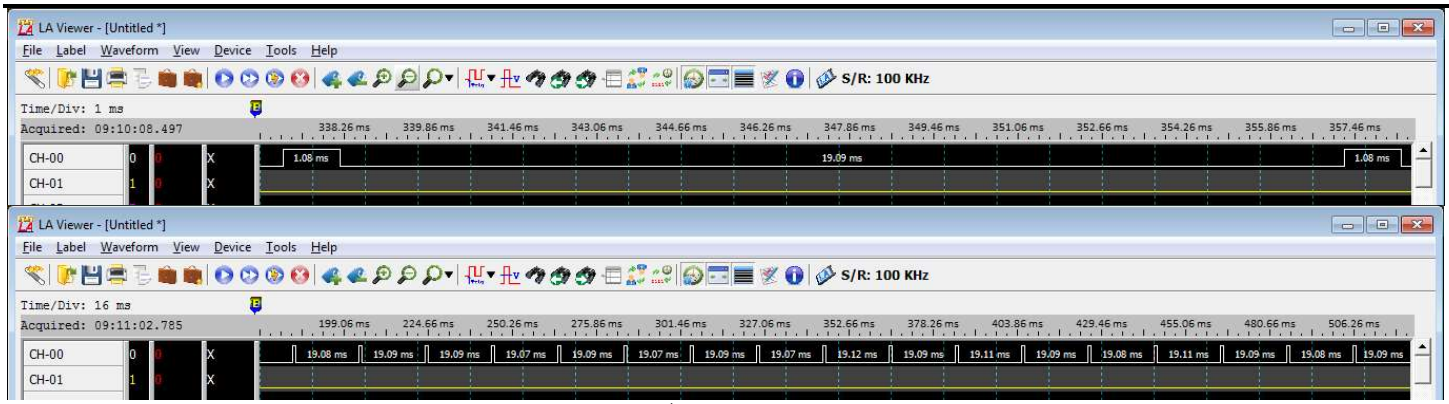
```
#include <pigpio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

int pwm=50;
int PWM_pin=18;
int running=true;

void gpio_stop(int sig);
void *servoPWM(void *param);

int main(){
    pthread_t tid;
```

```
    pthread_attr_t attr;
    int i;

    if(gpioInitialise() < 0) return -1;
    signal(SIGINT,gpio_stop);

    gpioSetMode(PWM_pin,PI_OUTPUT);
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,servoPWM,NULL);

    while(running){
        for(i=0;i<100;i++){
            pwm=i;
            if(!running)break;
            usleep(20000);
        }
        for(i=100;i>0;i--){
            pwm=i;
            if(!running)break;
            usleep(20000);
        }
    }

    pthread_join(tid,NULL);
    pthread_attr_destroy(&attr);
    gpioTerminate();
    return 0;
}

void *servoPWM(void *param){
    int pON,pOFF;
    while(running){
        pON = pwm*10+1000;
        pOFF = (100-pwm)*10+18000;
        gpioWrite(PWM_pin,1);
        usleep(pON);
        gpioWrite(PWM_pin,0);
        usleep(pOFF);
    }
    pthread_exit(NULL);
}

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    running=false;
}
```

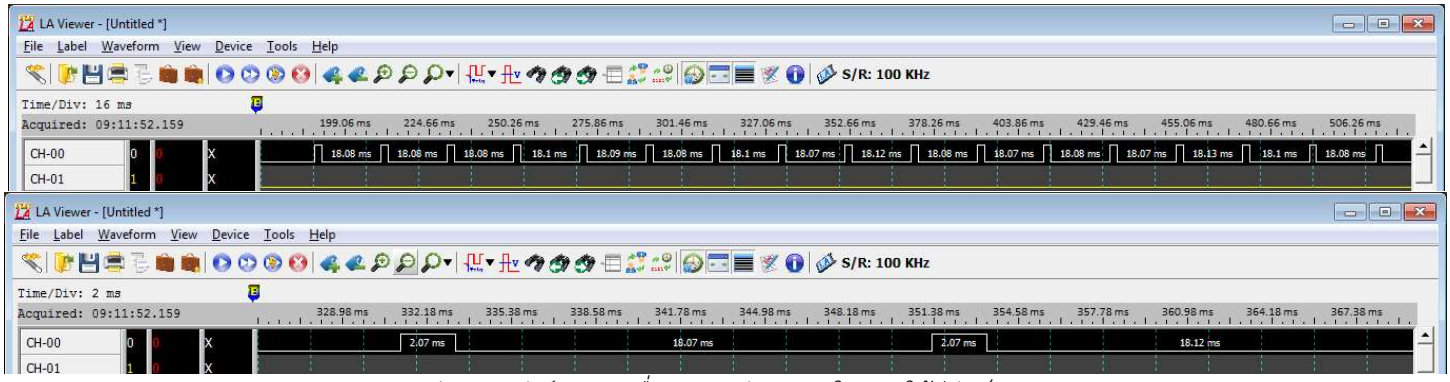
จากโปรแกรมตัวอย่างข้างต้น มีประเด็นต่างๆ ที่น่าสนใจดังนี้

- มีการนิยามตัวแปรส่วนกลาง running กำหนดค่าเริ่มต้นเป็น true เพื่อใช้บ่งสถานะว่าในขณะนี้โปรแกรมกำลังทำงานตามปกติ และเมื่อใช้กดปุ่ม CTRL-C บนแป้นพิมพ์ จะเกิดสัญญาณ SIGINT และสั่งทำงานฟังก์ชัน gpio_stop() ภายในนั้นเราเปลี่ยนค่าตัวแปร running ให้เป็น false ซึ่งตัวแปรนี้ถูกใช้เพื่อคุมการทำงานของวนรอบ while ในเธรดฟังก์ชัน servoPWM() ที่เราสร้างขึ้น และวนรอบในเธรดหลัก (ภายในฟังก์ชัน main()) ทั้งนี้เพื่อให้เธรดหลักรอจบการทำงานของเธรดสร้างสัญญาณ PWM และหลังจากนั้นจึงคืนทรัพยากรเธรดให้กับระบบ รวมทั้งหยุดการทำงานของไลบรารี pigpio อย่างเรียบร้อย ไม่เช่นนั้นเราจะพบว่าจะเกิดความผิดพลาดของการทำงาน เมื่อเรากดปุ่ม CTRL-C

- สังเกตว่าการหน่วงเวลาด้วย usleep() สลับกับการส่งสัญญาณ 1 และ 0 ออกไป เพื่อสร้างพัลส์ที่มีค่าเวลาช่วงสัญญาณ 1 และสัญญาณ 0 วนไปเรื่อยๆ เกิดเป็นสัญญาณพัลส์ที่มีค่า duty cycle เป็นไปตามที่ต้องการ เมื่อใช้ logic analyzer จับสัญญาณดู จะได้ดังนี้



รูปสัญญาณพัลส์ PWM เมื่อกำหนดค่า pwm ในเธรดให้มีค่าเป็น 0



รูปสัญญาณพัลส์ PWM เมื่อกำหนดค่า pwm ในเธรดให้มีค่าเป็น 100

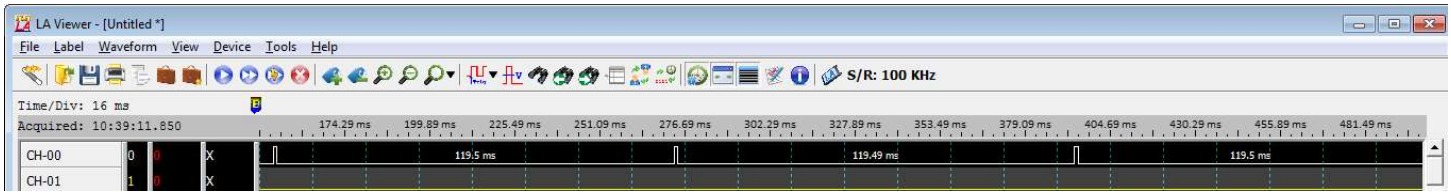
สังเกตจากสัญญาณพัลส์ที่วัดได้ จะเห็นได้ว่าค่าเวลาที่ได้มีความยาวกว่าที่ควรจะเป็นเล็กน้อย และค่าเวลาในแต่ละช่วงรอบสัญญาณไม่คงที่ ซึ่งมาจากเวลาที่โปรแกรมเองต้องทำงานอย่างอื่นในระหว่างขั้นตอนการรอ และประการสำคัญ ตัวระบบปฏิบัติการไม่ใช่ระบบปฏิบัติการเวลาจริง (realtime OS) ซึ่งต้องจัดสรรซีพียูไปทำงานในเธรดและโพรเซสอื่นๆ รวมทั้งไม่ได้มีการรับประกันว่าแต่ละเธรดจะต้องกลับมาทำงานต่อภายในช่วงเวลาที่กำหนด (รวมไปถึงไม่ได้รับประกันว่าการใช้ usleep(10) จะต้องเป็นการหยุดรอ 10 ไมโครวินาทีอย่างแม่นยำ) ส่งผลให้ หากเราลองแก้ไขโปรแกรมตัวอย่างข้างบน ให้ส่งแต่ค่า pwm คงที่ออกไป เราจะพบว่าแกนเซอร์โวจะขยับไปมาเล็กน้อยตลอดเวลาไม่นิ่ง

เราสามารถตัดปัญหาการขยับไปมาของแกนเซอร์โว อันเนื่องมาจากความไม่แม่นยำของซอฟต์แวร์ PWM ได้โดยการตรวจเช็คว่า หากค่า pwm ยังไม่เปลี่ยน เราจะไม่ส่งพัลส์ใหม่ออกไปอีก ทั้งนี้เนื่องจากตัวเซอร์โวนั้น อันที่จริงแล้วไม่จำเป็นต้องได้รับสัญญาณพัลส์ตลอดเวลา แต่ต้องการเพียงคาบเวลาช่วงสัญญาณ 1 เพื่อนำไปใช้ในการเปลี่ยนมุมของแกนเซอร์โว อนึ่ง การทำเช่นนี้ จะส่งผลให้หากมีแรงกระทำต่อแกนเซอร์โวให้บิดไปจากมุมเดิม ตัวเซอร์โวจะไม่มีแรงกระทำดันแกนให้กลับมาที่ค่าเดิมได้ (ซึ่งถ้าเราส่งพัลส์ต่อเนื่องไปยังเซอร์โว หากมีแรงกระทำให้แกนบิดไปจากเดิม ตัวเซอร์โวก็จะพยายามบิดคืนกลับมายังตำแหน่งที่อ้างโดยคาบเวลาช่วงสัญญาณ 1 ได้)

เธรดฟังก์ชัน servoPWM() ที่ปรับปรุงใหม่จะมีลักษณะดังนี้

```
void *servoPWM(void *param){
    int pON,pOFF;
    int oldPWM=-1;
    while(running){
        if(oldPWM!=pwm){
            oldPWM=pwm;
            pON = pwm*10+1000;
            pOFF = (100-pwm)*10+18000;
            gpioWrite(PWM_pin,1);
            usleep(pON);
            gpioWrite(PWM_pin,0);
            usleep(pOFF);
        }
    }
    pthread_exit(NULL);
}
```

หรืออีกทางเลือกหนึ่ง เราอาจจะทำการเว้นปล่อยพัลส์สัญญาณไปบ้างในช่วงที่ค่า pwm ยังคงที่ เพื่อให้เซอร์โวยังคงมีแรงบิดต้านอยู่ ตัวอย่างเธรดในที่นี้เรานิยามตัวแปรเพิ่มไว้อีกหนึ่งตัวคือ pwmPulse ในที่นี้กำหนดค่าเริ่มต้นไว้ที่ 10 หมายความว่าในกรณีที่ค่า pwm คงที่ เราจะปล่อยพัลส์ออกไปจริงๆ เพียง 1 รอบสัญญาณในทุกๆ สิบรอบสัญญาณ สัญญาณที่ได้จะเป็นดังรูป




```
void *servoPWM(void *param){
    int pON,pOFF;
    int oldPWM=-1;
    int count=0;
    int pwmPulse=10;

    while(1){
        if(oldPWM==pwm){
            count++;
            if(count<pwmPulse){
                count++;
                usleep(20000);
                continue;
            }else
                count=0;
        }
        oldPWM=pwm;
        pON = pwm*10+1000;
        pOFF = (100-pwm)*10+18000;
        gpioWrite(PWMport,1);
        usleep(pON);
        gpioWrite(PWMport,0);
        usleep(pOFF);
    }
    pthread_exit(0);
}
```

ปฏิบัติการ: หุ่นยนต์ตอบสนองต่อวัตถุที่เข้าใกล้

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด
- โปรโตบอร์ด และสายจัมพ์อีกตามต้องการ
- อัลตร้าโซนิกรุ่น HC-SRO4 หรือที่เทียบเท่า
- วงจร Logic Shifter ขนาด 4 channel หรือเทียบเท่า
- วงจรไฟเลี้ยง 5v DC
- เซอร์โวขนาดเล็ก 1 ตัว
- แผ่นฟิวเจอร์บอร์ดพร้อมคัตเตอร์ สำหรับนำมาสร้างเป็นแขนของเซอร์โว

อาศัยโปรแกรมที่ได้จากปฏิบัติการก่อนหน้านี้ทั้งหมด ให้เขียนโปรแกรมแตกออกเป็นหลายเธรด แต่ละเธรดทำงานจัดการในแต่ละส่วน สร้างแขนกลโดยติดเซอร์โวไว้ที่จุดหมุนและติดอัลตร้าโซนิคไว้ที่ปลายแกนดังรูปข้างต้น โดยตำแหน่งปกติ แขนกลจะมีลักษณะตั้งตรง เมื่อมีวัตถุมาใกล้แขนกล ให้แขนกลเคลื่อนหนีเบนออกไป แต่เมื่อวัตถุดังกล่าวไม่อยู่แล้ว ก็ให้แขนกลเคลื่อนกลับมายังจุดตั้งต้นดังเดิม

ให้นักศึกษาลองเลือกใช้ฟังก์ชันสร้าง PWM ในรูปแบบต่างๆ (รวมทั้งแบบการแตกเธรดออกทำงานแบบซอฟต์แวร์) เพื่อดูผลการทำงานว่ามีลักษณะเหมือนหรือแตกต่างกันหรือไม่อย่างไร

ปฏิบัติการแถม: สั่งให้บอร์ดส่งเสียงออกลำโพง

ระบบปฏิบัติการ Raspberry Pi OS ที่เตรียมไว้ให้นี้ มีโปรแกรมเล่นเสียงชื่อ omxplayer ไว้ให้ใช้งานได้ โดยสามารถรับไฟล์ mp3 และส่งเสียงออกลำโพงได้

สำหรับการเขียนโปรแกรมด้วยภาษาซี เราสามารถใช้ฟังก์ชัน system() เพื่อสั่งตัว omxplayer ได้ ดังตัวอย่างเช่น

```
system("omxplayer /home/pi/mp3/test.mp3");
```

ซึ่งจะเป็นการสั่งการให้มีการเล่นไฟล์เสียง test.mp3 ได้

ในทางปฏิบัติ นักศึกษาสามารถสร้างเธรดเพื่อเตรียมเล่นไฟล์เสียงไว้ และอาศัยการทริกจากเธรดอื่น มาสั่งงานเธรดเล่นไฟล์เสียงนี้ ซึ่งจะทำการส่งเสียงในขณะที่กลไกอื่นๆ ยังคงทำงานไปพร้อมๆ กันได้

<u>คำเตือน</u> *ไลบรารี pigpio ที่ใช้ จะมีปัญหากับการเล่นไฟล์เสียง หากโปรกรมมีการใช้งาน Hardware PWM เนื่องจาก PWM หนึ่งตัวจากสองตัวที่มีอยู่ในบอร์ด ถูกนำไปใช้กับการทำงานของเสียง ดังนั้น ให้ใช้ Software PWM ในตัวอย่างที่ให้ไว้ในบทนี้ หรือใช้งานบอร์ดวงจรเสริม Hardware PWM แยกต่างหาก*