微處理機期末專題報告

姓名:簡嘉妍

學號:4106056044

系級:大四資工系

目錄

模擬車子

I. 簡介：

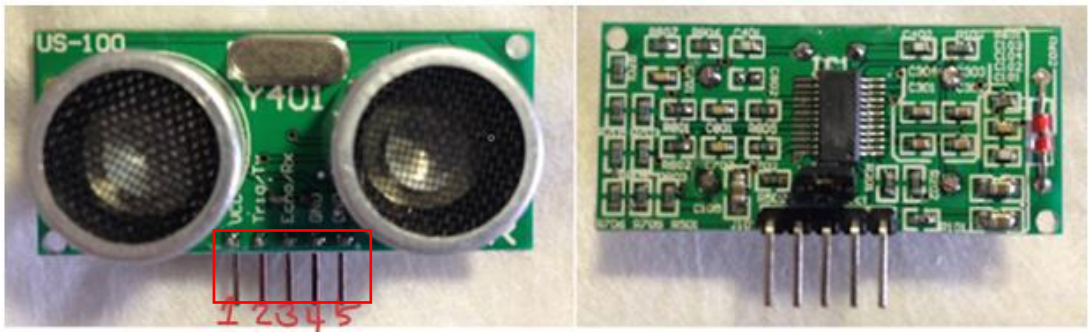
我們的目標是將車子整合 GPIO 連接超音波模組和 UART 連接超音波模組使車子能偵測與前方物體的距離避免發生碰撞，最後再使用 Interrupt 整合各個功能模組(超音波模組、陀螺儀模組、加速規模組和馬達電控板)於車體載具上。

II. 原理探討：

一台車子需具有的功能有：加/減速、前/後/左/右移動、和超音波測距。超音波模組的原理是車子發射一個電波然後利用電波反射回來的時間推算出車子與物體的距離，公式[1]:

距離=(音波發射與接收時間差 * 聲音速度(340M/S))/2

我們使用 US100(GPIO or UART)來實作如圖(一)[1],



圖(一) US-100 正反圖

[US-100 規格][2]：

- 直流電源：DC5V
- 靜態電流：小於 2mA
- 電平：1:5V /0:0V
- 感應角度：15 度以內
- 探測距離：2CM-4.5M
- 高精度：可達 0.5mm

[1]而且 US100 具有溫度感測，距離值已經有溫度調校，無需再根據環境溫度對超音波聲速進行校正。

在這個模組背面有一個 Jumper，來控制兩種模式[1]：

open：將 Jumper 拔起來，為 GPIO 模式(電位觸發模式)

short：將 Jumper 套上，此時設定為 UART 串列通訊模式

[US-100 pin 腳][1]：

- 1 號 Pin(VCC)：接 VCC 電源（範圍 2.4V~5.5V）。
- 2 號 Pin(Trig/TX)：當為 UART 模式時, 接外部電路 UART 的 TX 端；為

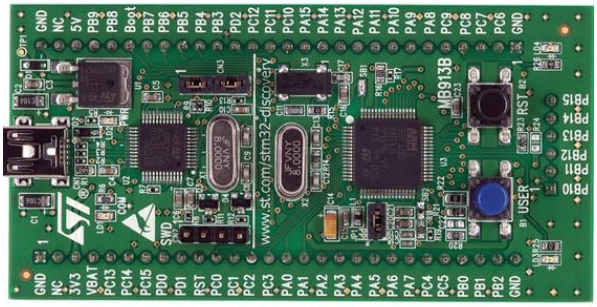
GPIO 模式時，接外部電路的 Trig 端。

- 3 號 Pin(Echo/RX)：當為 UART 模式時, 接外部電路 UART 的 RX 端；為 GPIO 模式時，接外部電路的 Echo 端。
- 4 號 Pin(GND)：接外部電路的地。
- 5 號 Pin(GND)：接外部電路的地。

III. GPIO/UART 連結超音波模組:

[材料]：

開發板：ST 官方的 STM32VL Discovery，圖(二)[3]。



圖(二) STM32VL Discovery

微控制器：STM32F100R8T6

超音波模組：US-100 超音波模組

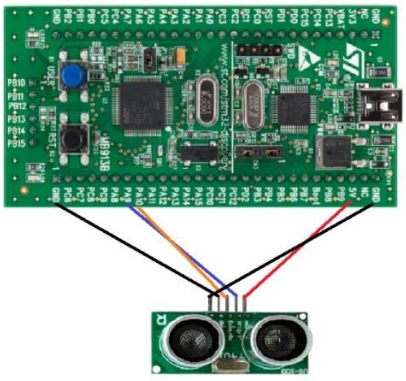
[實驗步驟]：

步驟 1：US-100 VCC 接 STM32VL Discovery 主板的+5V

步驟 2：US-100 Trig/TX 接 STM32VL Discovery 主板的 PA9

步驟 3：US-100 Echo/RX 接 STM32VL Discovery 主板的 PA10 步驟 4：US-100 GND 接 STM32VL Discovery 主板的 GND

以上步驟可參考圖(三)[1][3]。



圖(三)US100 與 STM32VL Discovery 接線圖

[工作方式][1]：

步驟 1：Trig(程式碼使用 Pin23)接腳發出一個 10us 以上的高電位(程式碼使用 10us)。

步驟 2：等待 Echo 高電平輸出。

步驟 3：一旦 Echo 高電平有輸出就開始計時，此時模組會發送 8 個 40khz 的方波，並開始自動檢測是否有返回信號。
步驟 4：當偵測到反射訊號時，Echo 接腳變為 0V 低電位。
步驟 5：計算 Echo 電位從 High 到 LOW 的時間，可知超音波來回時間而計算與物體距離。
P.s：如果加上 while(1)迴圈，就可以一直偵測前方移動物體的距離。

程式碼撰寫方式(參考[1]):
[設定 GPIO 連接超音波模組]：

```
/*每隔 1sec 會去計算一次車子與前方物體的距離*/
#include<stdio.h>
#include ”stm32f10x.h”
#include<time.h>
#define True 1
#define False 0
typedef int bool;
void delay_ms(uint16_t);
void send_trigger_pulse(void);
void wait_for_echo(bool, int);
double get_distance(void);

void delay_ms(uint16_t t)
{
	volatile unsigned long l = 0;
	for(uint16_t i = 0; i < t; i++)
		for(l = 0; l < 6000; l++){}
}
void send_trigger_pulse()
{
	GPIOA->ODR |= (1<<9);//set PA9 high
	delay_ms(0.01);//delay 0.01ms
	GPIOA->ODR &= (0<<9); //set PA9 low
}
void wait_for_echo(bool value, int timeout)
{
	int count = timeout;
	while(GPIOA->ODR |= (1<<10)!= value && count > 0)
		count = count - 1;
```

```
    //當 echo pin != value, 持續等待直到時間到(5sec)
}
double get_distance()
{
    send_trigger_pulse();
    wait_for_echo(True, 5000);//等待回復時間不超過 5sec
    clock_t start,end;
    start=clock();
    wait_for_echo(False, 5000);
    end=clock();
    double diff= start-end;
    double distance_cm=diff*340*100/2; //來回,故/2
    return distance_cm;
}

int main(void)
{
    RCC->APB2ENR |= 0xFC | (1<<14); //enable GPIO clocks

    /******** 設定 PA9 為 output 覆用 (trigger pin)********/
    /******** 設定 PA10 為 input 覆用 (echo pin)********/
    unit_t t=GPIOA->CRH;//get the content of CRH
    t=t&0xFFFFF00F;//clear the configuration bit for PA9 and PA10
    t=t|0x00000430;//configure bit for PA9 as output and PA10 as input
    GPIOA->CRH=t;//load CRH as the new value
    while(1)
    {
        printf("cm=%f" , get_distance());
        delay_ms(1000);//delay 1000ms=1sec
    }

    return 0;
}
```

[設定 UART 連接超音波模組]：

```
/*當輸入"check_distance"時,車子才會測量自己與前方物體的距離,並顯示出來*/
#include<stdio.h>
#include "stm32f10x.h"
#include<time.h>
```

```
#include<string.h>
#define SIZE_OF_CMD_BUF 16
#define True 1
#define False 0
typedef int bool;
void delay_ms(uint16_t);
void send_trigger_pulse(void);
void wait_for_echo(bool, int);
double get_distance(void);
void usart1_sendByte(unsigned char);
uint8_t usart1_recByte();
void usart1_sendStr(char);
void check();
char command1[]="cm=";
static char cmdbuf[SIZE_OF_CMD_BUF];
void delay_ms(uint16_t t)
{
	volatile unsigned long l = 0;
	for(uint16_t i = 0; i < t; i++)
		for(l = 0; l < 6000; l++){}
}
void send_trigger_pulse()
{
	GPIOA->ODR |= (1<<9); //set PA9 high
	delay_ms(0.01);//delay 0.01ms
	GPIOA->ODR &= (0<<9); //set PA9 low
}
void wait_for_echo(bool value, int timeout)
{
	int count = timeout;
	while(GPIOA->ODR |= (1<<10)!= value && count > 0)
		count = count - 1;
}
double get_distance()
{
	send_trigger_pulse();
	wait_for_echo(True, 5000);//等待回復時間不超過 5sec
	clock_t start,end;
```

```
	start=clock();
	wait_for_echo(False, 5000);
	end=clock();
	double diff= start-end;
	double distance_cm=diff*340*100/2; //來回,故/2
	return distance_cm;
}
void usart1_sendByte(unsigned char c)
{
	USART1->DR = c;
	while((USART1->SR&(1<<6)) == 0); //wait until the TC flag is set
	USART1->SR &= ~(1<<6); //clear TC flag
}
uint8_t usart1_recByte()
{
	while((USART1->SR&(1<<5)) == 0); //wait until the RXNE flag is set
	return USART1->DR;
}
void usart1_sendStr(char str[])
{
	unsigned int i=0;
	for(i=0;i<strlen(str);i++)
		usart1_sendByte(str[i]);

}


void check()//輸入的指令是否為"check_distance",是的話去求與物體的距離
{
	unsigned int i=0;
	char command1[]="check_distance";
	if(strcmp(cmdbuf,command1)==0)
	{
		float f=get_distance();
		char distance[50]; //size of the numbe
		sprintf(distance, "%g", f); //convert float to string
		usart1_sendStr(command1);
		usart1_sendStr(distance);
```

```
    }
    for(i=0;i<SIZE_OF_CMD_BUF;i++) //清空 buffer
        cmdbuf[i]='\0';
}
int main(void)
{
    RCC->APB2ENR |= 0xFC | (1<<14);
    /* Enable clocks for GPIO ports and USART1 clock */
    //USART1_init
    /******** 設定 PA9 為 output 覆用 ********/
    /******** 設定 PA10 為 input 覆用 ********/
    GPIOA->ODR |= (1<<10);
    GPIOA->CRH=0x444448B4;
    //Rx1(pin10)=input with pull-up, Tx1(pin9)=alt.func output
    USART1->CRI=0x200C;
    USART1->BRR=7500;
    while(1)
    {
        uint8_t c = usart3_recByte(); //接收輸入的指令
        check(); //對輸出的指令作相對應的動作
        delay_ms(1000);//delay 1000ms=1sec
    }

    return 0;
}
```

IV. 實現方式

(如何用 interrup 將各個功能模組整合)

[Interrupt 介紹][4]：

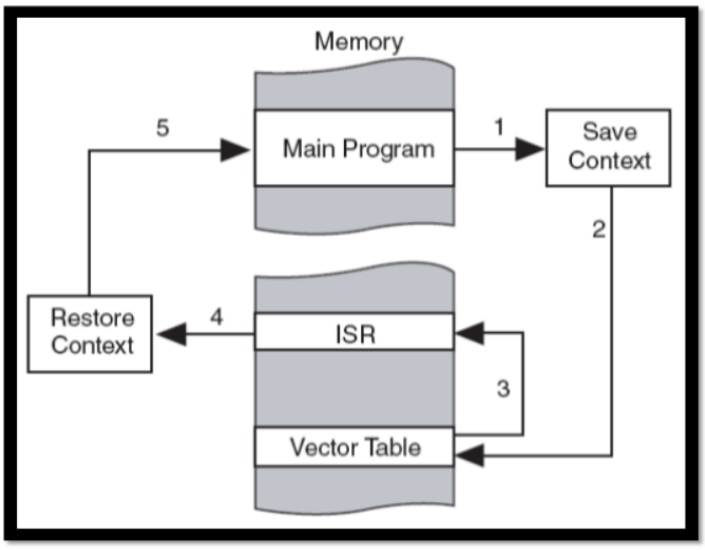
中斷是指處理器接收到外圍硬體或軟體的信號，導致處理器通過一個執行 context switch 並處理該事件。

[Interrupt 處理流程][4]：

1. 暫停目前 process 執行並保存此 process 當時執行狀況。
2. OS 根據 Interrupt ID 查尋 Interrupt vector 並取得 ISR (Interrupt Service Routine) 起始位址。
3. ISR 執行。
4. 執行完成，恢復先前 process 執行狀況。

5. 回到原先中斷前的執行。

以上步驟可參考圖(四)[4]。



圖(四) Interrupt 處理流程

[Interrupt 種類][5]：

◆ SysTick interrupt：

會定期生成中斷請求。允許操作系統(OS)執行上下文切換(context switch)以同時執行多個任務。對於不需要操作系統的應用程序，SysTick 可用於計時或作為需要定期執行的任務的中斷源。

範例[6]：

/*程式每隔 1sec 會去執行 SysTick_Handler 內的工作(toggle PC13), while 迴圈可以是空的*/

```
#include “stm32f10x.h”
void SysTick_Handler()
{
    GPIOC->ODR^=(1<<13);/*toggle PC13*/
}
int main()
{
    RCC->APB2ENR= 0xFC; /* Enable clocks for GPIO ports */
    GPIOC->CRH = 0x44344444; /* PC13 as output */

    SysTick->LOAD=9000000-1; /*STRELOAD=72000000/8-1*/
    SysTick->CTRL=0x03; /*Clock=AHB clock/8,TickInt enable, Enable=1*/
    while(1)
    {
    }
}
```

◆ I/O Port Interrupt：

因 Input/output port 所觸發的 interrupt。

範例[7]：

```
/*功能:每當 PB4 因外部因素而從 pull-up 變 pull-down,此時會發生 interrupt,
PC13 會發生切換(1->0 or 0->1)*/
#include “stm32f10x.h”
void delay_ms(uint16_t t);
void EXTI4_IRQHandler() /* interrupt handler for EXTI4 */
{
    EXTI->PR = (1<<4);
    /* Pending flag 在 Interrupt 時被 set, 故要 clear */
    GPIOC->ODR ^= 1<<13;
    /* toggle PC13 */
}
int main()
{
    RCC->APB2ENR |= (0xFC | 1); /* Enable clocks for GPIO ports and AFIO */
    GPIOB->CRL = 0x44484444; /* PB4 as input */
    GPIOB->ODR = (1<<4); /* pull-up PB4 */
    GPIOC->CRH = 0x44344444; /* PC13 as output */
    AFIO->EXTICR[1] = 1<<0; /* EXTI4 = 1 (selects PB4) */
    EXTI->FTSR = (1<<4); /*當 PB4 從 pull-up 被 pull-down 時 Interrupt 被觸發
*/
    EXTI->IMR = (1<<4); /* enable interrupt EXTI4 */
    NVIC_EnableIRQ(EXTI4_IRQn); /* enable the EXTI4 interrupt*/
}
```

◆ USART Interrupt：

因 Sending/Receiving data 所觸發的 interrupt。

範例[8]：

```
/*程式透過 USART1 取得 character,並作出相對應的對策*/
#include “stm32f10x.h”
void USART1_IRQHandler()
{
    uint8_t c=USART1->DR; //get received data
    if((c==’H’)||(c==’h’))
        GPIOC->ODR|=(1<<13); //make PC13 high
    else if((c==’L’)||(c==’l’))
        GPIOC->ODR&=~(1<<13); //make PC13 low
```

```
}
int main()
{
    RCC->APB2ENR |= 0xFC | (1<<14);
    /* Enable clocks for GPIO ports and USART1 clock */
    GPIOC->CRH = 0x44344444; /* PC13 as output */
    /*USART_intit*/
    GPIOA->ODR|=(1<<10); //pull up PA10
    GPIOA->CRH=0x444448B4; //RX1=Input TX1=output
    USART1->CR1=0x2024; //receive int. enable, receive enable
    USART1->BRR=7500; //72MHz/9600bps=7500
    NVIC_EnableIRQ(USART1_IRQn); /* enable USART1_IRQ*/
    while(1)
    {
    }
}
```

[Interrupt 整合各個功能模組]：

超音波測距離：

使用 SysTick Interrupt，設定固定時間週期，每當時間到時會跳到 SysTick_Handler function，此時如果將 function 內寫了測輛距離的指令則車子每隔固定時間會去量一次自己與周圍物體的距離。

除此之外，用超音波測距離可以使用 I/O Port Interrupt，車子先發出一波 (trigger pin=1，echo pin=0)，當此波因為碰撞到物體而反射回來時 trigger pin=1，echo pin=1，此時觸發 I/O Port interrupt，程式跳去 EXTIn_IRQHandler() functin 計算間格時間並推出車子與物體的距離。

控制智能車行駛方向和加減速：

用鍵盤控制智能車。使用 UART interrupt，當收到鍵盤 q (結束程式，車子停止)、w (前進)、x (後退)、d (向右轉)、a (向左轉)、s(減速)和 z(加速)的指令後，會跳到 USART1_IRQHandler function 去執行 function 相對應的指令。

範例：(設定 GPIO 連接超音波模組，並用 interrupt 整合)

```
#include<stdio.h>
#include ”stm32f10x.h”
#include<time.h>
#define True 1
#define False 0
typedef int bool;
void send_trigger_pulse(void);
void wait_for_echo(bool, int);
```

```
double get_distance(void);
void delay_ms(uint16_t t);
void SysTick_Handler()
{
	printf("cm=%f" , get_distance());
}
void delay_ms(uint16_t t)
{
	volatile unsigned long l = 0;
	for(uint16_t i = 0; i < t; i++)
		for(l = 0; l < 6000; l++){}
}
void send_trigger_pulse()
{
	GPIOA->ODR |= (1<<9); //set PA9 high
	delay_ms(0.01);//delay 0.01ms
	GPIOA->ODR &= (0<<9); //set PA low
}
void wait_for_echo(bool value, int timeout)
{
	int count = timeout;
	while(GPIOA->ODR |= (1<<10)!= value && count > 0)
		count = count - 1;
}

double get_distance()
{
	send_trigger_pulse();
	wait_for_echo(True, 5000);//等待回復時間不超過 5sec
	clock_t start,end;
	start=clock();
	wait_for_echo(False, 5000);
	end=clock();
	double diff= start-end;
	double distance_cm=diff*340*100/2; //來回,故/2
	return distance_cm;
}
```

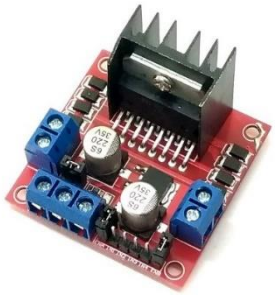
```
int main(void)
{
    RCC->APB2ENR |= (0xFC | 1); /* Enable clocks for GPIO ports and AFIO */
    /******** 設定 PA9 為 output 覆用 (trigger)********/
    /******** 設定 PA10 為 input 覆用 (echo)********/
    GPIOA->CRH = 0x44444834; /* PA9 as output and PA10 as input*/
    SysTick->LOAD=9000000-1; /*STRELOAD=72000000/8-1*/
    SysTick->CTRL=0x03; /*Clock=AHB clock/8,TickInt enable, Enable=1*/
    /*程式每隔 1sec 會去執行 SysTick_Handler 內的工作,故 while 迴圈可以是空
    的*/
    while(1)
    {
    }

    return 0;
}
```

加分內容：

I. 以 PWM 控制步進馬達：

馬達使用的型號是：L298N(GPIO and PWM)，圖(五)[9]。



圖(五) L298N 機電馬達驅動模組

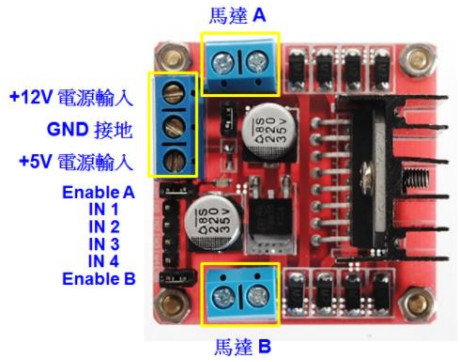
[L298N 規格][10]：

- 主控晶片：L298N
- 電壓：5V
- 驅動電壓：5V～35V
- 電流：0mA～36mA
- 驅動電流：2A

- 工作溫度：-20℃～135℃
- 最大功率：25W

[L298N 特點][10]：

工作電壓高(最高可達 46V)；輸出電流大(瞬間峰值可達 3A)；持續工作電流為 2A；額定功率 25W。內含兩個 H 橋的高電壓大電流全橋式驅動器可同時驅動兩個馬達，如圖(六)[10]。



圖(六) L298N 可同時驅動兩個馬達

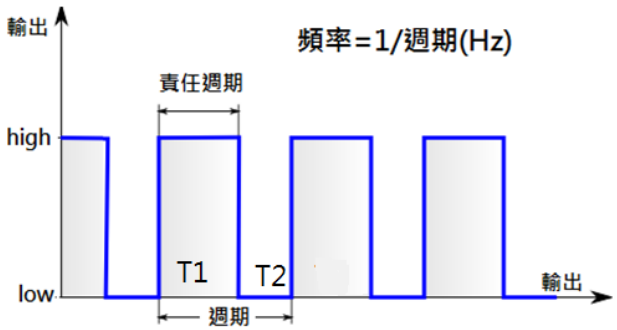
[控制馬達方向][11]：

腳位設定如下表，當 ENA 為 0 時，不提供電源，馬達漸漸停止。如果 ENA 為 1 時，則看 IN1、IN2 的值來決定，IN1、IN2 同時為 0，則馬達漸漸停止，同 ENA(EnableA)為 0 的結果；IN1=0、IN2=1，馬達反轉；IN1=1、IN2=0，馬達正轉。IN1=1、IN2=1，馬達剎車，會立即停止，也就是說，它會提供一個扭力來鎖住馬達。ENB(另一個馬達)想法同 ENA。

ENA	IN1	IN2	狀態	ENB	IN3	IN4
0	X	X	停止	0	X	X
1	0	0	停止	1	0	0
1	0	1	反轉	1	0	1
1	1	0	正轉	1	1	0
1	1	1	剎車	1	1	1

[控制馬達速度][12]：

所謂 PWM，就是脈衝寬度調製技術，其具有兩個很重要的參數：頻率和佔空比。頻率，就是周期的倒數，1/(T1+T2)；佔空比，就是高電平在一個周期內所佔的比例，T1/(T1+T2)。又因為這是數位的輸出，其輸出值不是邏輯上的 0 就是邏輯上的 1，所以又可視為週期性的方波，如下圖(七)[11]所示。

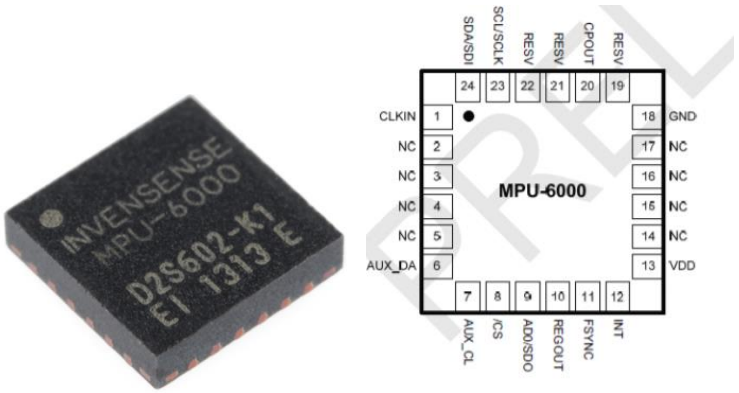


圖(七)PWM 介紹

通過改變單位時間內脈衝的個數可以實現調頻；通過改變佔空比可以實現調壓。佔空比越大，所得到的平均電壓也就越大，幅值也就越大，速度增加；占空比越小，所得到的平均電壓也就越小，幅值也就越小，速度下降。

II. 以 SPI 連接陀螺儀模組

使用的陀螺儀模組型號：MPU6000(SPI)，圖(八)[12][13]。



圖(八) MPU6000

[MPU6000 規格][14]：

- 供電電源：2.375～3.46V（內部低壓差穩壓）
- 通信方式：標準 IIC or SPI 通信協定
- 晶片內置 16bit AD 轉換器 / 16 位元資料輸出
- 陀螺儀範圍：±250 / 500 / 1000 / 2000°/s
- 加速度範圍：±2 / ±4 / ±8 / ±16g
- 引腳間距 2.54mm

[MPU6000 Pin 腳][15]：

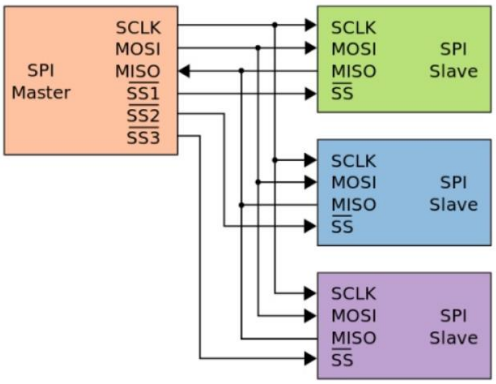
- CLKIN (pin1) – 可選的外部時鐘輸入，如果不用則連到 GND。
- AUX_DA (pin6) – I2C 主串行数据，用於外接傳感器。
- AUX_CL (pin7)– I2C 主串行時鐘，用於外接传感器。
- /CS – SPI (pin8) 片選（0=SPI mode）。
- AD0/SDO (pin9) – I2C Slave 地址 LSB（AD0）；SPI 串行數據输出（SDO）。

- REGOUT (pin10)– 校准濾波電容連線。
- FSYNC (pin11) – 偵同步数字输入。
- INT (pin12) – 中斷數字輸出（推挽或開漏）。
- VDD (pin13)– 電源電壓及數字 I/O 供電電壓。
- GND (pin18)– 電源地。
- RESV (pin19,21,22) – 預留，不接。
- CPOUT (pin20)– 電赫泵電容連線。
- SCL/SCLK (pin23)–I2C 串行時鐘（SCL）；SPI 串行時鐘（SCLK）。
- SDA/SDI (pin24)–I2C 串行數據（SDA）；SPI 串行數據输入（SDI）。
- NC (pin 2, 3, 4, 5, 14,15, 16, 17) – 不接。

[SPI 原理][16]：

SPI 與 I2C 相同是可以接多個裝置的，而且傳輸速度比 I2C 更快，而且與 UART/RS-232 一樣，發送與接收可同時進行。

不過 SPI 也有缺點，隨著連接裝置數的增加，線路也是要增加的，每增加一個連接裝置，至少要增加一條(如圖(九))，不像 I2C 可以一直維持只要兩條。而 SPI 在一對一連接時需要四條，一對二時要五條，一對三時要六條，即 N+3 的概念。另外 SPI 比 I2C 更少纜線化運用，多半是更短距離的連接。在實務上，I2C 較常用來連接感測器，而 SPI 較常用來連接 EEPROM 記憶體、Flash 記憶體（記憶卡），或一些液晶顯示器。



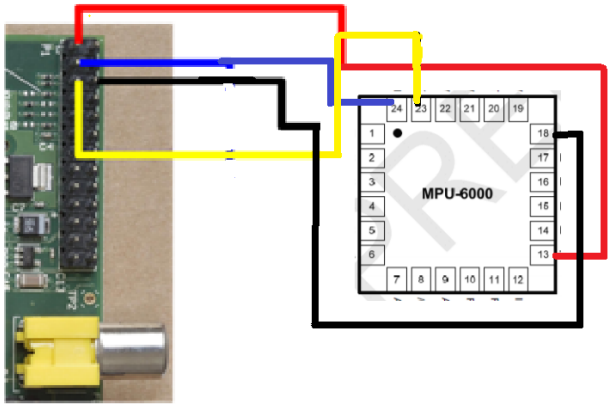
圖(九)SPI 每多連接一個裝置，必須增加一條新的 SS 線路

[材料]：

Raspberry Pi 2 Model B x 1、MPU-6000 模組 x 1、連接線 x 4 條。

[接線](參考[17])：

將 Raspberry Pi 的第 1 pin 接到 MPU-6000 的 VDD(pin13)；第 3 pin 接到 MPU-6000 的 SDI(pin24)；第 5 pin 接到 MPU-6000 的 SCLK(pin23)；第 6 pin 接到 MPU-6000 的 GND(pin18)，如圖(十)[13]。



圖(十) Raspberry Pi 與 MPU-6000 連接

撰寫程式碼如下(參考[18])：

1.include 函式庫：#include <MPU6000.h>、#include<stdio.h>

2.初始化 SPI。

```
bool mpu6000_spi::init(int sample_rate_div,int low_pass_filter)
{
    unsigned int response;
    spi.format(8,0);
    /*第一個參數:該值必須為 8，因為當前 SPI 數據值僅支持資料值得位數為 8 位。第二個參數: 0: the data line is active when SCK goes to high and the data values are read */
    spi.frequency(1000000); // set the frequency for the SPI
    //FIRST OF ALL DISABLE I2C
    select(); //enable MPU6000 的 communication bus
    response=spi.write(MPUREG_USER_CTRL);
    // Disable I2C bus (recommended on datasheet)
    response=spi.write(BIT_I2C_IF_DIS);
    deselect();//disable MPU6000 的 communication bus
    //RESET CHIP
    select();
    response=spi.write(MPUREG_PWR_MGMT_1);
    response=spi.write(BIT_H_RESET);
    deselect();
    wait(0.15);
    //WAKE UP AND SET GYROZ CLOCK
    select();
    response=spi.write(MPUREG_PWR_MGMT_1);
    response=spi.write(MPU_CLK_SEL_PLLGYROZ);
```

```
        deselect();
        //DISABLE I2C
        select();
        response=spi.write(MPUREG_USER_CTRL);
        response=spi.write(BIT_I2C_IF_DIS);
        deselect();
        //WHO AM I?
        select();
        response=spi.write(MPUREG_WHOAMI|READ_FLAG);
        response=spi.write(0x00);
        deselect();
        if(response<100){return 0;}//COULDN'T RECEIVE WHOAMI
        //SET SAMPLE RATE
        select();
        response=spi.write(MPUREG_SMPLRT_DIV);
        response=spi.write(sample_rate_div);
        deselect();
        // FS & DLPF
        select();
        response=spi.write(MPUREG_CONFIG);
        response=spi.write(low_pass_filter);
        deselect();
        //DISABLE INTERRUPTS
        select();
        response=spi.write(MPUREG_INT_ENABLE);
        response=spi.write(0x00);
        deselect();
        return 0;
    }
```

3.讀取 X、Y、Z 軸加速度的值。(回傳資料單位:Gs)

```
float mpu6000_spi::read_acc(int axis)
{
    uint8_t responseH,responseL;
    int16_t bit_data;
    float data;
    select();
    switch (axis)
    {
```

```
        case 0://X 軸
        responseH=spi.write(MPUREG_ACCEL_XOUT_H | READ_FLAG);
        break;
        case 1://Y 軸
        responseH=spi.write(MPUREG_ACCEL_YOUT_H | READ_FLAG);
        break;
        case 2://Z 軸
        responseH=spi.write(MPUREG_ACCEL_ZOUT_H | READ_FLAG);
        break;
    }
    responseH=spi.write(0x00);
    responseL=spi.write(0x00);
    bit_data=((int16_t)responseH<<8)|responseL;
    data=(float)bit_data;
    data=data/acc_divider;
    deselect();
    return data;
}
```

4.讀取陀螺儀數據。(回傳資料待為:度/秒)

```
float mpu6000_spi::read_rot(int axis){
    uint8_t responseH,responseL;
    int16_t bit_data;
    float data;
    select();
    switch (axis){
        case 0: //X 軸
        responseH=spi.write(MPUREG_GYRO_XOUT_H | READ_FLAG);
        break;
        case 1: //Y 軸
        responseH=spi.write(MPUREG_GYRO_YOUT_H | READ_FLAG);
        break;
        case 2: //Z 軸
        responseH=spi.write(MPUREG_GYRO_ZOUT_H | READ_FLAG);
        break;
    }
    responseH=spi.write(0x00);
    responseL=spi.write(0x00);
    bit_data=((int16_t)responseH<<8)|responseL;
```

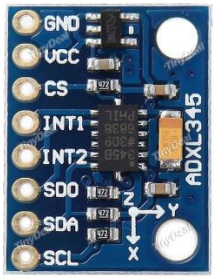
```
        data=(float)bit_data;
        data=data/gyro_divider;
        deselect();
        return data;
}
```

6. select/deselect function 撰寫如下：

```
/*usage: enable and disable mpu6000 communication bus*/
void mpu6000_spi::select() {
        //Set CS low to start transmission (interrupts conversion)
        cs = 0;
}
void mpu6000_spi::deselect() {
        //Set CS high to stop transmission (restarts conversion)
        cs = 1;
```

III. 以 I2C 連接加速規模組

使用的加速規模組型號：ADXL345(I2C)，圖(十一)[19] 。



圖(十一) ADXL345(I2C)

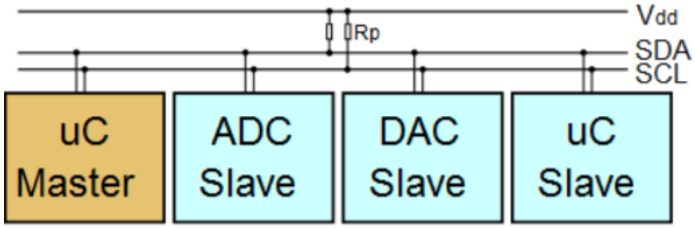
[ADXL345 Pin 腳][20]：

- VCC - Supply voltage range: 2.0 V to 3.6 V 電壓供給。
- GND - 接地。
- CS - Chip Select. 選擇晶片。
- INT1 - Interrupt 1 Output. 中斷輸出 1。
- INT2 - Interrupt 2 Output. 中斷輸出 2。
- SDO - Serial Data Output. 串列資料輸出 (SPI 通訊)。
- SDA - Serial Data. 串列資料線。
- SCL - Serial Communications Clock. 串列通訊時脈線。

[單晶片介面 I2C 介紹][21]：

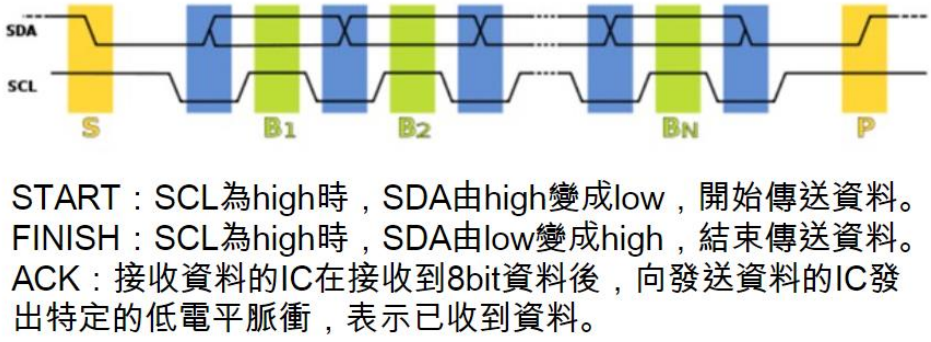
串列通訊 I²C (I square C)

I²C 是內部整合電路的稱呼，是一種串列通訊匯流排，使用多主從架構，如圖(十二)[21]。



圖(十二) 串列通訊 I²C 結構

[22]I²C 只使用兩條雙向開放集極(Open Drain)(串列資料(SDA)及串列時脈(SCL))並利用電阻將電位上拉。I²C 允許相當大的工作電壓範圍，但典型的電壓準位為+3.3V 或+5v。傳輸速率：標準模式（100 Kbit/s）~ 快速模式（400 Kbit/s），如圖(十三)[23]。



圖(十三) I²C 只使用兩條雙向開放集極 (串列資料及串列時脈)並利用電阻將電位上拉

[實作材料]：

開發板：ST 官方的 STM32VL Discovery

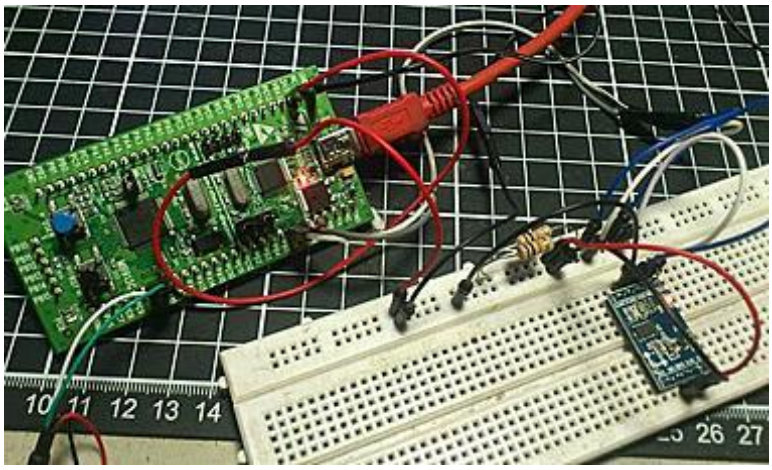
微控制器：STM32F100R8T6，圖(十四)[24](下面程式碼相容 F100、F103 、F105、F107 等所有 F1 系列)

圖(十四) STM32F100R8T6

三軸加速度器：ADXL345。

[接線](參考[25])：

VCC 接在 3V3 的位置；GND 接 GND；CS 和 VCC 接在一起，把 CS 的電位拉高和 VCC 相同，目的在告訴晶片是走 I2C 的協定；IN1 和 IN2 是負責驅動中斷的兩個輸出引腳，在這邊實作的過程不會用到，所以不用接；SDO 屬於 SPI 協定，因此在這個範例也用不到所以不用接；SDA 腳位設定 PB7，SCL 腳位設定 PB6。實際接線如圖(十五)[26]。



圖(十五) ADXL345 接線圖

[程式碼][26]：

```
#include "stm32f10x.h"
#include <stdio.h>
#define I2C_ADDRESS 0xA7 // ADXL345 I²C 地址
void Init_I2C() {
    I2C_InitTypeDef I2C_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    I2C_Cmd(I2C1,ENABLE); //啟用 I2C

    /*啟用 I2C1 RCC 時鐘*/
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* 設定 I2C1 的 SDA 與 SCL 腳位 PB6 = SCL ，PB7 = SDA */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
    GPIO_Init(GPIOB, &GPIO_InitStructure);


    /* 設定 I2C1 */
    I2C_InitStructure.I2C_Mode = I2C_Mode_SMBusHost;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0x00; // STM32 自己的 I2C 地址
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress =
    I2C_AcknowledgedAddress_7bit;
```

```
    I2C_InitStructure.I2C_ClockSpeed = 100000 ; // 設定 I2C 時鐘速度為 100K
    I2C_Init(I2C1, &I2C_InitStructure);
}

/*  讀 ADXL345 佔存器中的 x , y, z 的值*/
int xla, xha, yla, yha, zla, zha;
float x, y, z;
void readValue(){
    I2C_start(I2C1, I2C_ADDRESS, I2C_Direction_Transmitter);
    // 在 主發送 模式中 準備開始 發送
    I2C_write(I2C1, 0x32 | (1 << 7)); // 指定佔存器
    I2C_stop(I2C1); // 停止發送
    I2C_start(I2C1, I2C_ADDRESS, I2C_Direction_Receiver);
    // 在主發送模式中準備開始 發送
    xla = I2C_read_ack(I2C1);
    // 從 ADXL345 讀取一個 byte 並請求後面一個 byte
    xha = I2C_read_ack(I2C1);
    // 從 ADXL345 讀取一個 byte 並請求後面一個 byte
    yla = I2C_read_ack(I2C1);
    // 從 ADXL345 讀取一個 byte 並請求後面一個 byte
    yha = I2C_read_ack(I2C1);
    // 從 ADXL345 讀取一個 byte 並請求後面一個 byte
    zla = I2C_read_ack(I2C1);
    // 從 ADXL345 讀取一個 byte 並請求後面一個 byte
    zha = I2C_read_ack(I2C1);
    // 從 ADXL345 讀取一個 byte 並請求後面一個 byte
    I2C_read_nack(I2C1);
    // 讀取一個 byet 後，就不再讀取後面一個 byte (停止傳輸)
    /* 一定要強制轉成 short 型別，因為有些數字向左移動八位後
     * 會超過 short 大小，進而變成負數。
     * 例如：65535 + 1 = -65536
     *       65535 + 2 = -65535
     *       65535 + 3 = -65534
     */
    x = (((short)(xha << 8)) + xla) / 256.0F;
    y = (((short)(yha << 8)) + yla) / 256.0F;
    z = (((short)(zha << 8)) + zla) / 256.0F;
}
```

```
int main(){
    Init_I2C(); // 初始化 I2C
}
```

資料來源：

一般題目：

[1] Arduino 筆記(71)：MPU-6050 (GY-521) 三軸陀螺儀+三軸加速計感測模組，
https://atceiling.blogspot.com/2019/09/arduino57mpu-6050-gy-521.html

[2] US-100 超音波 (帶溫度補償)，
https://shop.playrobot.com/products/us-100-urtalsonic

[3] STM32VLDISCOVERY - Evalu. board STM32 Value line Discovery，
https://www.distrelec.biz/en/evalu-board-stm32-value-line-discovery-st-stm32vldiscovery/p/17387239

[4] OS - Ch2 中斷、I/O、系統呼叫、OS 結構設計 和 虛擬機，
https://mropengate.blogspot.com/2015/01/operating-system-ch2-os-structures.html

[5] Systick Timer (SYSTICK)，
https://www.keil.com/pack/doc/CMSIS/Core/html/group__SysTick__gr.html

[6] Program12-1 SysTick Interrupt，
https://lms.nchu.edu.tw/media/923418#24_d41d8cd98f00b204e9800998ecf8427e

[7] Program12-2 Using PB4 external Interrupt，
https://lms.nchu.edu.tw/media/923418#40_a839ff9ab132fd515f6b71d6797fba80

[8] Program12-3 Receiving data using the UART1 Interrupt，
https://lms.nchu.edu.tw/media/923418#43_4c3d03adaa97dc939984864caa45ce2b

加分題：

[9] L298N 馬達驅動模組 可控制直流電機 步進馬達 適用樹莓派 Arduino 智能車機器人 紅，
https://shopee.tw/L298N%E9%A6%AC%E9%81%94%E9%A9%85%E5%8B%95%E6%A8%A1%E7%B5%84%E5%8F%AF%E6%8E%A7%E5%88%B6%E7%9B%B4%E6%B5%81%E9%9B%BB%E6%A9%9F%E6%AD%A5%E9%80%B2%E9%A6%AC

%E9%81%94%E9%81%A9%E7%94%A8%E6%A8%B9%E8%8E%93%E6%B4%BE
Arduino%E6%99%BA%E8%83%BD%E8%BB%8A%E6%A9%9F%E5%99%A8%E4
%BA%BA-%E7%B4%85-i.4491023.2148260418

[10] Raspberry Pi 筆記(14)：用鍵盤透過無線網路控制智能車，
https://atceiling.blogspot.com/2014/03/raspberry-pi.html

[11] 陳昱仁 / 部落格 / Arduino / L298N 馬達驅動 IC & PWM8898，
https://bach.ccu.edu.tw/Site/nu14075/Blog/dir_Br2ysq/article_orvQpQ.html?group_login=-1

[12] MPU-6000 - Motion Processing Unit，
https://www.sparkfun.com/products/retired/11234

[13] MPU6000 Datasheet，
https://www.datasheetq.com/MPU6000-doc-Unspecified

[14] MPU-6050/60003 三軸陀螺儀加速度傳感器 6 軸姿態 傾斜度 GY-52，
https://www.ruten.com.tw/item/show?22025708186416

[15] 《MPU-6000 / MPU-6050 产品说明书》，
http://www.ruikang.net/upload/datasheet/MPU6050.pdf

[16] 【Maker 進階】認識 UART、I2C、SPI 三介面特性，
https://makerpro.cc/2016/07/learning-interfaces-about-uart-i2c-spi/

[17] Raspberry Pi 筆記(26)：MPU-6050 加速度計與陀螺儀感測器，
https://atceiling.blogspot.com/2017/02/raspberry-pi-mpu-6050.html

[18] MPU6000.cpp，
https://os.mbed.com/users/brunoalfano/code/MPU6000_spi//file/2edbd561b520/MPU6000.cpp/

[19] GY-291 ADXL-345 數位三軸重力加速度傾斜度模塊 GY291 ADXL345，
https://www.ruten.com.tw/item/show?21633321133008

[20] [Day 23]-用 JS 控制 Arduino 吧！三軸一起來，速度與激情！- Johnny Five 之 Accelerometer 三軸加速度計，

https://ithelp.ithome.com.tw/articles/10226147?sc=rss.iron

[21] I²C 維基百科，自由的百科全書，
https://zh.wikipedia.org/wiki/I%C2%B2C

[22] I²C，
https://cloud921.pixnet.net/blog/post/298361879

[23] SIOC 實驗 9：I2C 楊郁莉/陳慶瀚 MIAT 實驗室，
https://slideplayer.com/slide/5160539/

[24] 5pcs/lot STM32F100R8T6 STM32F100R8T6B LQFP64，
https://imall.com/product/5pcs-lot-STM32F100R8T6-STM32F100R8T6B-LQFP64/Home-Improvement-Computer-Office-Special-Category-Luggage-Bags-Men%27s-Clothing-Toys-Hobbies-Food-Lights-Lighting-Accessories/aliexpress.com/4000808323812/144-58525031/en

[25] 三軸加速度計 on webduino，
http://arduinojackychi.blogspot.com/2016/02/on-webduino.html

[26] STM32F1 入門教學：讀取 ADXL345 三軸加速度計【使用 IIC、I2C】，
https://lolikitty.pixnet.net/blog/post/165575970