朝陽科技大學
資訊工程系

【專題成果報告】

以體阻抗技術實現即時心率偵測系統

指導教授：劉省宏　　教授
專題組員：許恆誌　(10227047)
林哲均　(10227129)
鄧湘琦　(10234015)

中華民國 105 年 12 月

目　　錄

圖 目 錄

表　目　錄

摘要

本專題的主要目標為用體阻抗容積描繪方法來偵測即時心率。在心臟收縮和舒張時，心房及心室內的體積變化，會造成身體體阻抗有相對性的改變，測量此體阻抗訊號的變化，可以抓取到心跳周期的時間，進而顯示即時心率。我們實現的方法為藉由類比電路抓取心臟搏動時造成身體阻抗而產生的訊號，接著將此訊號送入數位系統，其核心微控器採用 TI MSP430 G2553，藉由程式設定與演算法算出心臟跳動的週期，實現心率偵測的目的。

關建字：體阻抗容積描繪法、心率、TI MSP430

Abstract

The main goal of this project is to use the body impedance plethysmogram signal to detect the heart rate in real time. In the heart contraction and relaxation, the volume changes of atrial and ventricular cause the relative change of the body impedance. Therefore, we can detect the heart rate in real time from the impedance plethysmogram signal, and show it. Our designed system including a TI MSP430 G2553 microcontroller which could do the analog to digital conversion, and detected the duration of each heart beat by the algorithm. Then we finish a heart rate monitor in real time.

Key words: Impedance plethysmogram, Heart rate, TI MSP430

1. 專題簡介

1.1 研究動機

健康監控是近期一直很熱門的話題，像是有計步或心率監控的運動手環一直都被受歡迎，慢跑選手或自行車騎士等都很需要這類功能的產品。這幾年來有許多科技大廠如蘋果或三星都有在這方面投資很大的心力來開發相關產品，不難看出這是潛在需求很大的一塊市場。

市面上已經存在的產品大部分都是用光容的方法來實現心率偵測。雖然這種方法被廣泛應用，但準確率容易受到外界光源的影響，所以需要緊貼著手腕，且功耗較高。體阻抗測量方法雖然比較不普及，但此方法可以降低功耗，且對人體的影響不大，實作產品也不會非常困難。若未來可以用此種方法做出實品，那對現存的市場產品將會是一個很大的衝擊。

1.2 研究目標

我們專題的目標為藉由類比電路抓取心臟搏動時造成身體阻抗變化的訊號，經過數位系統，其藉由微控器進行類比數位轉換、周邊控制與心跳抓取演算法，算出心臟跳動的週期，實現心率偵測的目的，同時藉由電源電路對電壓的轉換來供應類比電路板及數位系統所需不同電力。

以體阻抗的方式來偵測即時心率。相較於 Apple Watch 等等產品用光容的方法偵測心率，體阻抗測量方法有更低的耗電流與更穩定的特點。實驗初步階段需要將貼片貼於雙手形成較大的迴路，接下來電路優化後可以把貼片貼於單手縮小迴路，並且可以將貼片面積縮小到相當於手錶的寬度，斯毫不遜色於目前市場上的產品。

雖然我們的專題對體阻抗偵測心率僅於試驗階段，還有許多進步空間，若可以對電路改良及優化，那會是更令人期待的。在製作專題的過程由於時間有限，無法將每個地方都做到盡善盡美,像是電路板體積過於龐大,還有線路過長容易受到外界雜訊所干擾，都有改善空間。

2. 類比電路

人體組織為電解性的,且其阻抗效應為容抗為主。因此要測量體阻抗時，需輸入一交流的定電流訊號，始可產生正比於體阻抗變化的電位訊號變化。本專題針對經過心臟

的迴路阻抗，由於心臟的收縮和舒張期間，心臟內的體液有明顯的改變所反映的體阻抗變化亦會隨著心臟搏動而變化。圖 2.1 為體阻抗測量示意圖，由韋恩震盪器產生 100Khz 的交流訊號，再經由一定的電流電路將訊號以表面電送入人體內。再以表面電及抓取迴路上之訊號，前置放大器再經由一定電流電路，將訊號萃取出為單極訊號，其放大倍率為 10，並採用 JFET 型的運算放大器來增加皮膚、電擊和電路間的阻抗匹配。經由乘法器作為調解處理，將 100Khz 的調變訊號和心跳搏動的身體阻抗分離，經帶通濾波器的頻寬 1~3hz，取出心臟搏動的身體阻抗訊號，再將此微弱訊號放大 500 倍，即可得到即時心率訊號，如有訊號不乾淨等雜訊，可重複幾次帶通濾波器和放大等步驟。最後將此心率訊號經由數位系統處理，擷取出即時心率，並顯示在器七段顯示器上[1][2]。

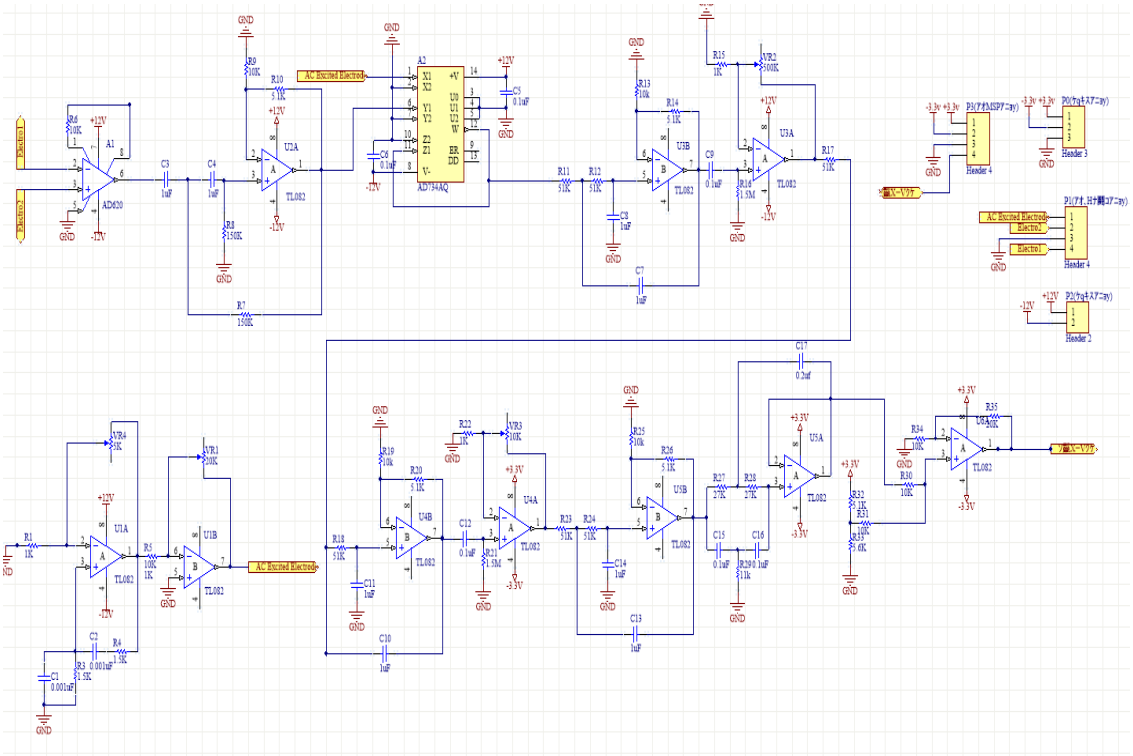


圖 2.1 類比電路電路圖

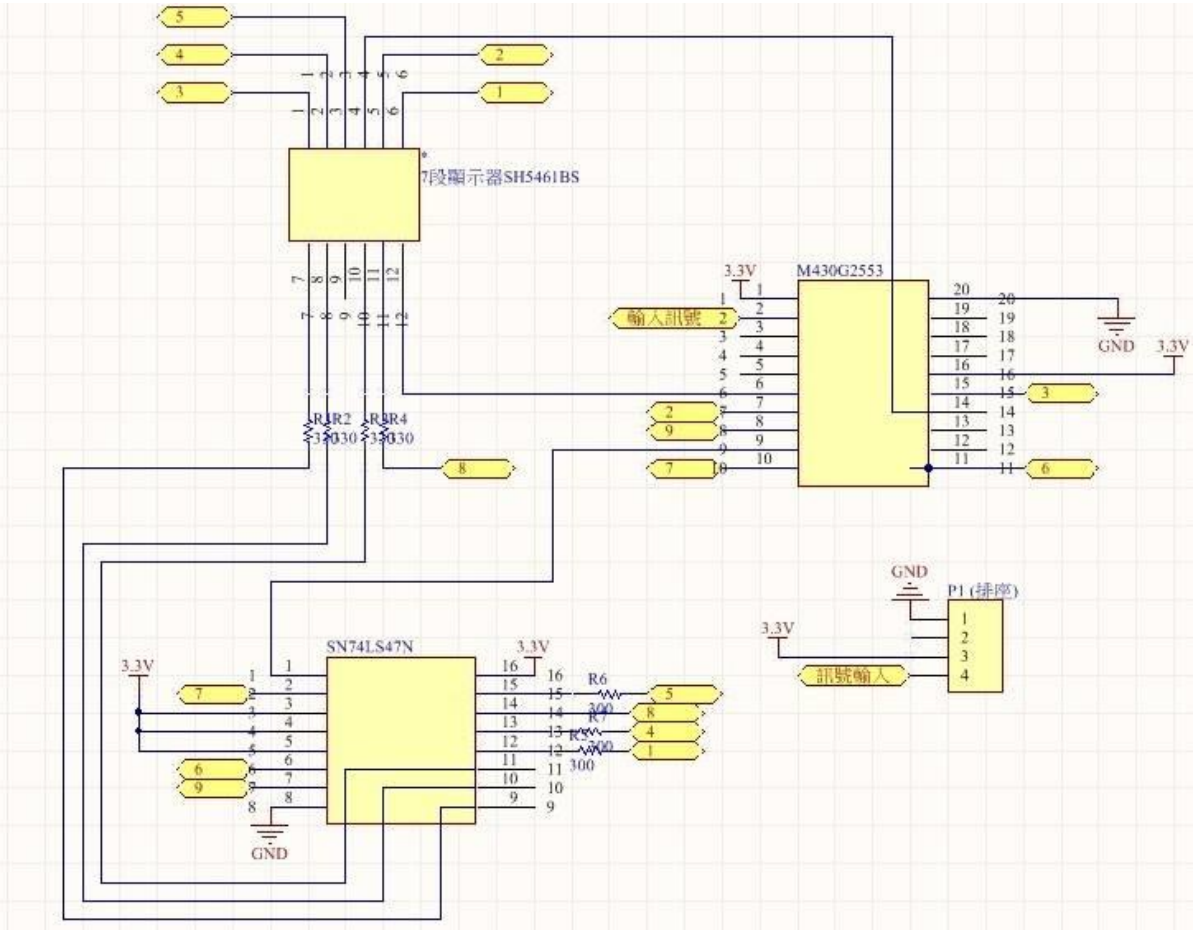


圖 2.2 七段顯示器電路板設計圖

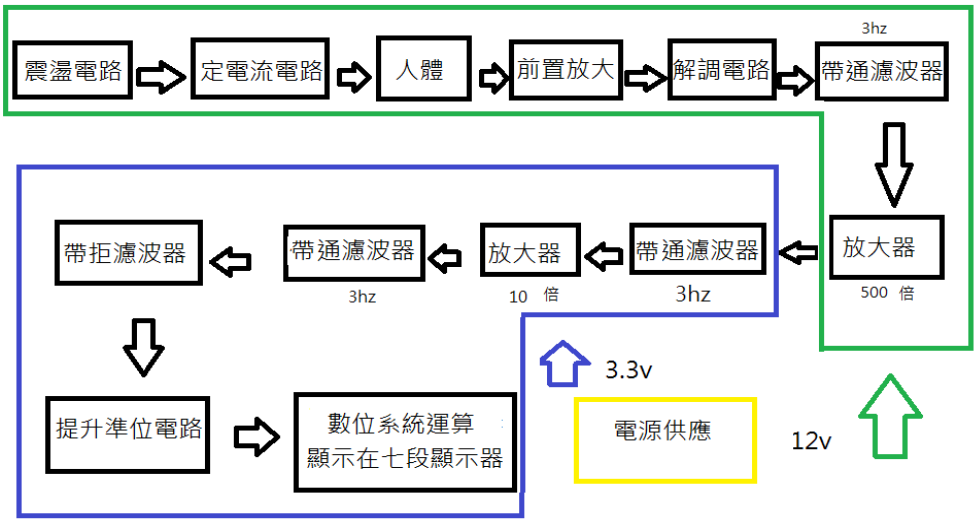


圖 2.3 體阻抗測量示意圖

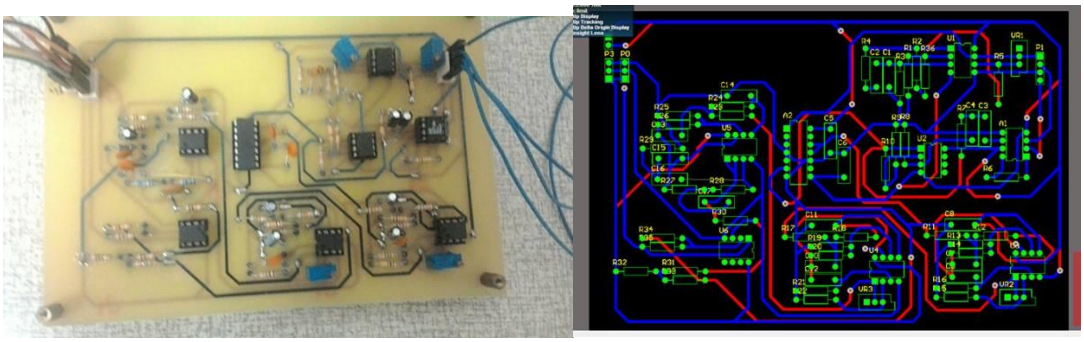


圖 2.4 類比電路電路板和電路圖

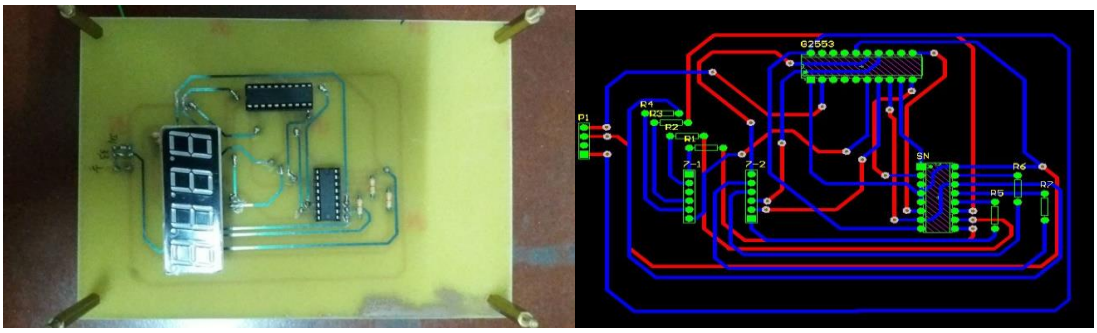


圖 2.5 七段顯示器電路板和電路圖

2.1 韋恩震盪器

韋恩震盪器是一種弦波式震盪器，利用正回授方式，電路維持等幅震盪。其基本原理是由一個電橋網路與非反向放大器所組成因。如圖 2.6 為放大器是非反向的所以其相角為 0 度為了滿足巴克豪生準則，故電橋網路相角需為 0 度，而電橋達到平衡才會有存電阻網路，此時相才會為 0 度。

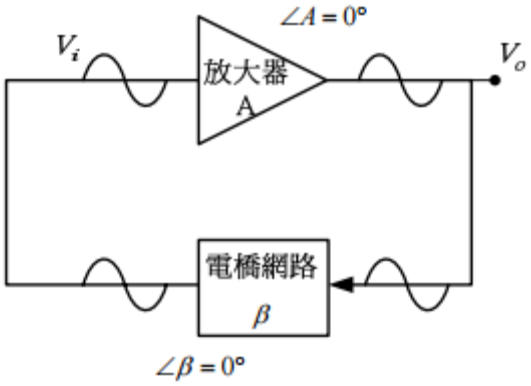


圖 2.6 電橋平衡角度

如圖 2.6 為例

$$\frac{R_3}{R_4} = \frac{Z_1}{Z_2}$$

$$\frac{R_3}{R_4}=\frac{R_1-jX_{c1}}{R_2||jX_{c2}}=\frac{X_{c1}R_2+X_{c2}R_1}{R_2X_{c2}}+j\frac{R_1R_2-X_{c1}X_{c2}}{R_2X_{c2}}$$

因為上面公式左邊為實數，所以右邊故為虛數，虛數部分需要為零。所以

$$R_1R_2-X_{c1}X_{c2}=0$$

$$\rightarrow R_1R_2=X_{c1}X_{c2}$$

$$\rightarrow \mathrm{f}=\frac{1}{2\pi\sqrt{R_1R_2C_1C_2}}$$

這樣的話，實部就會為

$$\frac{R_3}{R_4}=\frac{X_{c1}R_2+X_{c2}+R_1}{R_2X_{c2}}=\frac{R_1}{R_1}+\frac{X_{c1}}{X_{c2}}=\frac{R_1}{R_2}+\frac{1/\omega C_1}{1/\omega C_1}$$

如果 R_1=R_2=R，C_1=C_2=C 則

$$\mathrm{F}=\frac{1}{2\pi RC}$$

$$\frac{R_3}{R_4}=2$$

故保證震盪，必須$\frac{R_3}{R_4}\geq 2$

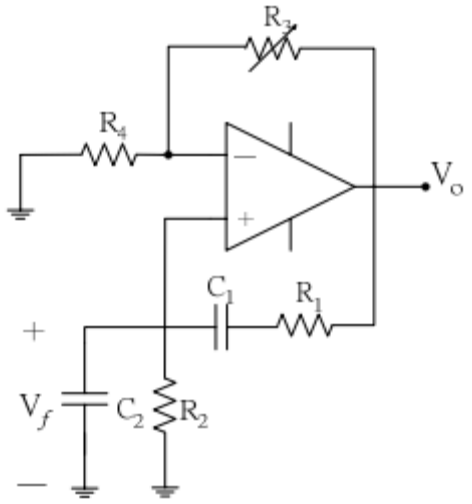


圖 2.7 韋恩震盪電路如要達到電橋平衡須滿足$\frac{R_3}{R_4}=\frac{Z_1}{Z_2}$

2.2 定電流電路

定電流源顧名思義就是輸出電流為一定值，無論負載 R_L 如何改變，其通過之電流 I_L都保持不變，圖 2.8 為一簡單定電流電路，其電路是由一個電流源和輸出電阻並聯而成，電流源所供給的電流即為輸出電阻上的電流 I_S與負載電阻電流 I_L之和，若電流源之輸出電阻遠大於負載電阻，則大部分之電流便將流過負載，而構成一定電流源。

$I_s R_s = I_L R_L = I \times R_t (R_t = R_s // R_L)$

$$I_L = \frac{I * R_t}{R_L} = \frac{I * \frac{R_L R_s}{R_L R_s}}{R_L}$$

$$\therefore I_L = \frac{R_s}{R_L + R_s} * I$$

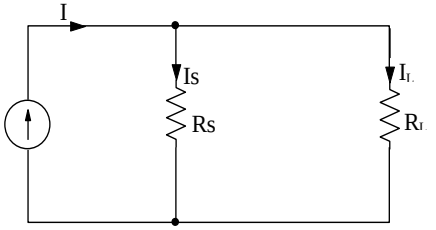


圖 2.8 簡易定電流電路

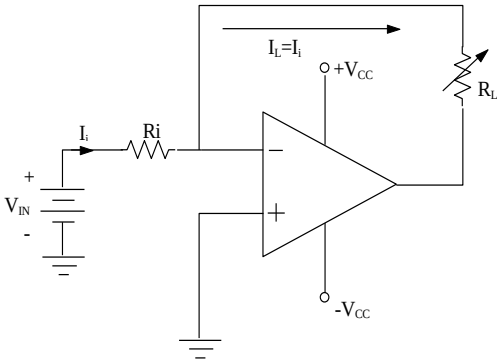


圖 2.9 基本 OPAMP 構成定電流電路

圖 2.9 為基本 OP. AMP 構成的定電流源電路，其電路是利用一電壓源 V_{IN} 經過電阻器 Ri 以提供一定電流 I_i，因為 OP. AMP 的反相輸入端為一虛擬接地點，因此輸入電流 $I_{i=\frac{V_{IN}}{R_i}}$，由於此電路輸入阻抗非常高，幾乎沒有電流可以流進內部，所以全部 I_i 流經 R_L 即 $I_L=I_i=V_{IN}/R_i$ 。由此只要 V_{IN} 及 R_i 保持常數，即使 R_L 再怎樣改變，I_L 仍為定值。此電路電流一旦由 R_i 決定後，不論負載電阻 R_L 如何變化，負載電流 I_L 均幾乎不受影響。

2.3 儀表放大器(前置放大)

功用是表面電及抓取迴路上之訊號，儀表放大器是由三個放大器所共同組成，其中電阻 R 與 R_x 需要再可變電阻 1k~10k 範圍以內，設固定電阻為 R，調整 R_x 來改變放大增益值。

儀表放大器公式:$V_0 = (1 + \frac{2R}{R_x})(V_1 - V_2)$

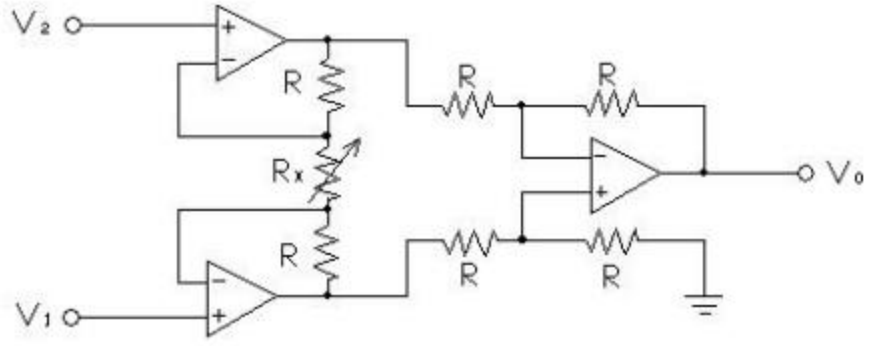


圖 2.10 由三個放大器所共同組成的儀表放大器

2.4 解調電路

人體是一個體阻抗容積描繪法，心臟搏動的訊號耦合在送入身體的載波訊號中，因此，須透過解調電路，將心臟搏動所造成的體阻抗訊號解調出來人體是一個體阻抗容積描繪法，心臟搏動的訊號耦合在送入身體的載波訊號中，因此，須透過解調電路，將心臟搏動所造成的體阻抗訊號解調出來市面上已有許多 ic 可做為解調電路，使用 AD734 這顆 ic 作為解調電路。

針對調變電路中最常使用到 FM 調變(Frequency modulation)來解調訊號(恢復到原來的訊號。

FM 調變方式為將載波頻率變化後傳送的方式進行,下圖有 2 個波(圖 2.11)(圖 2.12)

圖 2.11 為載波，圖 2.12 為信號波，圖 2.13 為解調波(FM 波)

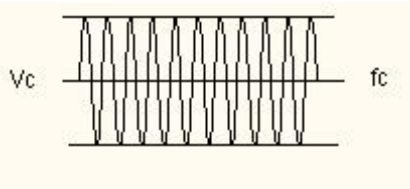


圖 2.11 載波

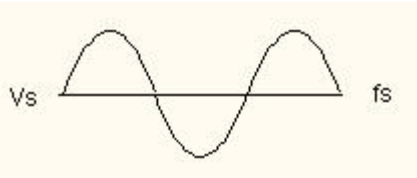


圖 2.12 訊號波

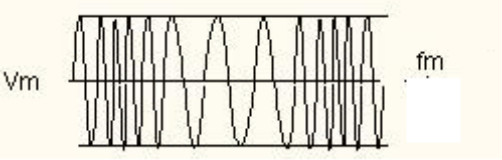


圖 2.13 解調過的 FM 波

$$V_C = V_{cm}\cos\omega_C t = V_{cm}\cos 2\pi f_C t$$
$$V_S = V_{sm}\cos\omega_S t = V_{sm}\cos 2\pi f_S t$$

圖 2.14 相對關係公式

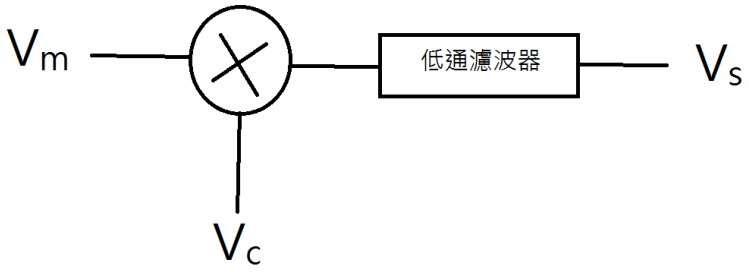


圖 2.15 乘法器調變概念圖

$$\text{載波 } V_c = V_{cm}\cos 2\pi f_c t$$

$$\text{訊號波} V_s = V_{sm}\cos 2\pi f_s t$$

V_c和 V_s調變為 V_m，V_m的公式如下

$$V_m = V_{mm}\cos 2\pi(f_c + f_s)t$$

將調變訊號與載波訊號相乘，再經由乘法器調變可獲得調變訊號。

調變訊號為下式

$$V_m * V_c = V_{cm}V_{sm}\cos(2\pi f_c t)\cos(2\pi(f_c + f_s)t)$$

$$= \frac{1}{2}V_{cm}V_{sm}\left(\cos(2\pi f_c)t + \cos 2\pi(2f_c + f_s)t\right)$$

低頻　　高頻

2.5 濾波器

（1）低通濾波器:

顧名思義就是讓低頻訊號通過，高頻訊號會衰減幾乎為 0。

先定義 $n_1=n_2=0$，$n_0 \neq 0$。

則$T(S) = \frac{n_0}{S^2+S\frac{\omega_p}{Q_p}+\omega_p^2}$

圖 2.16 為其振幅響應之區分，當 $n_0=\omega_P$ 時，直流增益為 1(0 dB)。加入交流訊號後，當其訊號頻率低時(小於ω_P)，訊號幾乎未被衰減即通過，此區為通帶；當頻率逐漸增加，超過轉折頻率後，因分母為 S 頻域之二次方程式，故增益會以斜率(Slope) 約為-40dB/decade 的速度下降，輸出訊號被衰減，此區為轉換帶；當頻率高於截止頻率ω_S後，不論輸入訊號準位為何，均無輸出，此時為阻絕帶(Stop Band)。

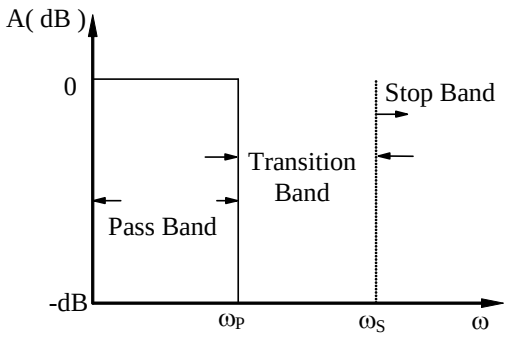


圖 2.16 低通濾波震幅響應

(2) 高通濾波器(High-Pass Filter，HP)：

先定義 $n_0=n_1=0$，$n_2\neq 0$

則$T(S)=\frac{n_2S^2}{S^2+S\frac{\omega_p}{Q_p}+\omega_P^2}$

兩個傳輸零點均位於 S=0 處，表示其直流增益為 0，且當 $n_2=1$ 時，其高頻增益為 1。高通濾波器之振幅響應區分如圖 2.16 所示。

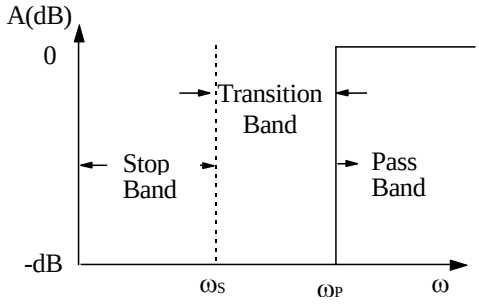


圖 2.17 高通濾波震幅響應

(3)帶拒濾波器

只讓特定頻率無法通過，帶拒濾波器各分別有兩個阻絕帶與通帶，以單一電路設計製作時，困難度較高，故通常用高通與低通濾波器之串並聯來合成。

先定義 $n_1=0$。

則$T(S)=\frac{n_2S^2+n_0}{S^2+S\frac{\omega_p}{Q_p}+\omega_p^2}=n_2\frac{S^2+\omega_n^2}{S^2+S\frac{\omega_p}{Q_p}+\omega_p^2}$

傳輸零點在 $S=\pm j\omega_n$，故傳輸增益在$\omega=\omega_n$ 時為 0。由ω_n 與ω_P 間大小關係，帶拒濾波器又可細分三種：

(a) $\omega_n=\omega_P$，點拒濾波器。

直流增益=高頻增益=n_2。

(b) $\omega_n \geq \omega_P$，低通點拒濾波器(LPN)。

直流增益=$n_2\omega_n^2/\omega_P^2$，高頻增益=n_2。

(c) $\omega_n \leq \omega_P$，高通點拒濾波器(HPN)。

直流增益=$n_2\omega n^2/\omega P^2$，高頻增益=n_2。

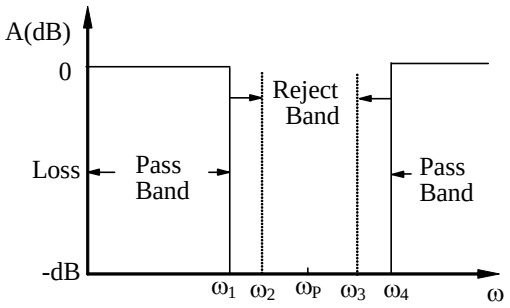


圖 2.18 兩旁唯有訊號功率輸出，中間衰減幾乎為 0。

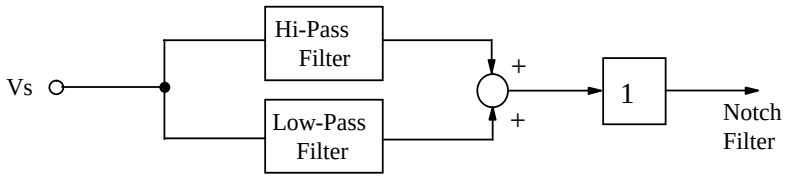


圖 2.19 帶拒濾波器合成原理

2.6 放大器

典型的運算放大器中設有非反相輸入（V_{in}(+)）、反向輸入（V_{in}(-)）與輸出（V_{out}）。雖然未顯示於圖表中，但運算放大器另有兩個電源輸入（正極與負極），並且可能包含輸入補償及其他端子

反向放大器:圖 2.19 中的電路會放大並反轉（逆轉）輸入訊號（相位），然後輸出結果。電路使用負反饋：部分輸出訊號會被反轉，並回到輸入。在由於輸出 V_{out} 是透過電阻器 R 2 連接至反相輸入(-)，因此會產生反饋。 若輸出未連接電源電壓，則外放至反相(-)及非反相(+)輸入的電壓將會相等；這兩個輸入會像短路由於虛擬短路與非反相輸入間的差動為 0 V，因此 A 點也將為 0 V。根據歐姆定律，可得出 I = V_{in}/R 的結果。由於運算放大器的輸入阻抗極高，因此幾乎不會有流入反相輸入(-)的電流。因此，I 會流過 A 點和 R_2，表示 I_1 和 I_2 幾乎相等。接著，根據歐姆定律， V_{out} = −I × R，其中由於 I_2 從電壓為 0 的 A 點流出。

如何使用輸入與輸出間的關係，計算出運算放大器的增益。具體而言，$V_{out}/V_{in} = (-I_1) \times R_2)) / ((I_1) \times R_1)) = -R_2)/R_1)$。因為輸出波形的相位與輸入波形相位相反，因此增益為負數。增益是完全取決於電阻 R_2 及 R_1 的比例。因此，只要改變電阻，便可改變增益。雖然運算放大器本身的增益較高，但適度使用負反饋便可將實際的放大降低至所需的等級

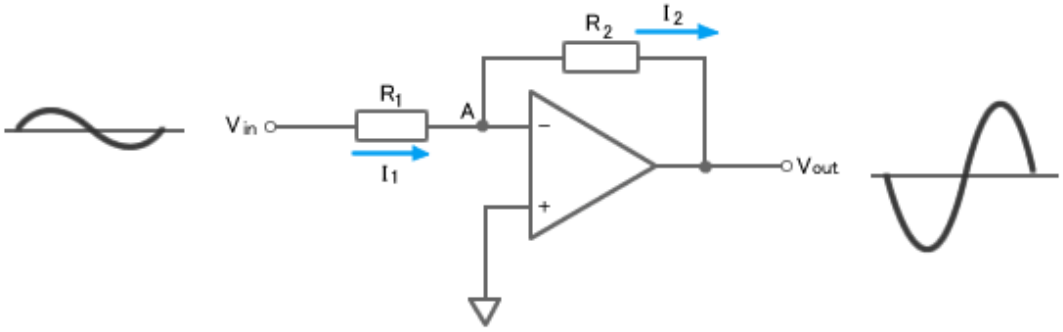


圖 2.20 反向放大器

非反向放大器:

非反相放大器與反相放大器的兩大不同之處在於輸出波形與輸入波形相位同步且輸入會進入非反相輸入端子(+)。(如圖 2.20)非反相及反相電路都是使用負反饋。關於此電路的運作方式，非反相(+)及反相(-)輸入皆在電壓 V_{in}。因此，A 點也在 V_{in}，根據歐姆定律可得知，R_1 的電壓為 $V_{in} = R_1 \times I_1$。此外，由於運算放大器的輸入基本上沒有電流流入，因此可得出 $I_1 = I_2$，由於 V_{out} 即 R_1 與 R_2 電壓的總和，因此可得出 $V_{out} = R_2 \times I_2 + R_1 \times I_1$。將這些等式重新整理後，便可算出增益 G，例如：$G = V_{out}/V_{in} = (1 + R_2/R_1)$。

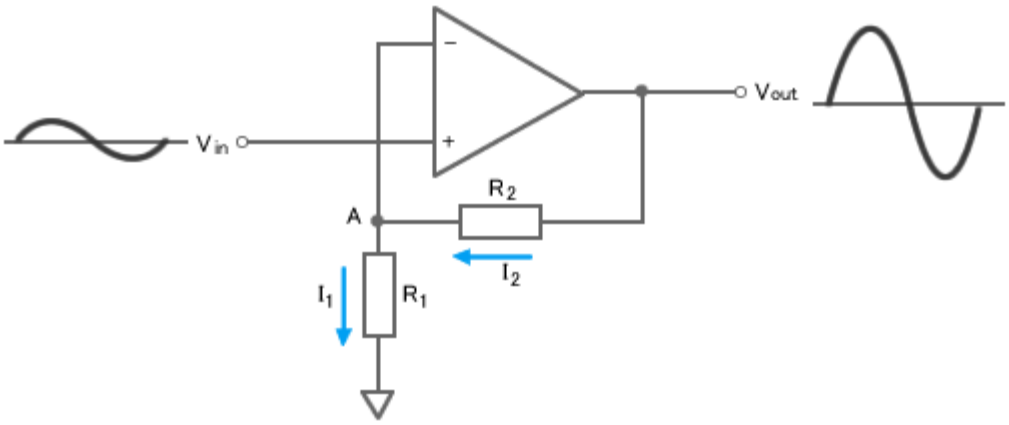
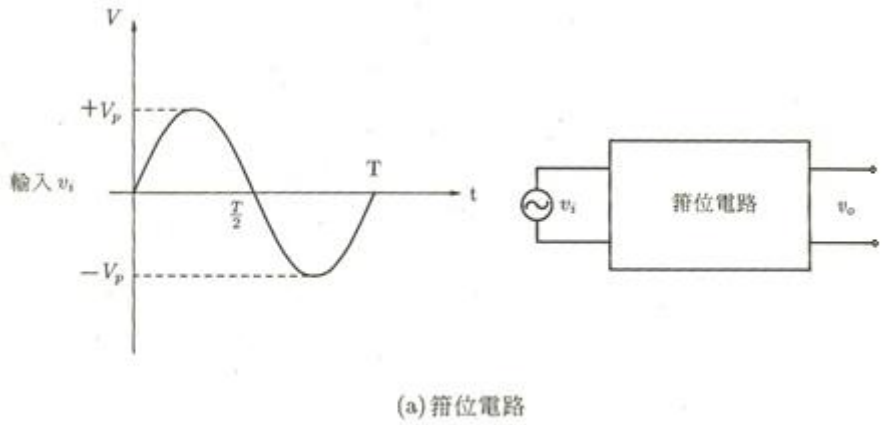


圖 2.21 非反向放大器

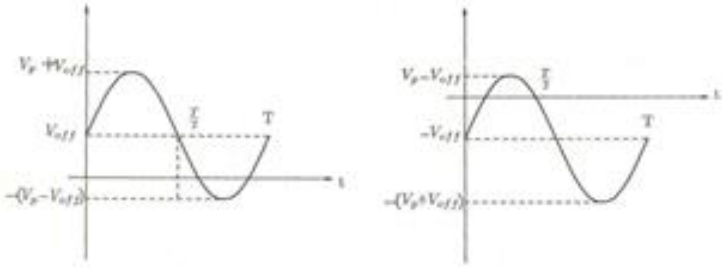
2.7 提升準位電路

又稱為箝位電路，功能就是將一個輸入訊號的平均值，作往上或往下的偏移，使其輸出的交流訊號以另一個新的平均值來輸送，波形中的頻率與振幅不會改變，只有輸入與輸出波形的平均值不同而已。



(a)箝位電路

圖 2.22 簡單的箝位電路和相對波形與時間



(b)輸出電壓往上偏移 V_{off} 量　(c)輸出電壓往下偏移 V_{off} 量

圖 2.23　　提升準位電路示意圖

當輸入 Vi 為負值時，二極體導通則電容經由二極體快速充電，最後電容會充到 Vp。

可得 $V_o = V_i + V_c$，即 $V_o = V_i + V_p$(圖 2.23)

V_i	V_C	$V_i + V_c = V_O$
V_p	V_p	$2V_p$
V_p	V_p	0

圖 2.24 電容中 V_0　V_i V_c的對應關係

以正弦信號為例(圖 2.24)：輸入為$V_i = V_m \sin(\omega t)$來分析該電路是如何箝位的。設電容的初始電壓$V_c(0) = 0$，二極體 D 是理想的。則當時間 t 由 0 時刻增至 T/4 時，V_i達到其峰值 V_m，電容的電壓也被充至峰值 V_m。隨之，V_i下降，很顯然，二極體處於反偏截至狀態，電容的電壓沒有地方放電，只能保持 V_m不變。因而可得輸出電壓$V_o = -V_c + V_i = -V_m + V_m \sin(\omega t)$。由此可見，輸出電壓被箝住了，輸出與輸入的波形相同，不同的只是輸出波形進行了-V_m的直流平移。

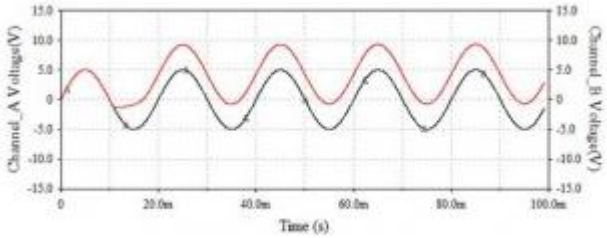


圖 2.25　黑色為原波形，紅色為提升過後波形

2.8 電源電路

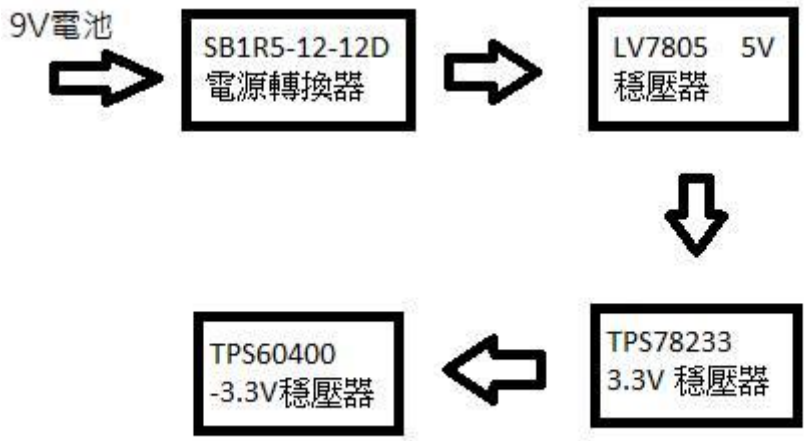


圖 2.26 電源流程

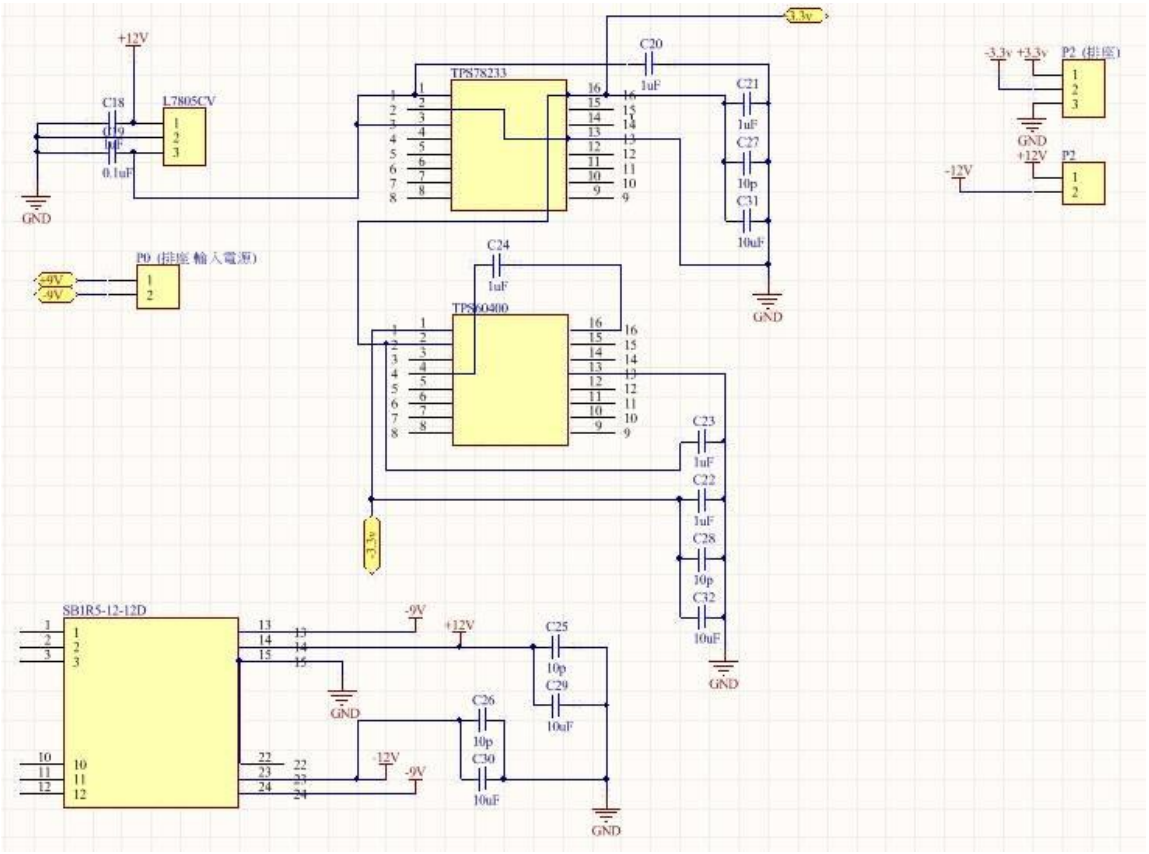


圖 2.27 電源電路圖

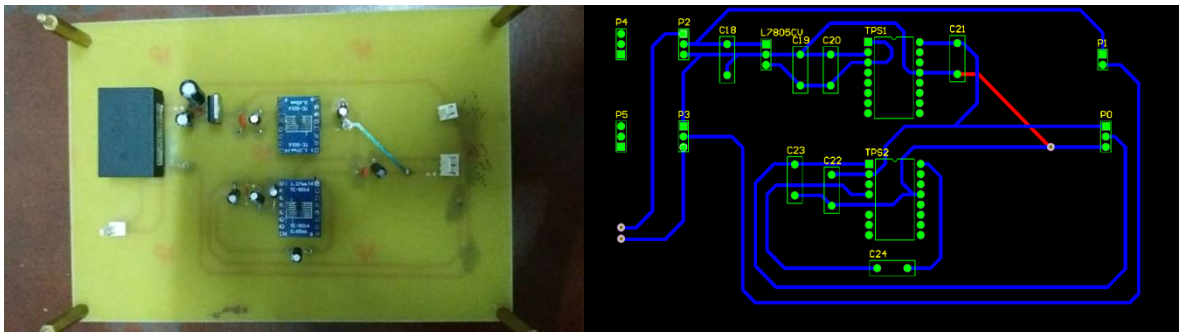


圖 2.28 電源電路板和電路圖

3. 數位系統

3.1 開發平台

我們所使用的開發平台為 IAR Systems，其為針對開發嵌入式系統為主的工作平台，圖 3.1 為 IAR 開啟畫面。此開發平台可針對 C、C++兩種程式語言進行編譯及 debug。它具有可以對 8 bit 及 16 bit 處理器 debugging 的 firmware，但現今大多針對較熱門的 16 及 32 bit 處理器進行開發。

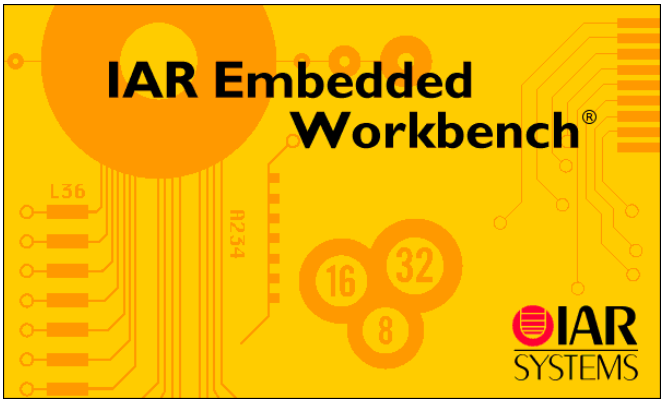


圖 3.1 IAR 啟動時畫面

3.2 IC 及開發板

我們所使用的 IC 型號是 MSP430G2553，生產商為德州儀器，圖 3.2 為搭配 G2553 的燒錄開發板，圖 3.3 為 G2553 腳位功能。德州儀器旗下有許多型號的 IC，各種 IC 都有不同特色及功能，當然價格也會隨著 IC 功能多寡而異，但它們都脫離不了嵌入式系統處理器的特色，就是低成本、客製化及低功耗，表 3.1 為 G2553 的規格表。G2553 相較於其他 G 系列的 IC 多出了類比轉數位的功能[3]。

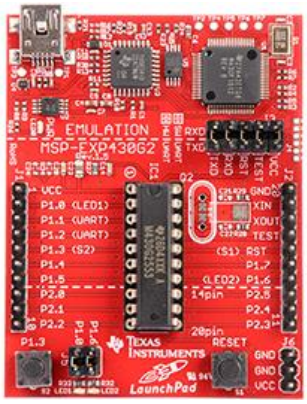


圖 3.2 G2553 開發板

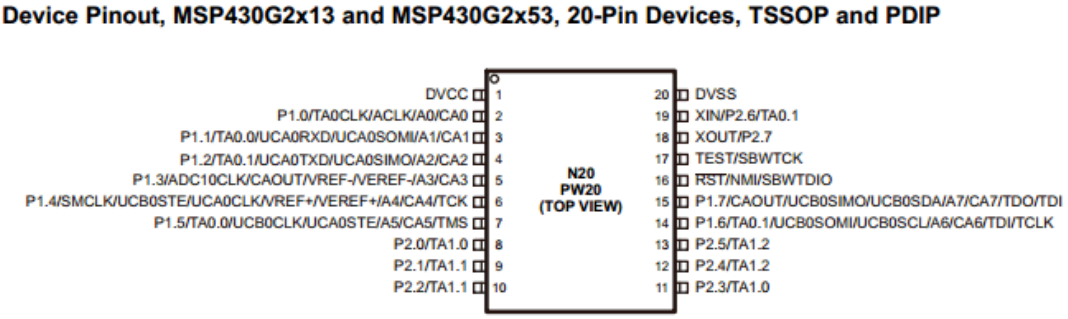


圖 3.3 G2553 腳位簡介

表 3.1 TI MSP430G2553 基本規格表

	MSP430G2553
CPU	MSP430
Frequency (MHz)	16
Non-volatile Memory (KB)	16
RAM (KB)	0.5
ADC	ADC10 - 8ch
Timers - 16-bit	2
Timers - 32-bit	0
Min VCC	1.8
Max VCC	3.6
Active Power (uA/MHz)	330
Standby Power (LPM3-uA)	0.7

3.3 程式基本設定

圖 3.4 為基本設定流程圖。首先將用不到的功能如看門狗關閉，將工作時脈設為 1MHz，接下來定義 IC 腳位的輸入輸出方向及功能，包括 ADC 輸入及七段顯示器輸出。將 P1.0 設為 ADC 輸入腳位，P1.4 至 P1.7 為七段顯示器控制腳位，P2.0 至 P2.3 為七段顯示器資料腳位，計時器設為上數模式[4][5]。

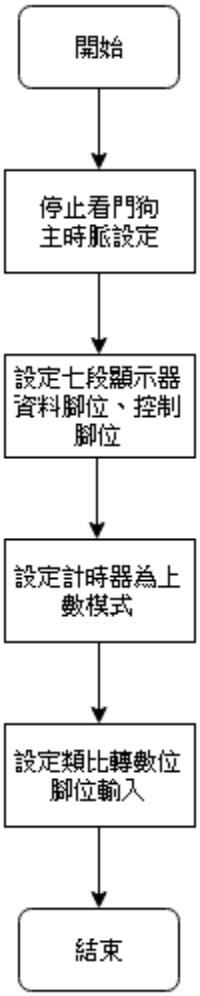


圖 3.4 基本功能設定流程圖

3.4 心率抓取演算法

圖 3.5 為心率週期抓取流程圖。當 ADC 開始讀入訊號時，若訊號小於 1.1 時則 index++，再繼續等到下次取樣，直到訊號大於 1.1 後才開始尋找 Peak 值。大於 1.1 後，若下一個取樣值相較之前任何值都高就將 index 設為零且執行 index++，並將此值存放，反之則只執行 index++，直到訊號低於 0.7 為止。當程式得到小於 0.7 的訊號時，就可以將 A 及 Peakvalue.index 相加以得到 Peak 到 Peak 的 index，也就是兩次脈搏間隔時間。

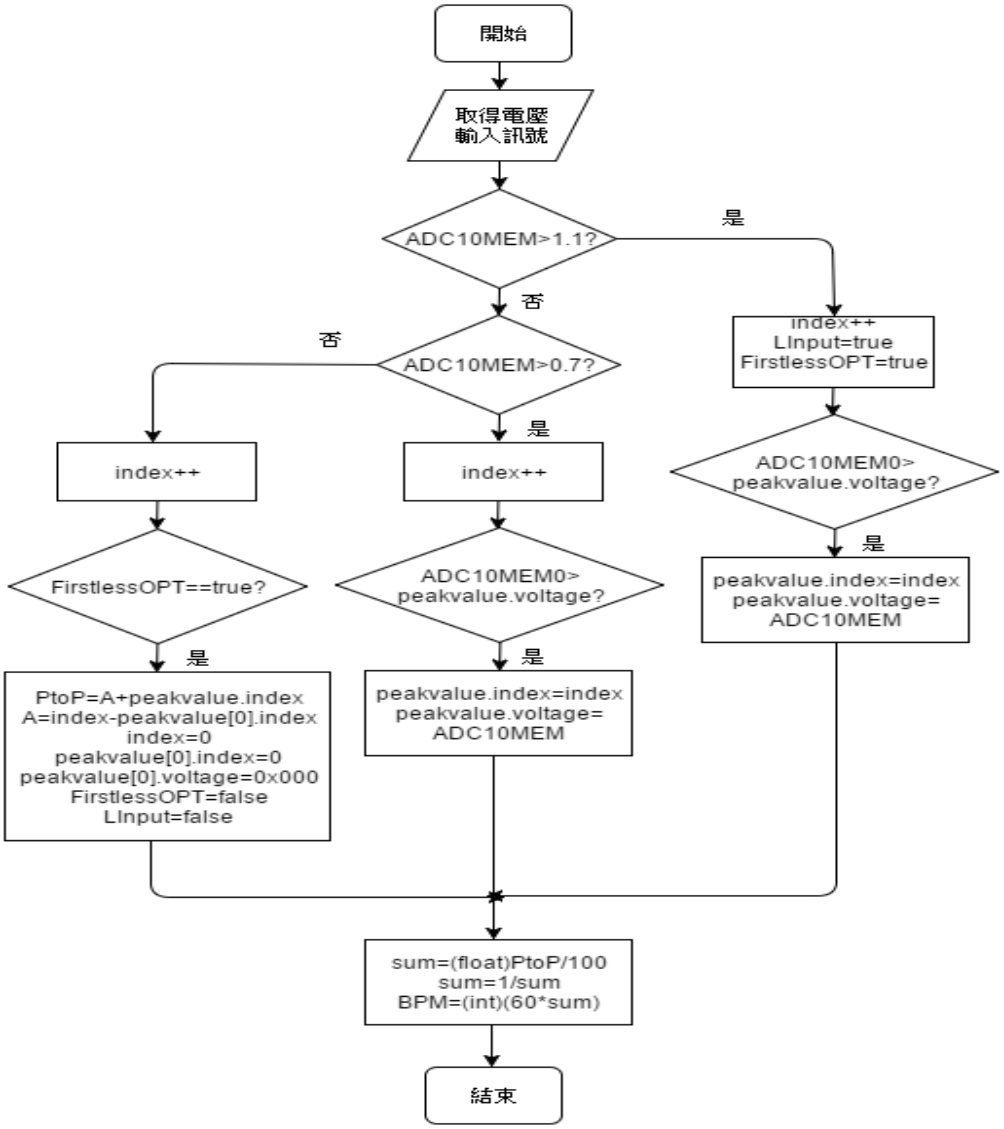


圖 3.5 心率週期抓取流程圖

圖 3.6 為心率訊號運算處理說明圖。從第一個波峰開始，只要訊號低於 0.7v 就將 A 的長度，也就是時間的長度儲存下來。接下來訊號若大於 1.1v 就開始偵測波峰值，直到訊號再次低於 0.7v，就可以確定 Peakvalue，相加 A 及 Peakvalue 就可以得到此次心跳周期。接下來的訊號也是重複上述動作算出周期。

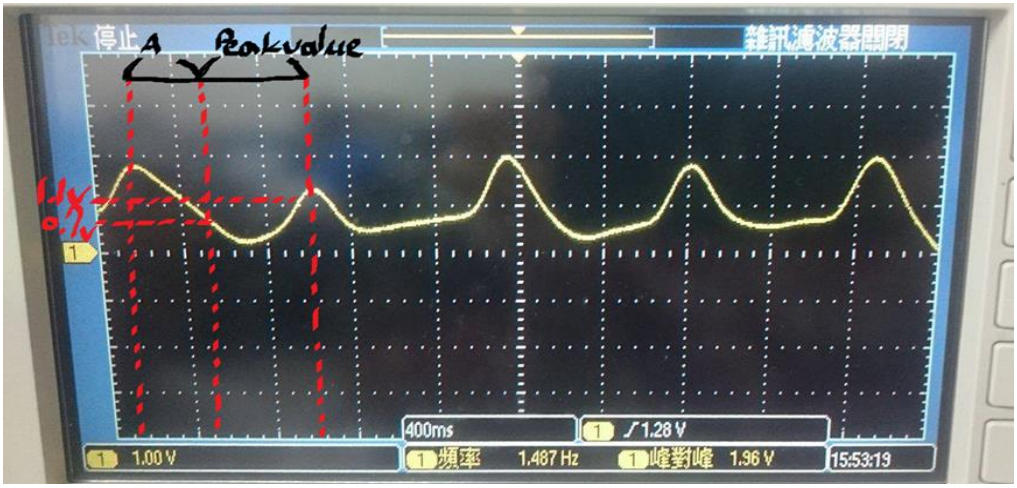


圖 3.6 心率訊號運算處理

圖 3.7 為陣列處理與結果運算流程圖。得到兩個 Peak 值之間的時間間隔後，再將 PtoP 除以 100 再取倒數，接著乘 60 即可求得以一個脈搏所算出的每分鐘心跳頻率。若連續五個數據誤差都在±20%內，就將其值存入 bpm_arr 陣列中。當 bpm_arr 陣列存至 10 個每分鐘心跳頻率數據時，就將 10 個數據取平均的結果顯示在七段顯示器。

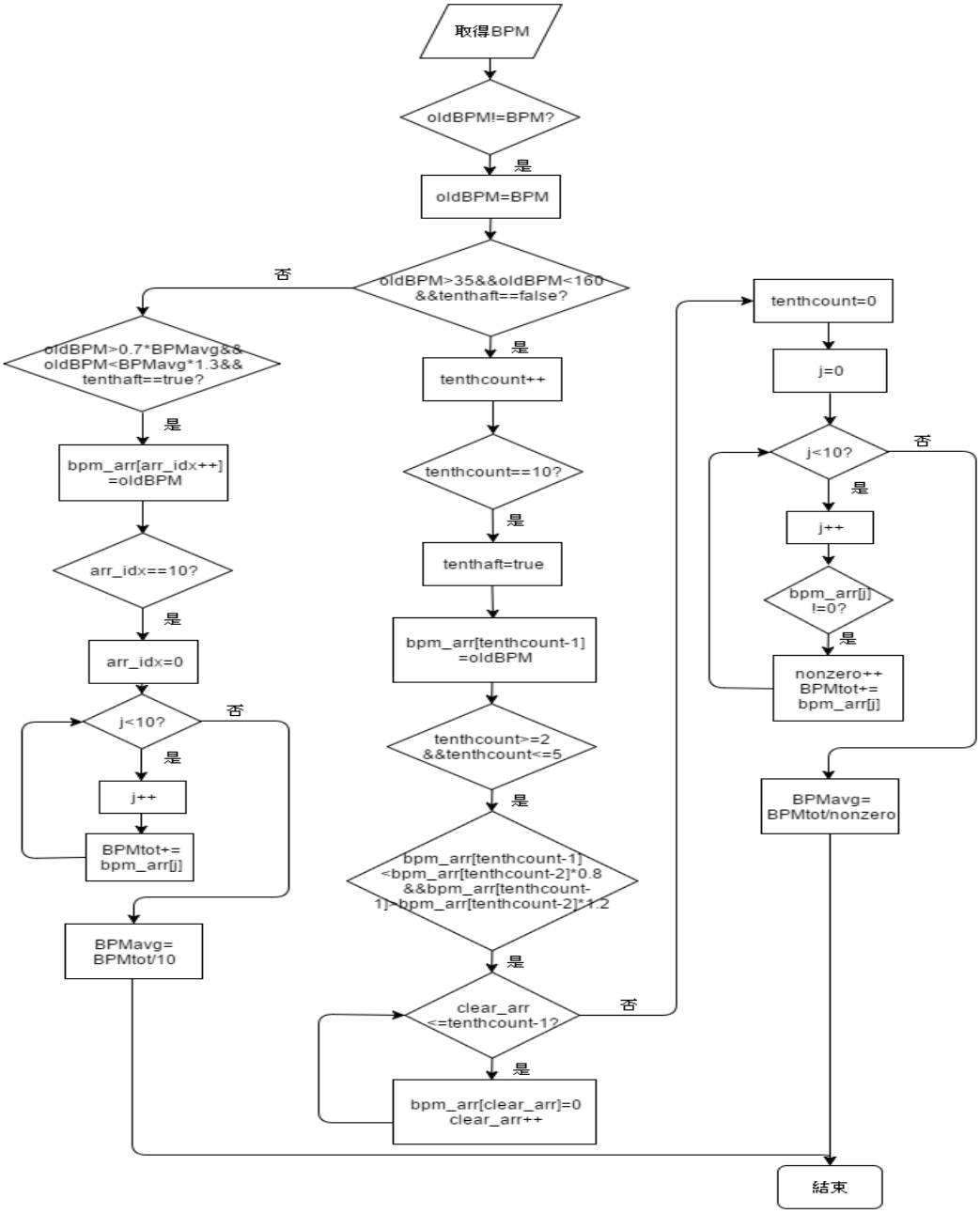


圖 3.7 陣列處理與結果運算流程圖

4. 實際測量和結果

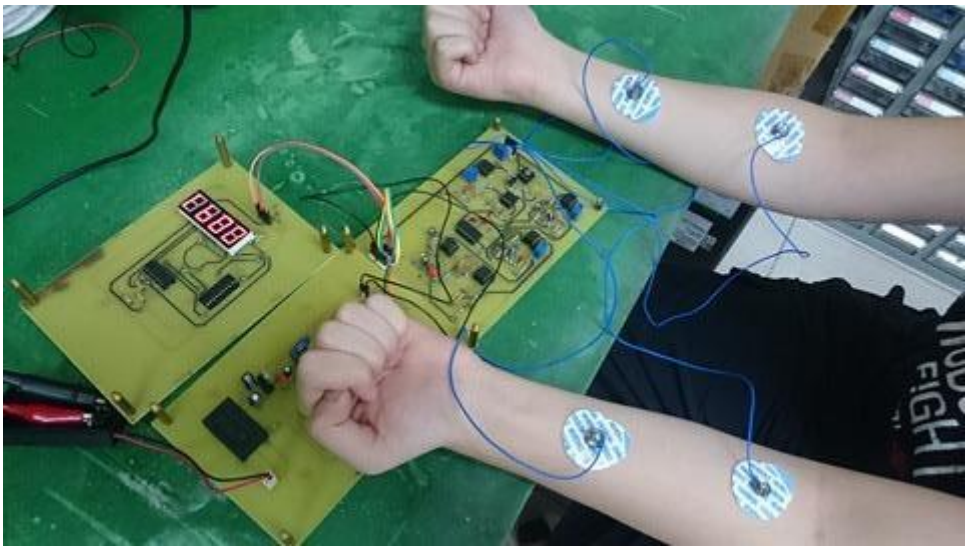


圖 4.1 人體測試

圖 4.1 為測量人體，測量貼片分別為定電流、高頻訊號、低頻訊號、接地，圖 4.2 為韋恩震盪器產生弦波，頻率 74 kHz，振福 6.9V，圖 4.3 為定電流電路輸出，輸出電流 0.3A，由圖 4.7 中即時心率的週期為 650ms，所以頻率為 1000/650=1.5384Hz/1 秒，因此即時心率 1 分鐘跳動為 1.5384*60=92.30 下，和圖 4.8 七段顯示器結果相符合。

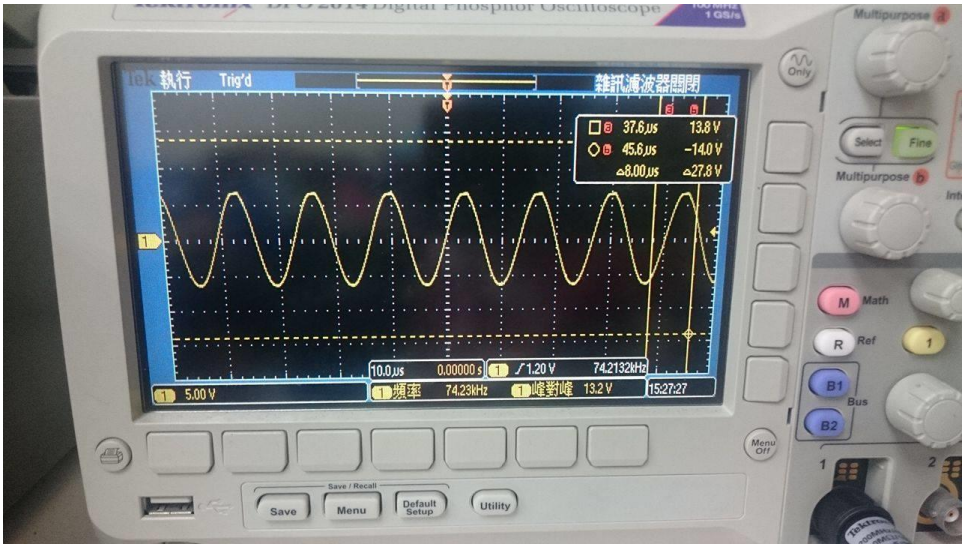


圖 4.2 韋恩震盪器所產生的正弦波

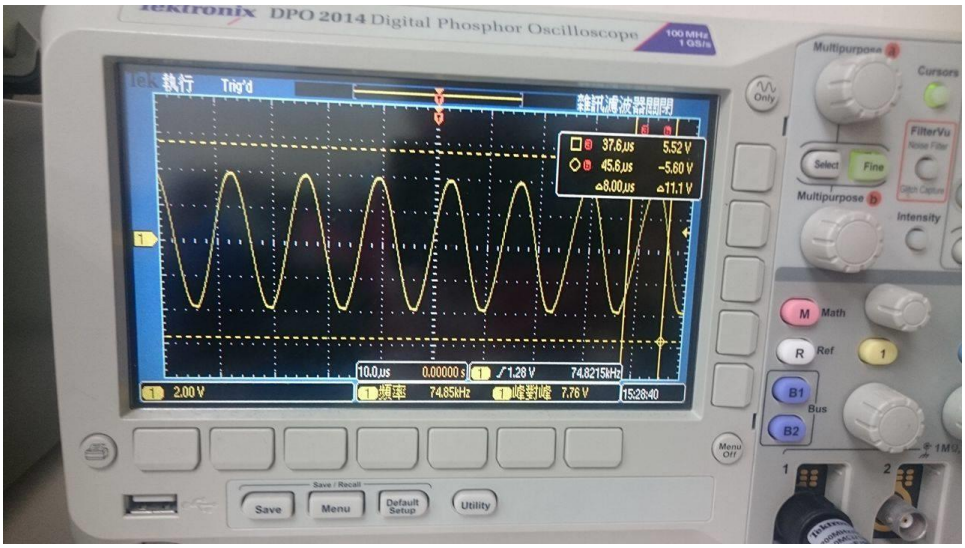


圖 4.3　確保電流穩定而放大過後的弦波(定電流)

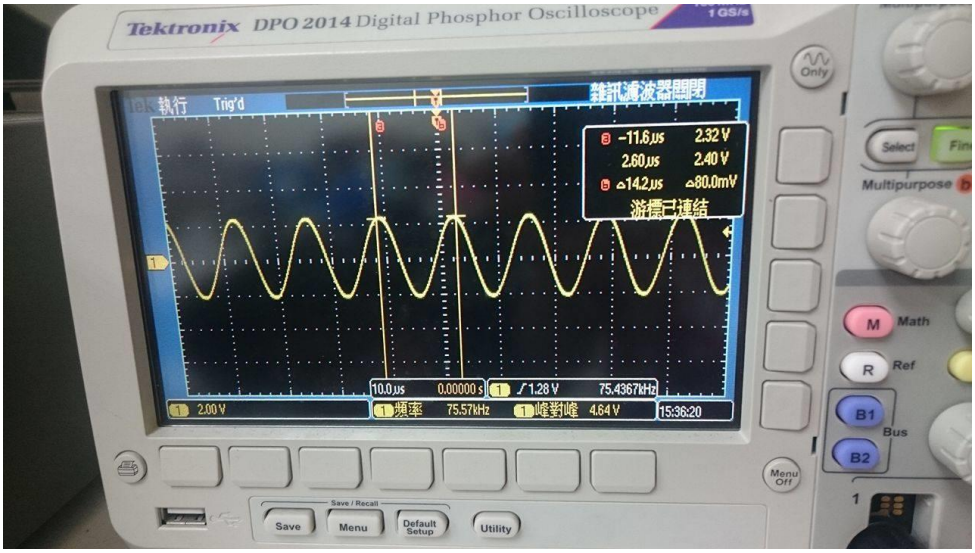


圖 4.4 Ad620 pin2 的波

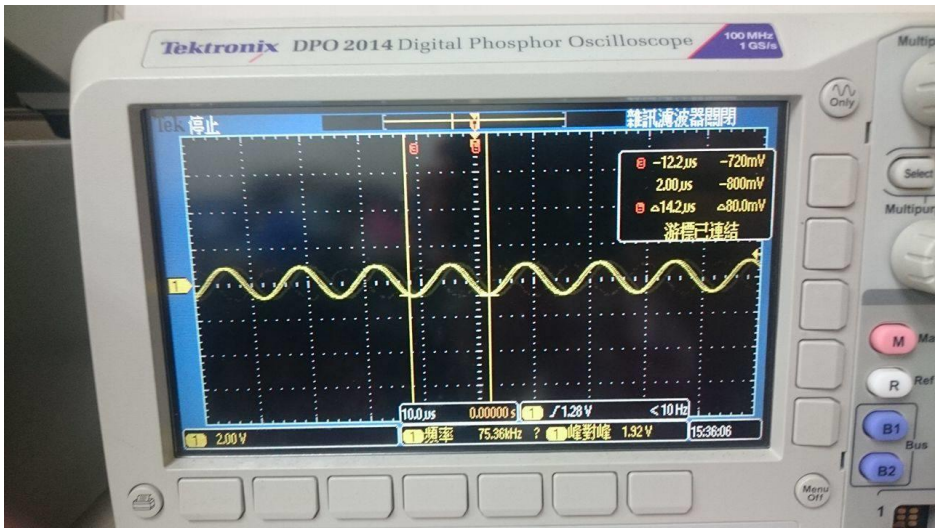


圖 4.5 Ad620 pin3 的波

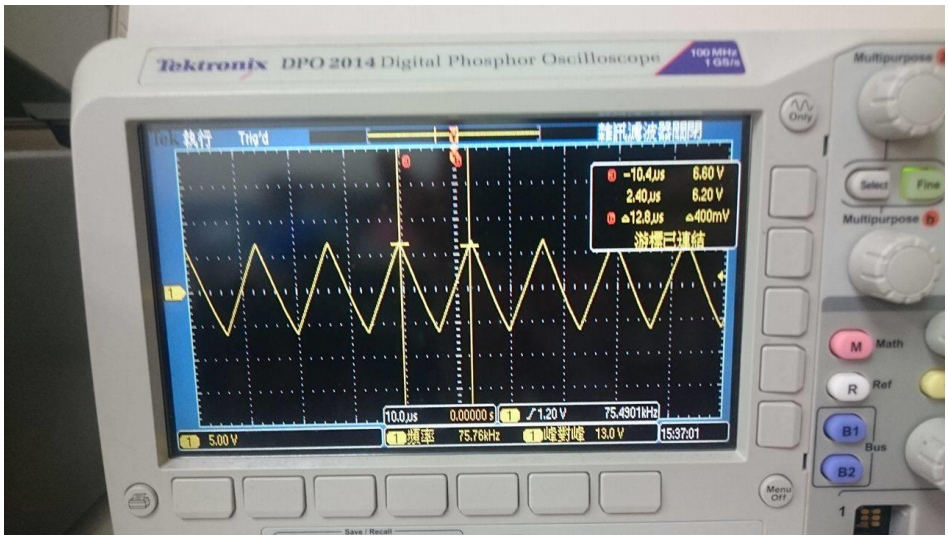


圖 4.6 儀表放大輸出的結果

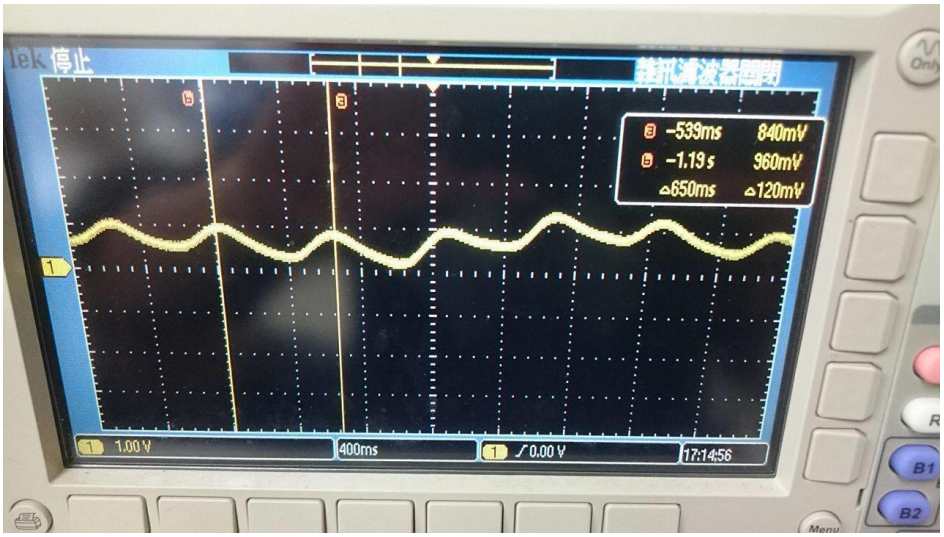


圖 4.7 即時心率

圖 4.8 七段顯示器顯示即時心率為 92

5. 結論與心得

在這一年多的專題學習過程中，相較於其他課程只有作業考試或報告，專題比較像是一項綜合的測驗。做專題的過程中必須先分工，等到每個人完成各自部分的事情後才能夠整合再一起，每個人都必須完成各自部分才能夠呈現出結果，缺一不可。我們的專題可分為電路板及程式兩部分，每個人也都有自己比較拿手的部分。雖然在做的過程中有碰到一些麻煩，但經過老師及學長的協助我們都可以一一解決。

雖然我們以經可以做到動態顯示每分鐘脈搏的次數，但還是有很多的方能夠加強。相較於市售的穿戴式脈搏監測的輕便性，我們板子的成品顯得過於累贅，穩定性也有加強的空間，顯示的部分也可以由七段顯示器改良成顯示於手機。若可以改變這些缺陷將可易讓這項成品更有看頭。

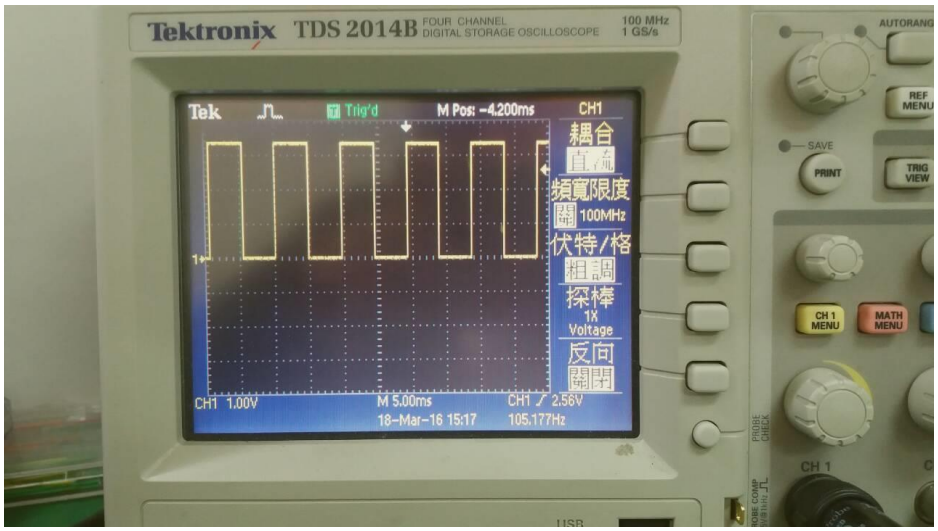
6. 致謝

專題製作過程中，老師及學長學姊們的適時幫助對我們有很大的影響，不論是程式或電路都幫我們很大的忙。感謝指導教授劉省宏老師及學長學姊們對於我們觀念上的導正與解決問題的辦法，在他們的幫助下我們更可以培養獨立解決問題的能力與對問題更深入的了解，對我們未來有非常大的幫助。

7. 參考文獻

[1] http://www.auto.fcu.edu.tw/wSite/publicfile/Attachment/f1255510699623.pdf
[2] http://ir.lib.ncu.edu.tw:88/thesis/getfile.asp?date=2008-7-23&file=7350955201129.pdf
[3] http://www.ti.com/lit/ug/slau144j/slau144j.pdf
[4] http://coder-tronics.com/msp430-adc-tutorial/
[5] http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/Cexamples/MSP430G2xx3%20Code%20Examples/C/

8. 附錄



ADC 取樣頻率

```
26
27  void main(void)
28  {
29    WDTCTL = WDTPW + WDTHOLD;      // Stop WDT
30    BCSCTL1 = CALBC1_1MHZ;         // Set range   DCOCTL = CALDCO_1MHZ;
31    BCSCTL2 &= ~(DIVS_3);          // SMCLK = DCO = 1MHz   //user guide P.284
32    P1SEL |= BIT0;              // ADC input pin P1.0
33
34    //*************************************************************************
35    //7seg setting
36    P1DIR  |= 0xF2;
37    P1OUT  =  0x00;
38    P2DIR  |= 0x0F;
39    P2OUT  =  0x00;
40    //*************************************************************************
41
42
43    //*************************************************************************
44    //timer setting
45    //P1DIR |= 0x02;                          // P1.0 output
46    CCTL0 = CCIE;                           // CCR0 interrupt enabled
47    CCR0 = 8050;  //8050
48    TACTL = TASSEL_2 + MC_2;                // SMCLK, contmode
49    //*************************************************************************
50
51    ConfigureAdc();           // ADC set-up function call
52    __enable_interrupt();         // Enable interrupts.
53    __bis_SR_register(GIE);   // Low Power Mode 0 with interrupts enabled
54  }
55
```

腳位基本設定

```
55
56  // Function containing ADC set-up
57  void ConfigureAdc(void)
58  {
59    ADC10CTL1 = INCH_0 + ADC10DIV_3 ;         // Channel 0, ADC10CLK/3
60    ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON + ADC10IE;  // Vcc & Vss as
61    ADC10AE0 |= BIT0;                         // ADC input enable P1.3
62  }
63
64  // Timer A0 interrupt service routine
65  #pragma vector=TIMER0_A0_VECTOR
66  __interrupt void Timer_A (void)
67  {
68    ADC10CTL0 |= ENC + ADC10SC;       // Sampling and conversion start
69    P1OUT ^= 0x02;                          // Toggle P1.0
70    CCR0 += 8050;                           // Add Offset to CCR
71  }
72
```

ADC 及 Timer 設定

```
82  void signal_processing(void)
83  {
84      //adc processing
85    int D4,D3,D2,D1;  int i,j,nonzero=0,BPMtot=0,clear_arr=0;
86    float sum;
87    static unsigned int A=0,PtoP=0,BPM=0,BPMavg=0,tenthcount=0, oldBPM=0;
88    static unsigned int index = 0,bpm_arr[10]={0},arr_idx=0;
89
90    if(ADC10MEM>0x155&&LInput==false){//大於1.1V
91      index++;
92      LInput=true;
93      FirstlessOPT=true;
94       if(ADC10MEM>peakvalue[0].voltage){
95         peakvalue[0].index=index;
96         peakvalue[0].voltage=ADC10MEM;
97      }
98    }
99
100   else if(ADC10MEM>0x0D9&&LInput==true){//大於0.7V
101      index++;
102      if(ADC10MEM>peakvalue[0].voltage){
103        peakvalue[0].index=index;
104        peakvalue[0].voltage=ADC10MEM;
105     }
106   }
107
108   else if(ADC10MEM>0x020){//0.15V<voltage<0.7V
109     index++;
110
111     if(FirstlessOPT==true){//第一次小於1.3
112       PtoP=A+peakvalue[0].index;
113       A=index-peakvalue[0].index;//1.3-peak的時間
114       index=0;
115       peakvalue[0].index=0;
116       peakvalue[0].voltage=0x000;
```

心率週期抓取設定

```
116         peakvalue[0].voltage=0x000;
117         //
118         FirstlessOPT=false;
119         LInput=false;
120         newbpm=true;
121       }
122
123     }
124     else{index++;}//小於0.3(no signal)
125
126
127      //輸出BPM
128       sum=(float)PtoP/100;
129       sum=1/sum;
130       BPM=(int)(60*sum);
131
132
133      //*****************************************************************  //HI
134      //前十個BPM(陣列滿前，有可能不是前十個)用正常值比較,接下來用20%比較
135
136       if(oldBPM!=BPM){//是新的BPM才執行
137         oldBPM=BPM;
138
139         if(oldBPM>35&&oldBPM<160&&tenthaft==false){
140           //決定判斷方法
141           tenthcount++;
142           if(tenthcount==10){
143             tenthaft=true;
144           }
145
146           //put into array
147           bpm_arr[tenthcount-1]=oldBPM;
148
149
150         //*****************************************************************
151         //如果前五個BPM值在20%誤差以上就把五個值全部丟掉(1vs2 2vs3 3vs4 4vs5)
```

陣列值比較判斷

```
150         //*****************************************************************
151         //如果前五個BPM值在20%誤差以上就把五個值全部丟掉(1vs2 2vs3 3v
152
153         if(tenthcount>=2&&tenthcount<=5){ //要有二個以上值才開始比較
154
155           if(bpm_arr[tenthcount-1]>bpm_arr[tenthcount-2]*0.8
156           &&bpm_arr[tenthcount-1]<bpm_arr[tenthcount-2]*1.2){}//最新
157
158           else{//超出20%誤差，清除陣列值
159             while(clear_arr<=tenthcount-1){
160               bpm_arr[clear_arr]=0;
161               clear_arr++;
162             }
163             tenthcount=0;
164           }
165         }
166
167         //*****************************************************************
168
169
170         //BPM avg setting
171         for(j=0;j<10;j++){
172           if(bpm_arr[j]!=0){//如果不是空值就讓divider++
173             nonzero++;
174             BPMtot+=bpm_arr[j];
175           }
176         }
177
178         BPMavg=BPMtot/nonzero;
179
180       }
181
182       else if(oldBPM>0.7*BPMavg&&oldBPM<BPMavg*1.3&&tenthaft==true){
183
184         //取代陣列最舊值
```

陣列存取設定

```
182     else if(oldBPM>0.7*BPMavg&&oldBPM<BPMavg*1.3&&tenthaft==true){
183
184       //取代陣列最舊值
185       bpm_arr[arr_idx++]=oldBPM;
186
187       if(arr_idx==10){
188         arr_idx=0;
189       }
190       //BPM avg setting
191       for(j=0;j<10;j++){
192         BPMtot+=bpm_arr[j];
193       }
194
195       BPMavg=BPMtot/10;
196
197     }
198
199     //oldBPM在範圍以外
200     else{}
201
202   }
203
```

陣列存取設定

```
204     //七段處理
205
206     if(tenthaft==true){   //開始正常顯示七
207       D4 =(BPMavg /1000)%10;
208       P2OUT = D4;
209       P1OUT = 0x80;
210
211       for(i = 0 ;i<DT;i++);
212
213       D3 = (BPMavg /100)%10;
214       P2OUT = D3;
215       P1OUT = 0x40;
216
217       for(i = 0 ;i<DT;i++);
218
219       D2 = (BPMavg /10)%10;
220       P2OUT = D2;
221       P1OUT = 0x20;
222
223       for(i = 0 ;i<DT;i++);
224
225       D1 = BPMavg %10;
226       P2OUT = D1;
227       P1OUT = 0x10;
228
229       for(i = 0 ;i<DT;i++);
230     }
231
```

七段顯示器設定