第二章 系統架構

2.1 系統單元

在本章，我們將介紹遙控車的硬體架構，其架構如圖 2.1 所示，可分為無線傳輸模組、控制模組、電源供應模組、及馬達驅動界面等四個模組。其中電源供應模組負責提供整個系統中各個電路所需要電源；無線傳輸模組則負責接收來自主控端電腦的命令，此命令即是用來控制遙控車行駛的方向及前進路線；而控制模組則是依循無線傳輸模組所接收到的路徑規劃，來控制遙控車的馬達依規劃路徑前進。在本章以下各節中我們將對以上四個模組做一詳細的介紹。

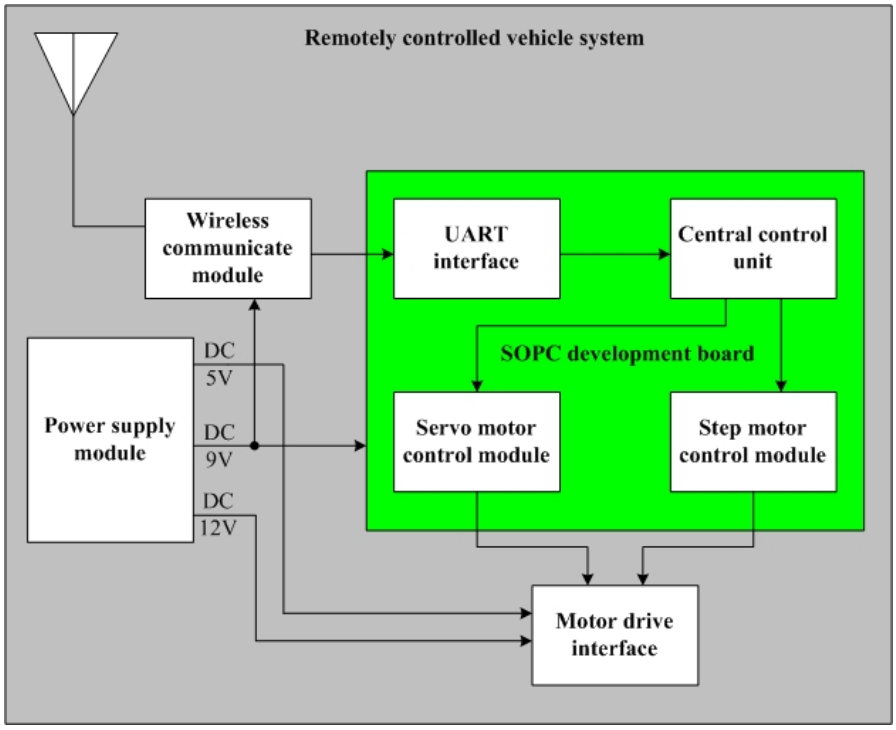


圖 2.1 遙控車之硬體架構

2.1.1 無線傳輸模組

無線傳輸模組主要是負責接收來自主控端電腦的路徑規劃，在我們的遙控車上採用了研廣科技股份有限公司的 RF3105 無線 RS232 收發模組(圖 2.2)。藉由此通訊介面來達到主控端電腦對遙控車控制指令的傳遞，其最大的傳輸距離在空曠地區可達到六佰公尺，在這個距離內的資料傳輸量最大可達 9600bps，因此此一無線傳輸模組用以遙控車在停車場模擬場內(模擬場大小約 5 公尺見方)傳輸資料可以符合我們所需的要求。而電源需求方面也只要供給 6 伏特到 10 伏特的電壓就可以穩定運作，在遙控車系統電力自給自足的情形下，它是適合運作在此一系統當中的。基於以上理由，我們選用此一通訊模組做為我們遙控車的傳輸介面。

圖 2.2 無線 RS-232 傳輸模組

2.1.2 電源供應模組

電源供應模組主要是提供遙控車上各硬體電路所需之電源，包括無線傳輸模組所需的 DC 9V、控制模組所需的 DC 9V、以及馬達驅動界面所需的 DC5V 及 DC12V。各硬體之電源需求如表 2.1 所示。

在此我們使用的是湯淺公司(YUASA Batteries Inc.)的 NP1.2-12 鉛酸電瓶(12 伏特，1.2 安培/小時)，見圖 2.3。由於其電池的封裝方式，不同於一般鉛酸電瓶不能倒置使用，因此可以自由的安置於遙控車上的任意位置。我們以此鉛酸電瓶所提供的 DC12V，配合 7805 及 7809 穩壓 IC 的穩壓以得到我們所需要的 DC5V 及 DC9V。

圖 2.3 湯淺公司-NP1.2-12 鉛酸電瓶

硬體名稱	需求電壓
伺服馬達	5V
馬達驅動界面	5V、12V
控制模組(SOPC 發展板)	9V
無線 RS-232 傳輸模組	9V
步進馬達	12V

表 2.1　各硬體電路之電壓需求

2.1.3 控制模組

控制模組主要負責將接收自主控端電腦的命令，轉換為馬達的驅動命令，以控制馬達的轉動及轉向，此模組包括：非同步串列接收界面(UART) 、中央控制電路、前輪方向控制、及後輪驅動控制。此控制模組部份是以 SOPC(System On a Programmable Chip)來設計，由於 SOPC 不但可以輕易的掛上 UART、及 CPU(Nios)，而且內部 FPGA 的架構可以讓我們規劃所需的硬體電路，如前輪方向控制及後輪的驅動控制等。

在本論文中我們即是以 SOPC 的 UART 做為無線傳輸模組的界面，而以 Nios 做為中央控制電路，將 UART 所接收到的路徑規劃命令，轉換為馬達的控制命令，以控制前輪馬達轉動(方向)或後輪馬達的轉動(前進或後退)。SOPC 的發展板如圖 2.4 所示，以下我們將再對控制模組中的前輪方向控制電路及後輪驅動控制電路做進一步的介紹。

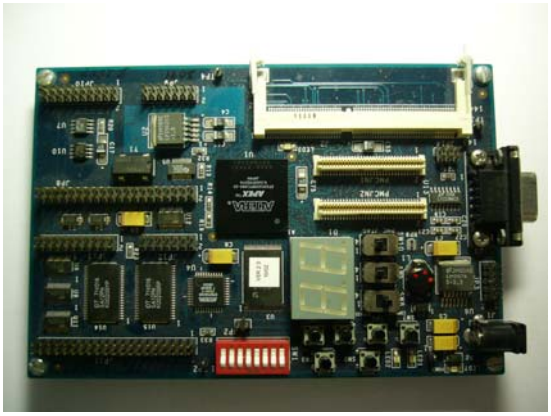


圖 2.4 SOPC 發展板實際外觀

2.1.3.1 前輪方向控制

遙控車的前輪主要是控制車輛的轉彎角度，在我們所設計的遙控車中前輪的轉彎是由一顆直流伺服馬達來控制，在此遙控車中所使用的伺服馬達是由日本 FUTABA 公司所出產的 S3003 型直流伺服馬達(DC Servo Motor)，如圖 2.5 所示。

圖 2.5 S3003 型 Servo Motor

欲對此類型馬達做控制，需先了解其控制方法。一般直流馬達的控制方式有兩種，分別為電壓調變、電流調變。電壓調變方式的特性是給予馬達的電壓差越大，則轉速越快，反之則越慢。而電流調變方式的特性則是給馬達的電流越大，則其輸出的力矩越大，即扭力越強，反之則越弱。不過我們所使用的 SOPC 技術只有數位訊號輸出，因此要產生不同電壓差的類比訊號，則需使用脈波寬度調變(pulse

width modulation, PWM)的方式，它是利用對輸入馬達脈波寬度的不同來控制馬達的旋轉。

此一伺服馬達在經實驗後，得知週期為 20ms 的情形下，其脈波寬度在 0.50ms 到 2.50ms 間即可控制馬達由左轉 45 度角到右邊 45 度角(如圖 2.6)，由此得知每 0.01ms 可使馬達旋轉 0.9 度。

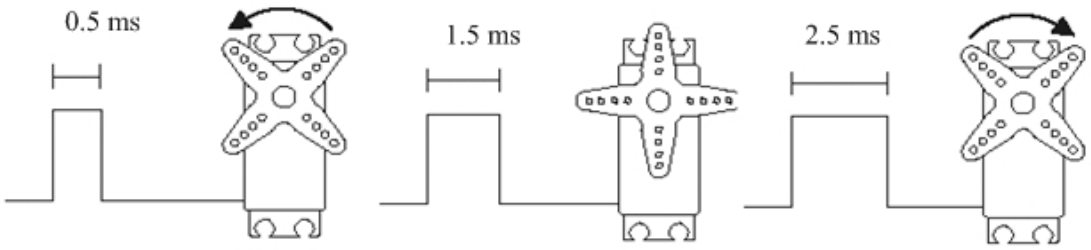


圖 2.6 脈波寬度不同的馬達轉向

因此我們設計一個 PWM 電路來控制此一伺服馬達。圖 2.7 為 PWM 電路之方塊圖，為一可程式規劃(Programmable)的 PWM 電路，可以藉由一 8 位元的位元組輸入，來產生所需的脈波寬度。其中輸入轉換器的功能是將位元組輸入做範圍上的轉換，將輸入 0 到 256 的範圍轉換成 0 到 200。

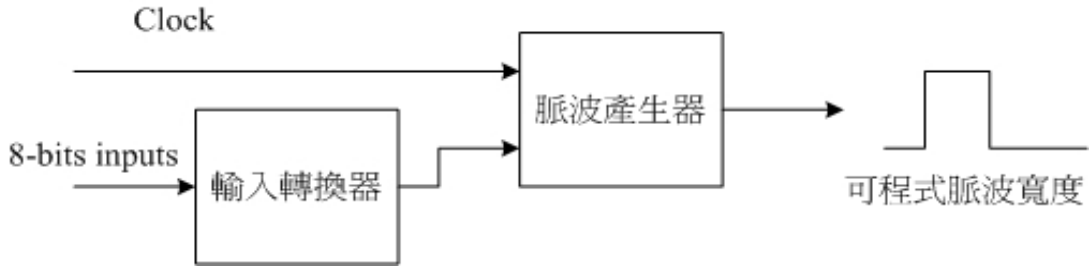


圖 2.7 PWM 電路

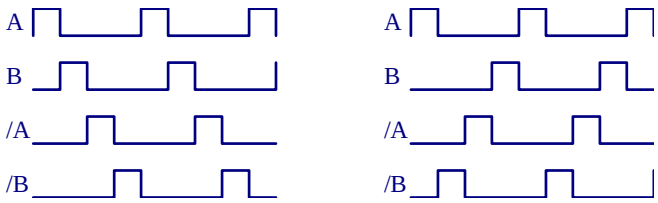
2.1.3.2 後輪驅動控制

遙控車的後輪是用來驅動車子前進或後退，為了準確的控制車子前進或後退的距離，我們以步進馬達做為後輪的驅動馬達，在此我們使用了日本 Minebea 公司的 17PM 型步進馬達(Step Motor，如圖 2.8), 為一二相步進馬達。

圖 2.8 17PM 型步進馬達

在步進馬達控制電路包含了供應電壓的電力控制電路和使馬達轉動激磁循序電路。二相步進馬達的激磁方式有幾種方式：一相激磁方式、二相激磁方式、一-二相激磁方式。下面對這幾種激磁方式做簡單的介紹。

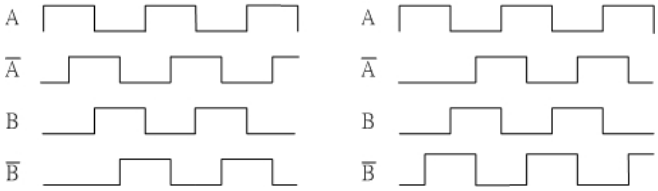
(1) 一相激磁方式：於每次驅動馬達時，只對馬達中的一組線圈產生激磁，它的優點是較為省電而且在角度轉動的精確度良好，可是因為一相激磁的關係，因此產生力矩較小的缺點。其控制波形產生如圖 2.9。



(a). 馬達正轉　　(b). 馬達反轉

圖 2.9 步進馬達一相激磁方式

(2) 二相激磁方式：在每次驅動馬達時，同時對馬達中兩組線圈產生激磁，與一相激磁方式相比較，會有耗電量較大的缺點及散熱的問題，但也由於同時有兩相位同時產生激磁，因此能產生較大的力矩。這種激磁方式和一相激方式在每次激磁皆轉動相同角度。其控制波形產生如圖 2.10。

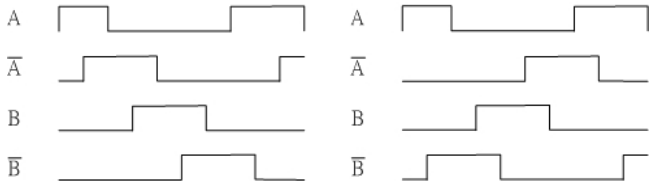


(a). 馬達正轉　　(b). 馬達反轉

圖 2.10 步進馬達二相激磁方式

(3) 一-二相激磁方式：此一步進馬達控制方式是以一相激磁、二相激磁兩種方式輪流驅動馬達。因此其特點為一相及二相激磁方式的綜合體。而此種方式的最大不同點是每次激磁馬達轉動角為前兩種激磁方式的一半。因此

在此控制方法可做較細微的步進控制。其控制波形產生如圖 2.11。



(a). 馬達正轉　　(b). 馬達反轉

圖 2.11 步進馬達一-二相激磁方式

在了解步進馬達的激磁方式後，我們是選擇以一相激磁的方式來對步進馬達做驅動，是由於其較為省電且其產生的力矩以足夠克服遙控車所遇到的阻力。圖 2.12 為後輪驅動控制電路方塊圖。

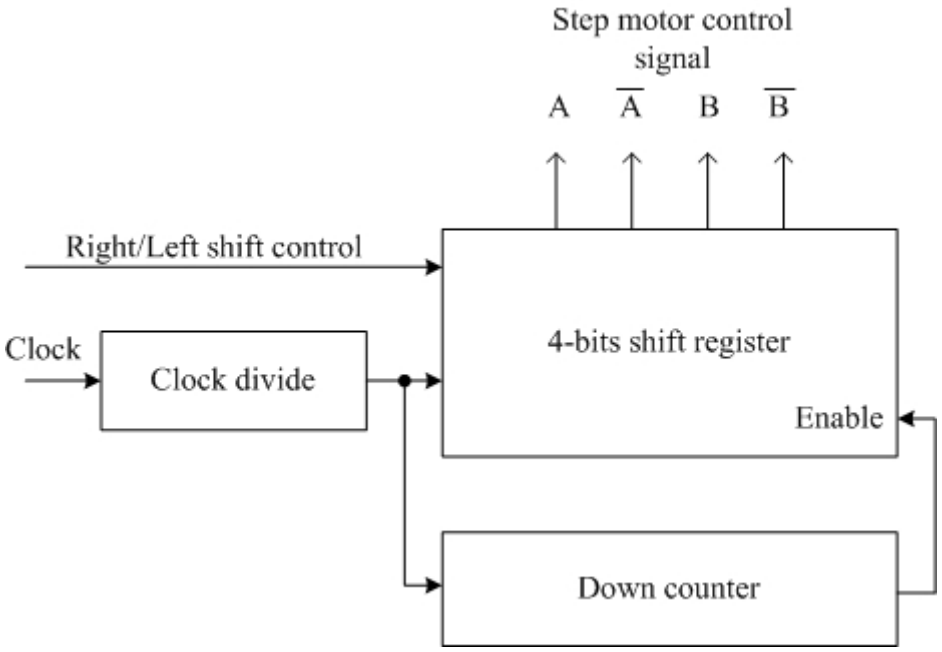


圖 2.12 步進馬達控制電路方塊圖

此電路是由一 4-位元移位暫存器(4-bits Shift Register)、除頻器(divider)、及計數器(counter)所組成。4-位元移位暫存器用來產生步進馬達所需的相位；藉由 Right/left 的控制可以控制馬達正轉或反轉；而除頻器則用以控制移位暫存器的 clock 速度，藉以控制馬達轉動的速度；最後計數器用來控制馬達轉動的步數，藉以控制車子前進或後退的距離。最後完成的電路圖如圖 2.13

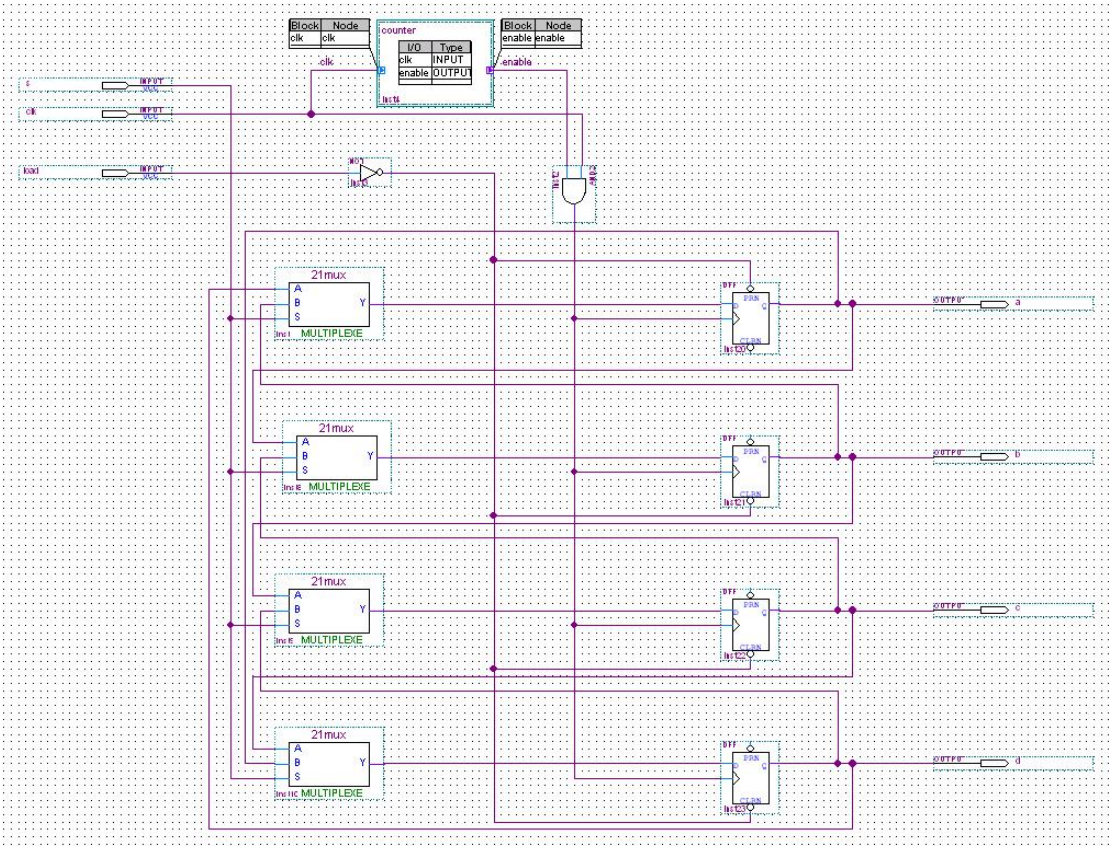


圖 2.13 步進馬達實際驅動電路

2.1.4 馬達驅動界面

由於馬達的驅動電路是以 SOPC 設計，而且一般的步進馬達需要以 DC 12V 來驅動，為了能順利的驅動馬達及防止馬達的驅動對 SOPC 的影響，我們另外設計了一個馬達驅動電路如圖 2.14 所示，其中 U1 是當 SOPC 與馬達間的緩衝器，而 U2 則為步進馬達的驅動器。

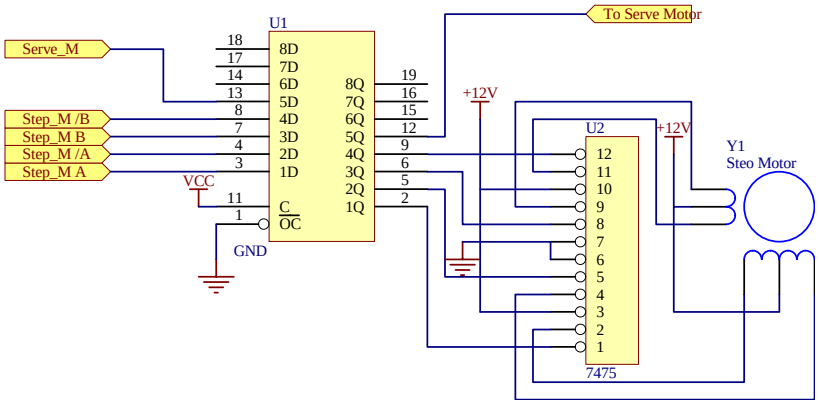


圖 2.14 馬達驅動界面

最後我們將祐盈企業有限公司所製造的四輪模型車輛(其車體規格如表 2.2 所示)加以改裝，車體外觀如圖 2.15 所示。我們將原本模型車上的燃油引擎置換成欲做為實驗用途的步進馬達，用以控制車輛後輪進前後退，配合上面所提的硬體電路，組成我們所要的遙控車，如圖 2.16 所示。

車輛參數	
Length approx.	330 mm
Width approx.	160 mm
Height approx.	130 mm
Wheelbase approx.	205 mm
Front track approx.	135 mm
Rear track approx	135 mm
Wheel diameter approx.	55 mm
Weight ready to run approx.	1310 g
Scale	1:10

表 2.2 實驗用模型車輛諸元

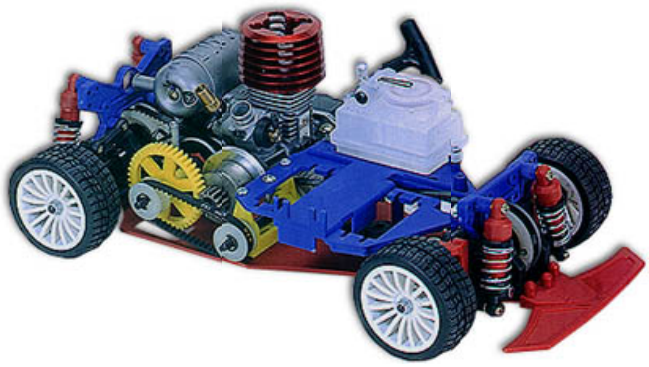


圖 2.15 模型車外觀

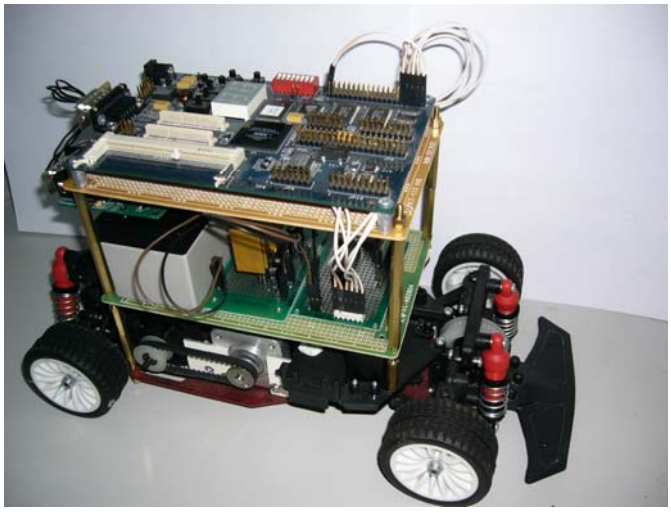


圖 2.16 改裝後遙控車的外觀

2.2 SOPC 系統設計

此一小節將對 SOPC 系統的軟硬體規劃、設計流程及運作方式作介紹，且說明如何將遙控車各模組、單元整合至 SOPC 系統中。

SOPC 系統是由美商 Altera 所發展的可程式化系統晶片。它能將軟、硬體設計快速的整合至單一 FPGA 晶片中。這種晶片設計方式具有可程式化，靈活度高、較低設計成本的優點，而且它可以減少類比零件的使用，此外由於套件中附有許多電子自動化設計軟體(electronic design automation, EDA)，因此可以大幅縮短系統設計時間。

2.2.1 SOPC 發展板簡介

用於此一遙控車控制核心的 FPGA 晶片為 APEX20K200E。SOPC 發展板如圖上除了此一 FPGA 晶片外，還有其它硬體資源可供利用(實際硬體資源分配如圖 2.17)：

- 1Mb 的快閃記憶體(Flash memory)，以存放硬體規劃及軟體設計
- 256Kb 靜態隨機存取記憶體(static random access memory, SRAM)，做為暫時的軟體測試，此外也可提供給 Nios 處理器使用
- 3.3V 及 5V 接腳各 40 根，提供使用者自行定義連結其它裝置
- 一組 RS-232 序列連接器給使用者定義及用來做為下載軟體設計
- 一組 JTAG(joing test action group)連接器用以下載硬體設計

- 一組相容於 SDRAM 模組的 DIMM 插槽、及兩組 IEEE-1386 PCI 連接器(peripheral component interconnect connector)
- 一組 8 位元 DIP 開關及四個可使用者定義的按鈕開關
- 兩組七段顯示器及兩個可使用者定義的 LED 燈
- 內建震盪器及零時差(zero-skew)時鐘

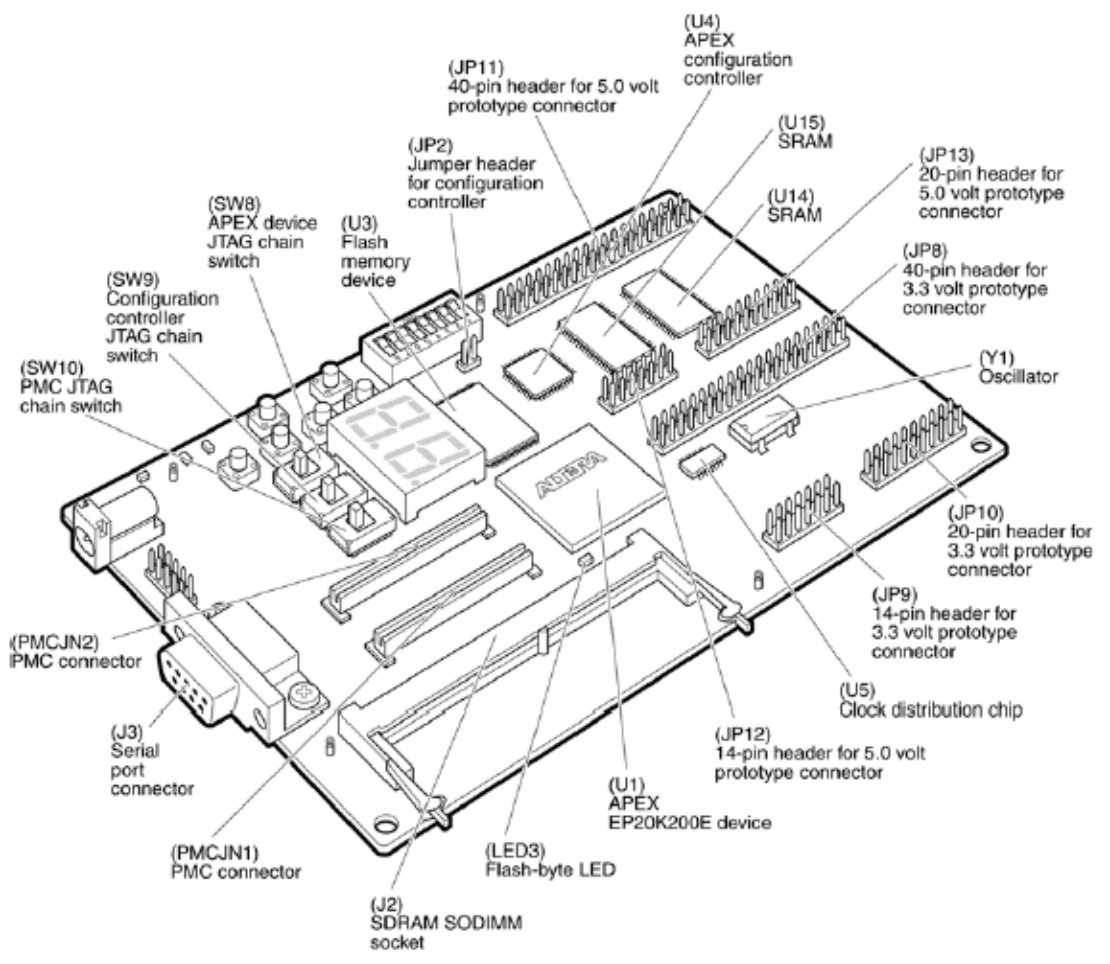


圖 2.17 SOPC 發展板各元

2.2.2 SOPC 設計流程

SOPC 發展大抵是以下面流程進行開發，流程圖如圖 2.18。

1. 系統設計需求：在系統一開始設計所需系統前，需先對系統所需硬體及效能定出要求，如遙控車系統時鐘、傳輸介面所需的 UART 傳輸格式、馬達與 SOPC 發展連接 pin 腳位的需求及定義等。

2. 軟硬體設計

2.1 Nios 處理器及週邊規劃：以 SOPC Builder 進行規劃。在這階段會產生硬體設計所需的 HDL 檔以及軟體發展所需使用的 header 檔。

2.2 硬體規劃：此步驟是使用 EDA 工具-Quartus II 來進行設計。它可使用 HDL 語言來對欲設計的硬體進行設計，也可以以方塊圖的形式來設計硬體。

2.3 軟體設計：使用 GNUPro 發展工具進行系統軟體方面的開發。且在步驟 2.1 產生的 header 檔，在此可用來做為 library 使用。

3. 電路模擬：在此步驟將對設計的電路進行模擬，使用 ModelSim 模擬工具。在模擬過程中如有不符要求的地方，則再回到軟硬體設計重新設計，且重覆此步驟，至模擬部份達到要求。

4. 電路驗證：接著便可將整個系統的軟硬體設計下載至 SOPC 發展板上進行實際的電路驗證。

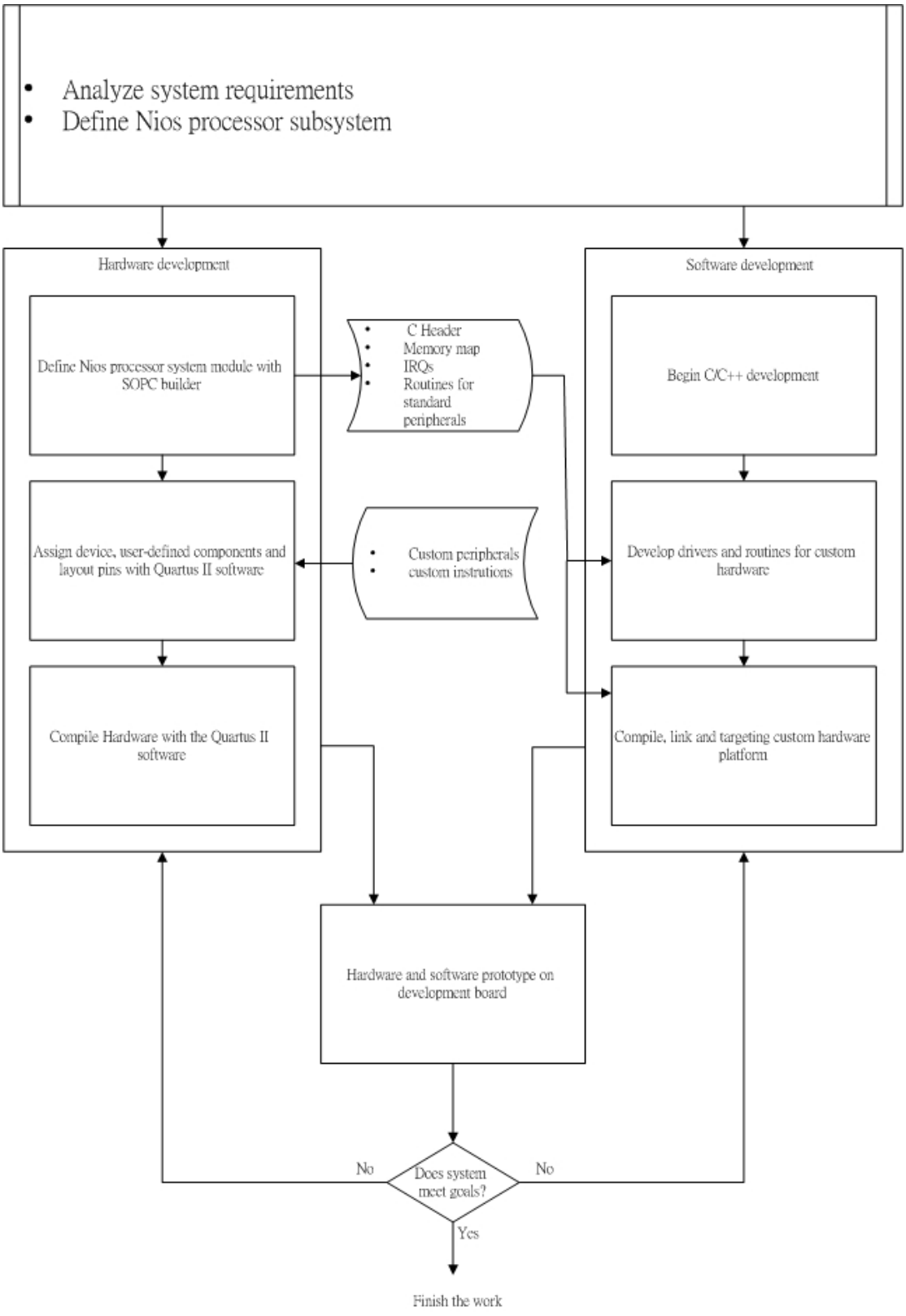


圖 2.18 SOPC 設計流程圖

2.2.3 硬體設計

本小節將說明 SOPC 中的硬體設計。

2.2.3.1 Nios 處理器及週邊規劃

在這裡是使用 SOPC Builder 這套軟體來規劃 SOPC 發展板上的軟核心-Nios 及其硬體資源(如記憶體、區段顯示器、按鈕等…)。SOPC Builder 為一 GUI 的發展工具，它可依照系統開發者定義完成處理器與週邊的連結。圖 2.19 為遙控車系統於 SOPC Builder 內相關周邊規劃。在這裡除了系統基本運作所需的硬體外，遙控車需要 UART 來做無線模組的傳輸介面，及兩組 PIO(parallel I/O)作為遙控車兩顆馬達的控制腳位。

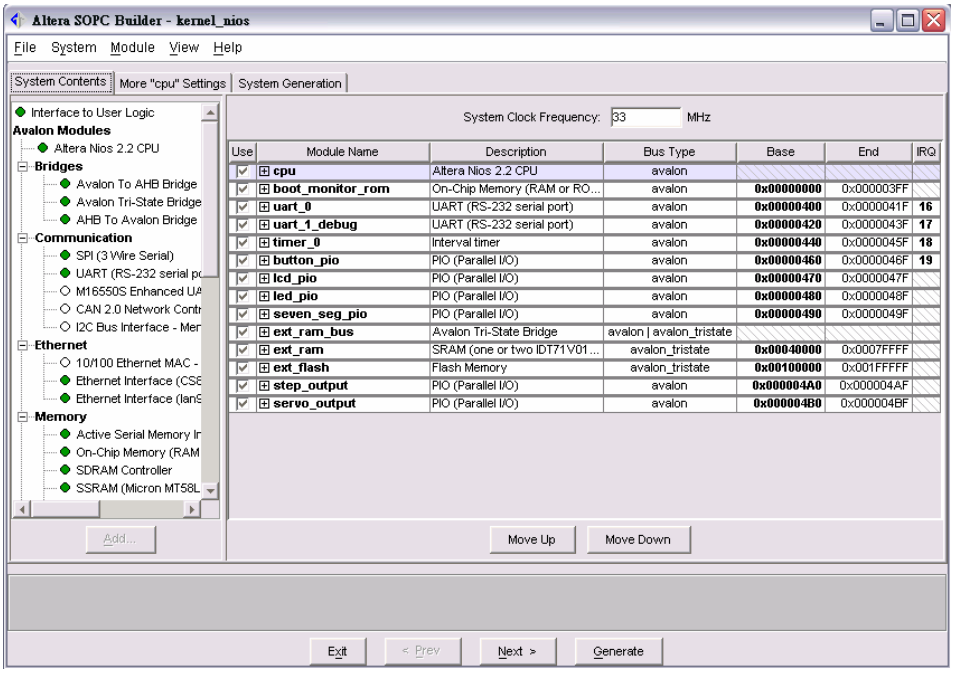


圖 2.19 SOPC Builder

2.2.3.2 硬體規劃

這部份包含了遙控車馬達控制元件的設計、電路方塊圖對映至 SOPC 發展板的腳位輸出定義。在前面 SOPC Builder 所產生的的系統核心以及 2.1.3 節所提到的馬達控制電路也在這裡進行電路連結。圖 2.20 為遙控車硬體設計電路連結圖。

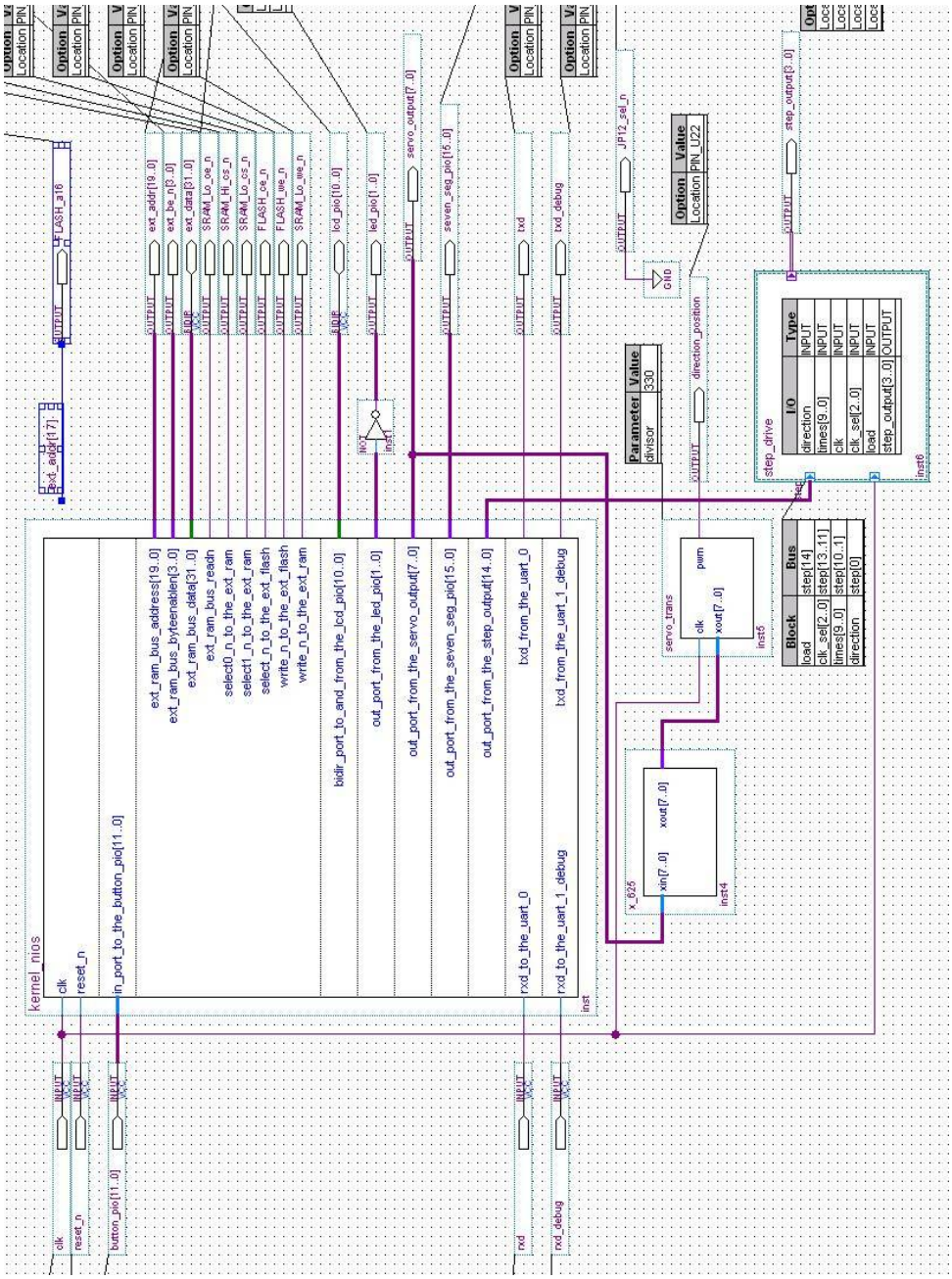


圖 2.20 遙控車硬體設計電路連結圖

2.2.4 軟體設計

此小節說明 SOPC 的軟體設計

2.2.4.1 C/C++軟體設計

在以 SOPC Builder 建立系統核心後，會產生一 excalibur.h 的標頭檔，此檔中包含了在 SOPC Builder 中設定含入系統中的硬體在記憶體的對映位址。如圖 2.21

```
#define na_uart_0_base                          0x00000400
#define na_uart_0_irq                           16
#define na_step_output          ((np_pio *)     0x000004a0) // altera_avalon_pio
#define na_step_output_base                     0x000004a0
#define na_servo_output         ((np_pio *)     0x000004b0) // altera_avalon_pio
#define na_servo_output_base                    0x000004b0
```

圖 2.21 Excalibur.h 硬體記憶體對映

接著既可以指標形態的變數存取方式對硬體做控制。如圖 2.22

```
np_pio *mystep = na_step_output;
np_pio *myservo = na_servo_output;
```

圖 2.22 以軟體控制硬體裝置

2.2.4.2 伺服馬達軟體控制

在前面我們已經定義 myservo 為控制伺服馬達控制訊號接腳的輸出，此為一八位元的資料型態。我們可以根據式子 2.1，藉由輸入 myservo 的不同來控制輸出脈波寬度在 0.50ms 到 2.50ms 之間。

$$0.5ms + \frac{myservo}{255} \times 2.00ms \qquad (2.1)$$

2.2.4.3 步進馬達軟體控制

步進馬達是由一組 15 位元的 PIO 訊號來控制，其格式如下表：

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
load	clk_sel			Times										Direction

表 2.3 步進馬達控制格式

Load：即一開始使位移暫存器有 1000 的初值。

Clk_sel：可選擇輸入步進馬達的脈波頻率，可改變馬達的轉動速率

Times：可設定輸入步進馬達的脈波次數，因此可決定馬達的轉動步數

Direction：控制步進馬達轉動方向的訊號

2.2.4.4 中斷副程式

在遙控車中需要中斷程式來進行控制。當遠端主控電腦對遙控車傳送指令時，遙控車需產生一中斷並停止目前動作，執行主控電腦傳送的指令。圖 2.23 為 nios 中斷處理程序

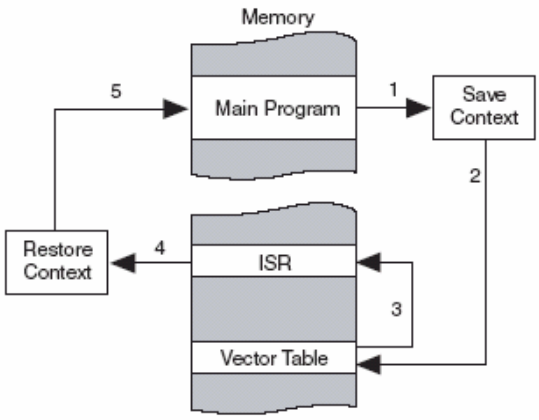


圖 2.23 中斷處理程序(http://www.altera.com)

1. 當接收到中斷訊號時，Nios 會先儲存目前系統狀態

2. 接著在系統向量表(vector table)中取出中斷號碼對映的中斷程式位址

3. 接著跳至中斷程式存放的記憶體位址執行

4. 中斷程式執行完畢後，再回存系統狀態接著繼續中斷前的系統運作

此一中斷程序於 Nios SDK(software develop kit)有提供副程式(如圖 2.24)供系統開發者使用。

```
void nr_installuserisr(int trapNumber,
     void *nios_isrhandlerproc, int context);
```

圖 2.24 Nios 提供之中斷副程式

trapNumber 是以 SOPC Builder 進行設計時所設定的中斷號碼，nios_isrhandlerproc 則為中斷發生時執行的程式，而 context 則是做為中斷傳遞資料的用途。

在此遙控車系統中，我們欲以 UART 介面接收到訊號時，做為中斷產生的事件，進而使遙控車停止目前的動作以執行 UART 介面所接收的指令。圖 2.25 為遙控車內之中斷程式碼

```
void isr(int context)
{
        int status;
        int rxChar;
        char s[256];

        status = na_uart_1->np_uartstatus;
        rxChar = na_uart_1->np_uartrxdata;

        na_uart_1->np_uartstatus = 0;
        if(status & np_uartstatus_rrdy_mask)
        {
                if(((rxChar <= 57) && (rxChar >= 48)) || ((rxChar <= 122) && (rxChar >= 97)))
                {
                        sprintf(s,"Your command is %c", rxChar);
                        nr_pio_lcdwritescreen(s);
                }
                else
                        nr_pio_lcdwritescreen("Error!!!        WRONG COMMAND!!!");

        switch(na_uart_1->np_uartrxdata) {
                        .
                        .               //轉向指令
                        .
                }
        }
}
int main(void)
{
                .
                .
                .
        nr_installuserisr(na_uart_0_irq, isr, context);
        na_uart_1->np_uartcontrol = np_uartcontrol_irrdy_mask;
                .
                .
                .
}
```

圖 2.25 遙控車 ISR 程式