第三章 嵌入式數位訊號處理系統工作原理

3.1 訊號處理流程

本論文所發展系統的訊號處理流程，主要是透過類比/數位轉換器(A/D converter, ADC)擷取輸入的類比訊號轉換成數位資料，再靠著CPLD(complex programmable logical devices)的控制，將 16 組數位資料儲存於靜態隨機存取記憶體(static random access memory, SRAM)中，等待 DSP(digital signal processor)的讀取。經過 DSP 運算之後，輸出到具有脈寬調變(pulse width modulation, PWM)功能的 CPLD，以產生切換訊號控制外部的電力電子開關。為了簡化描述，將其相關元件以圖 3.1 中的簡稱來表示，詳細的處理流程可以分成以下四個步驟：

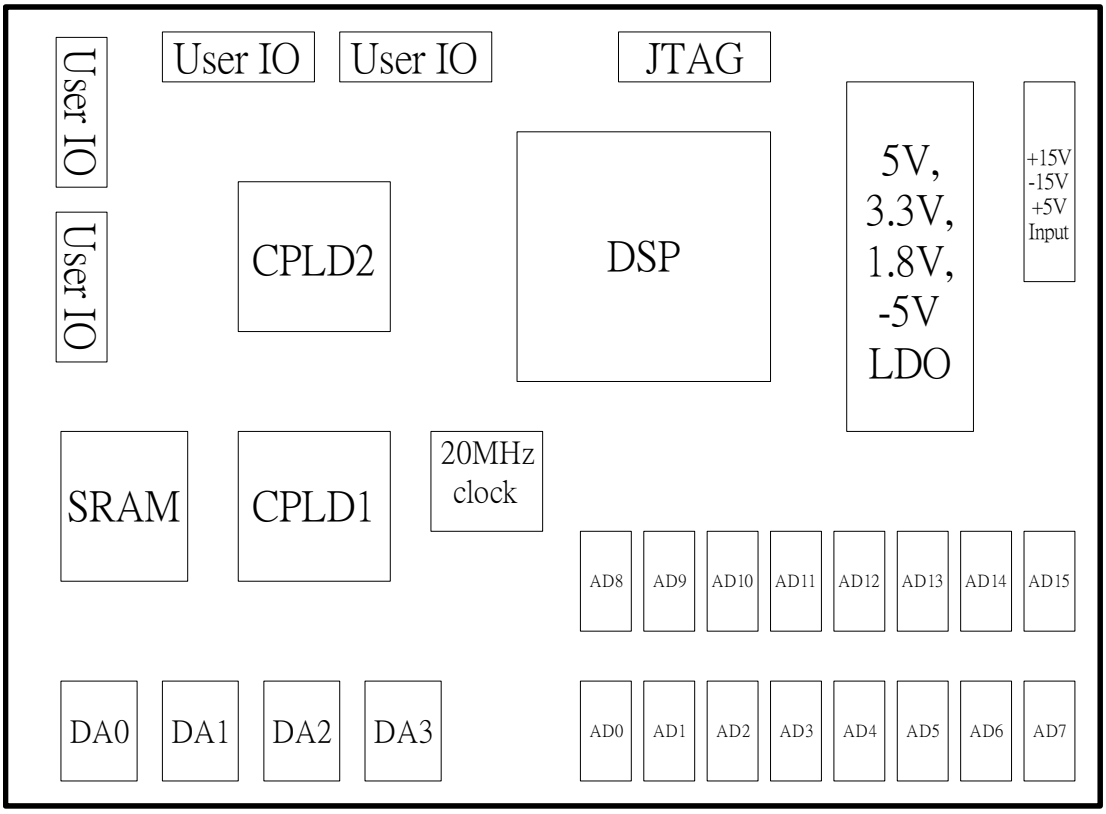

圖 3.1 硬體元件簡稱及配置圖

- 從類比/數位轉換器到靜態隨機存取記憶體

CPLD2 具有一個 8 位元的上下數計數器(up-down counter)，作為其 PWM 功能中的三角波。此計數器可視使用者的需求，產生一個通知 CPLD1 開始取樣的脈衝訊號，以控制本系統的外部類比訊號取樣頻率。若此脈衝訊號設計在計數器等於 0x0（或 0xFE，即此計數器的最大值）時產生，則系統對於外部類比訊號的取樣頻率等於三角波頻率；若設計此訊號在計數器等於 0x0 和 0xFE 時都必須產生，則對於外部類比訊號的取樣頻率即變為三角波頻率的兩倍。為了方便描述，將此訊號定義為 CPLD1 的取樣重置訊號。以本論文的設計為例，當計數器等於 0x0 時，CPLD2 輸出取樣重置訊號給 CPLD1，以使 16 組類比/數位轉換器開始取樣外部的類比訊號，如圖 3.2 所示。

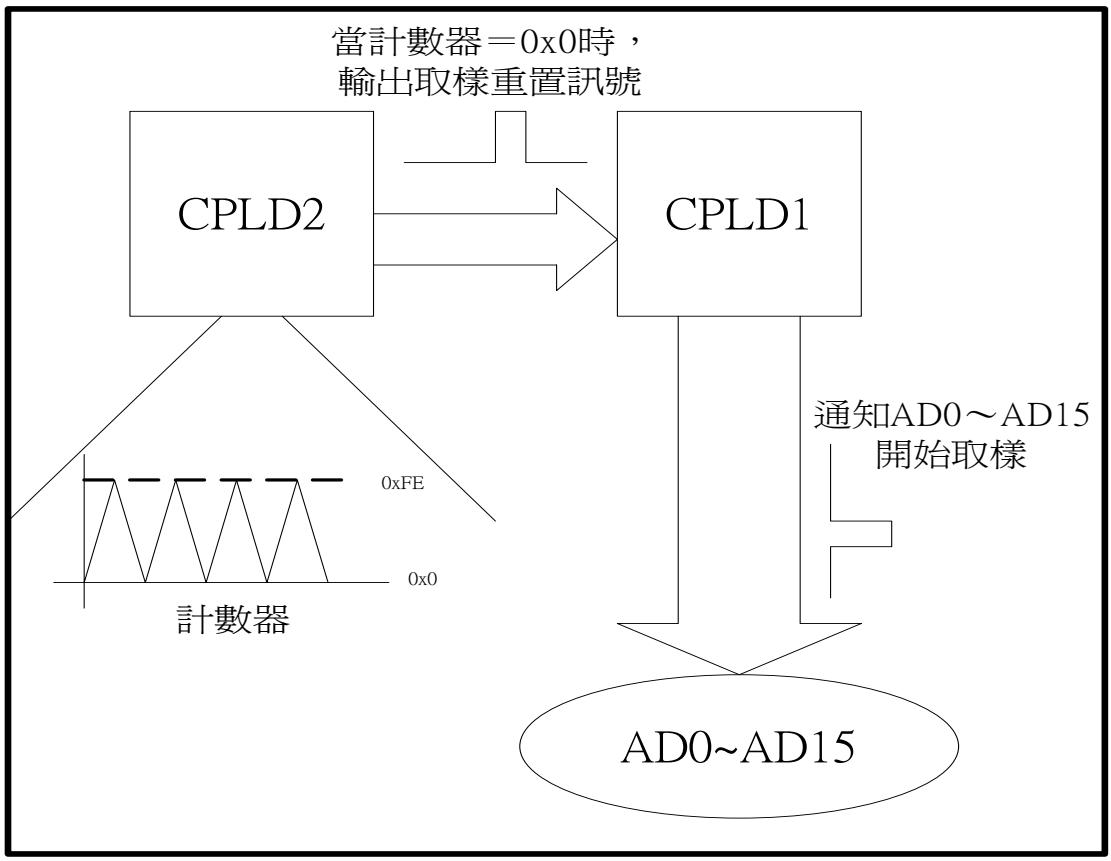

圖 3.2 取樣重置訊號的產生及作用流程圖

16 組類比/數位轉換器開始取樣後，會將外部類比訊號轉換成 12 位元的數位資料，以串列格式傳輸到 CPLD1。由於 CPLD1 的容量不夠，所以必須透過 CPLD1 的控制，將這些資料存放於 SRAM 中，供 DSP 讀取。一般使用並列格式傳輸的類比/數位轉換器之系統，其存取資料於 SRAM 的方式，大都採一個位址存一組轉換後的資料。因此，有幾組類比/數位轉換器，SRAM 就儲存幾筆數位資料。

本系統若採用並列格式傳輸的類比/數位轉換器，必須在電路板上佈出 192 條傳輸線路到 CPLD1。如此一來，不但會大幅增加電路板面積；而且 CPLD1 的輸出入腳位也不敷使用。所以，本系統使用串列格式傳輸的類比/數位轉換器，來傳輸轉換後的數位資料。為了要保持同時取樣 16 組外部類比訊號的特點，加上 CPLD1 又無足夠的空間進行串列格式至並列格式的轉換，勢必得改變在 SRAM 中儲存這些數位資料的方式。圖 3.3 表示出本系統如何同時將 16 組經轉換後的數位資料，透過 CPLD1 的控制寫入 SRAM 中。

CPLD1 會提供每一組類比/數位轉換器一個時脈訊號（clock signal，如圖 3.3 中的 CLK）及一個取樣轉換訊號（convert signal，如圖 3.3 中的 CONV）。當 CONV 訊號使每一組類比/數位轉換器開始取樣轉換之後，必須先經過 2 個時脈週期的轉換時間(conversing time)。從第三個時脈開始，每一個類比/數位轉換器就從最高位元(MSB)依序地將轉換後的數位資料，隨著時脈傳送出去。等到 12 位元傳送完畢之後，類比/數位轉換器就輸出高阻抗(high impedance)，等待下一次的取樣轉換。

CPLD1 從類比/數位轉換器取樣轉換後的第三個時脈週期開始，陸續接收到每一具類比/數位轉換器所輸出的數位資料。也就是說，CPLD1 會從第三個時脈週期到第十四個時脈週期，依序接收到 12 筆 16 位元的資料。CPLD1 一接收到一筆資料，在這個時

脈週期之內，就對 SRAM 輸出一個位址、一個寫入致能(write enable, WE)訊號，且馬上存入這筆 16 位元的資料。同樣的步驟會進行十二次，即把每一組類比/數位轉換器轉換後的 12 位元數位資料，按照其位元順序劃成 12 組，再依序地寫入 SRAM 中。舉例來說，就是將每一組數位資料的最高位元儲存於 SRAM 中的位址 0；次高位元儲存於位址 1；第三位元儲存於位址 2；依序至最低位元儲存於位址 11。如此一來，CPLD1 只是負責輸出這些元件的控制訊號，並且充當數位資料寫入 SRAM 時的緩衝器而已。

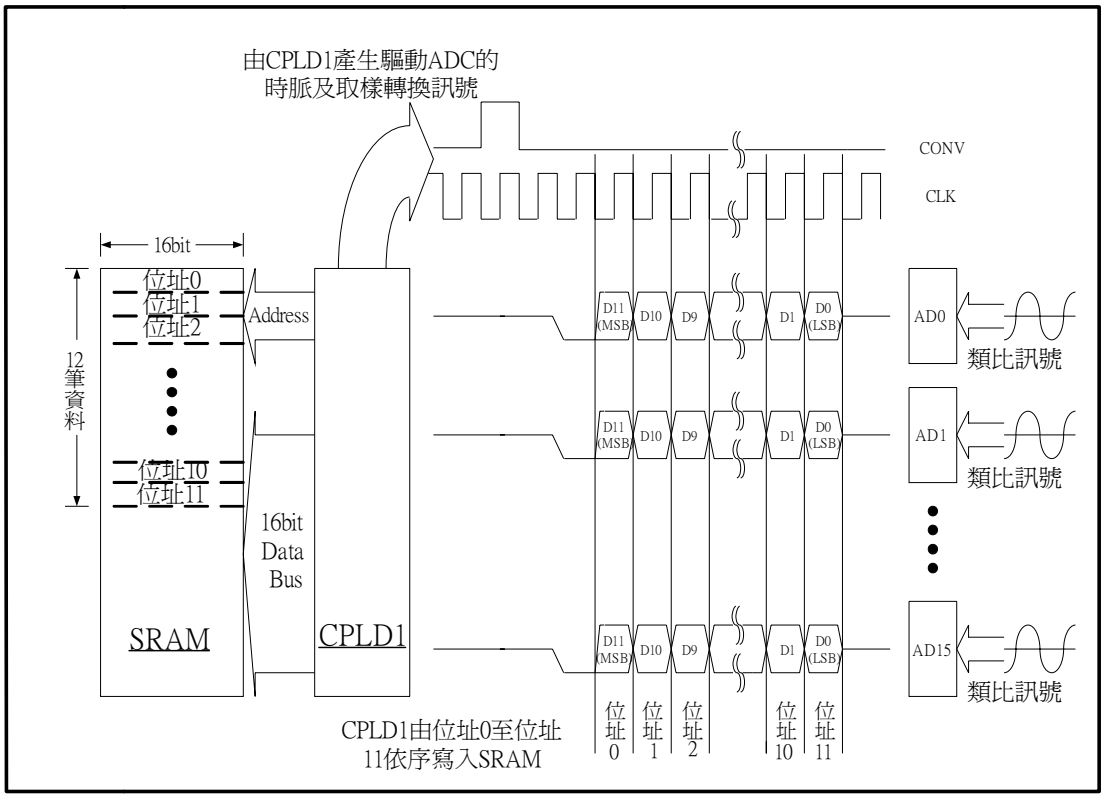

圖 3.3 16 組 ADC 透過 CPLD1 將數位資料寫入 SRAM 流程圖

- 從靜態隨機存取記憶體到數位訊號處理器

  當所有轉換後的數位資料寫入 SRAM 之後，CPLD1 會輸出

一個脈衝訊號致使 DSP 產生外部中斷(external interrupt)，讓 DSP 的程式能夠進入中斷函式，執行裡面的工作。此時，DSP 可以透過外部記憶體介面(external memory interface, EMIF)來完成經由 CPLD1 的解碼器(decoder)，對 SRAM 輸出對應位址和讀取致能(read enable, RW)訊號的動作。SRAM 則會根據接收到的位址，輸出對應的資料經由 CPLD1 供 DSP 讀取。整個流程如圖 3.4 所示。

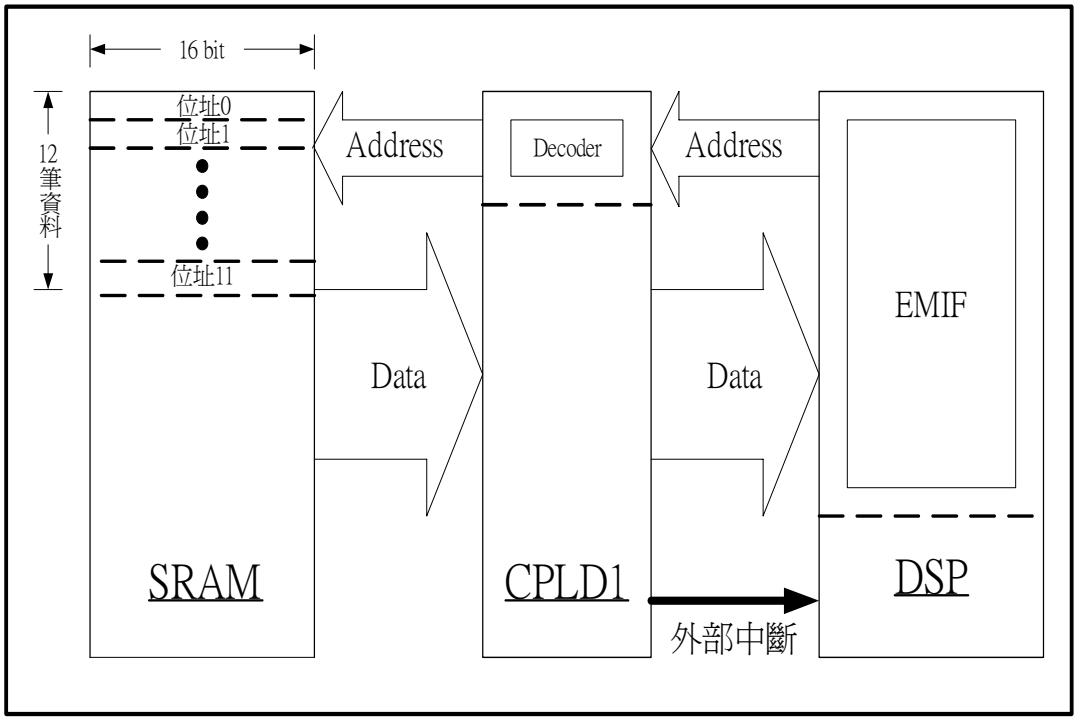

圖 3.4 DSP 透過 CPLD1 讀取 SRAM 流程圖

由於 DSP 真正所要處理的是每一組經類比/數位轉換器轉換後之數位資料。但是在 SRAM 裡的每一個位址，卻是儲存這 16 組數位資料的每一個位元組合。例如在位址 0 儲存的是最高位元；位址 1 儲存的是第二位元；位址 11 儲存的是最低位元等，如圖 3.5 所示。所以，DSP 必須把 SRAM 裡 12 個位址的資料連續讀入，存在內部記憶體(internal memory)中。然後，透過設計程

式的方式,將這 12 筆 16 位元的資料轉換成 16 筆 12 位元的資料。每一筆皆對照每一組類比/數位轉換器所取樣轉換後的數位值,以供程式呼叫使用。

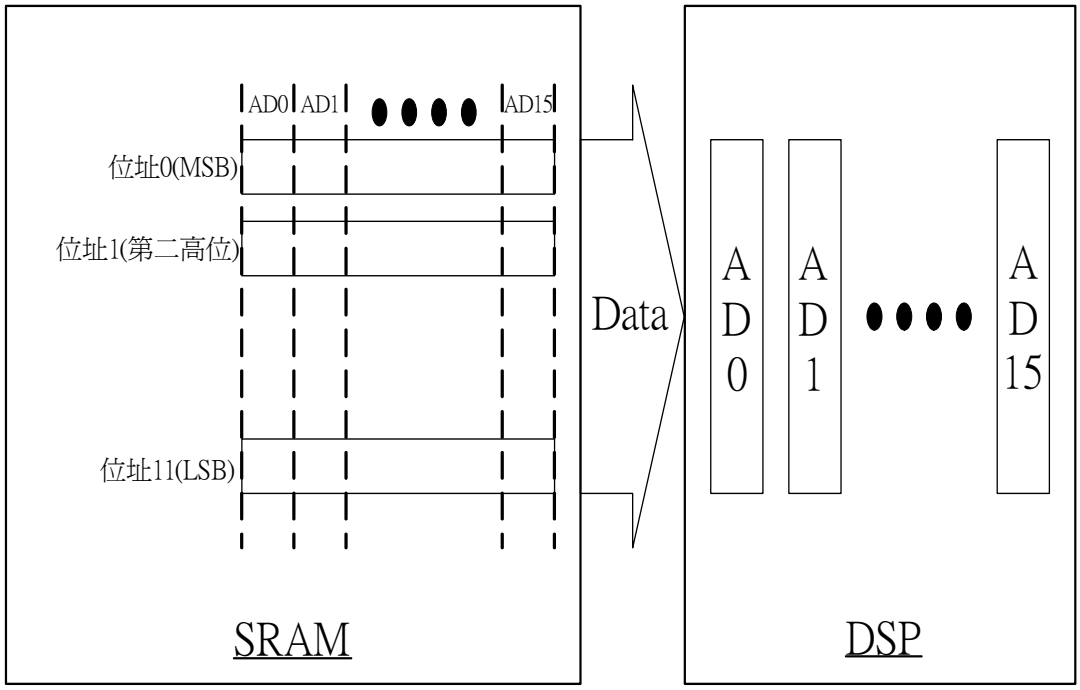

圖 3.5 DSP 讀取 SRAM 資料後，位址重新配置示意圖

- 從數位訊號處理器到數位/類比轉換器

本系統為了偵錯與量測的方便,設計 DSP 可以透過外部記憶體介面的位址輸出埠和資料傳輸連接埠，將要觀察的資料（必須是 DSP 內部記憶體所儲存的資料）存入 CPLD1 中。再由 CPLD1 將這些 16 位元的資料，取其高位 12 位元進行並列轉串列的處理，如圖 3.6 所示。最後從最高位元(MSB)依序傳送到數位/類比轉換器，轉換成易於觀察的類比訊號輸出，如圖 3.7 所示。由於本系統中數位/類比轉換器只設計了 4 組，故 CPLD1 可以視作一個具有四組 16 位元容量的靜態隨機存取記憶體，僅供 DSP 寫入欲觀測的資料而已。

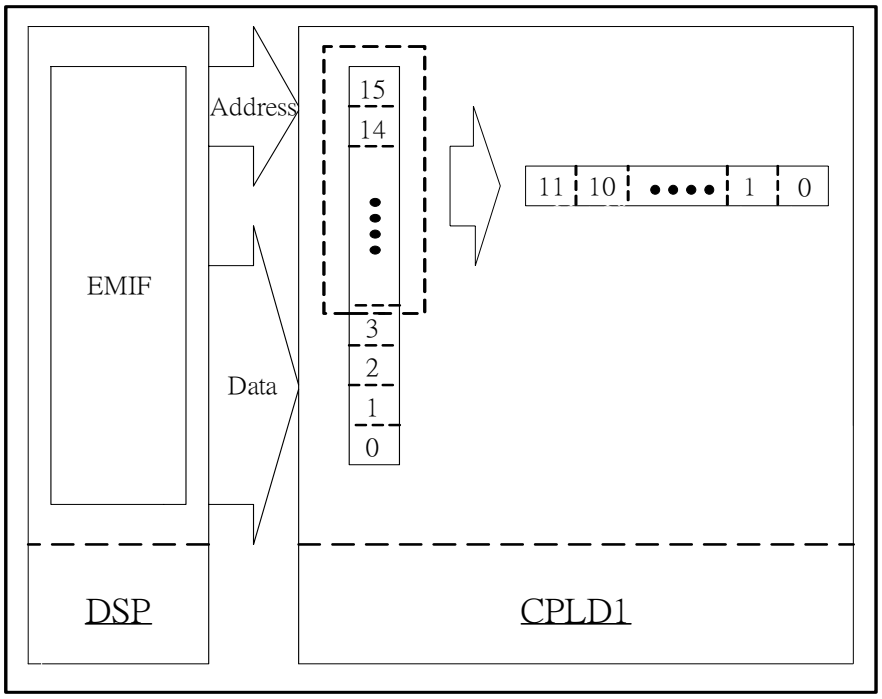

圖 3.6 DSP 寫入 CPLD1 資料進行並列轉串列示意圖

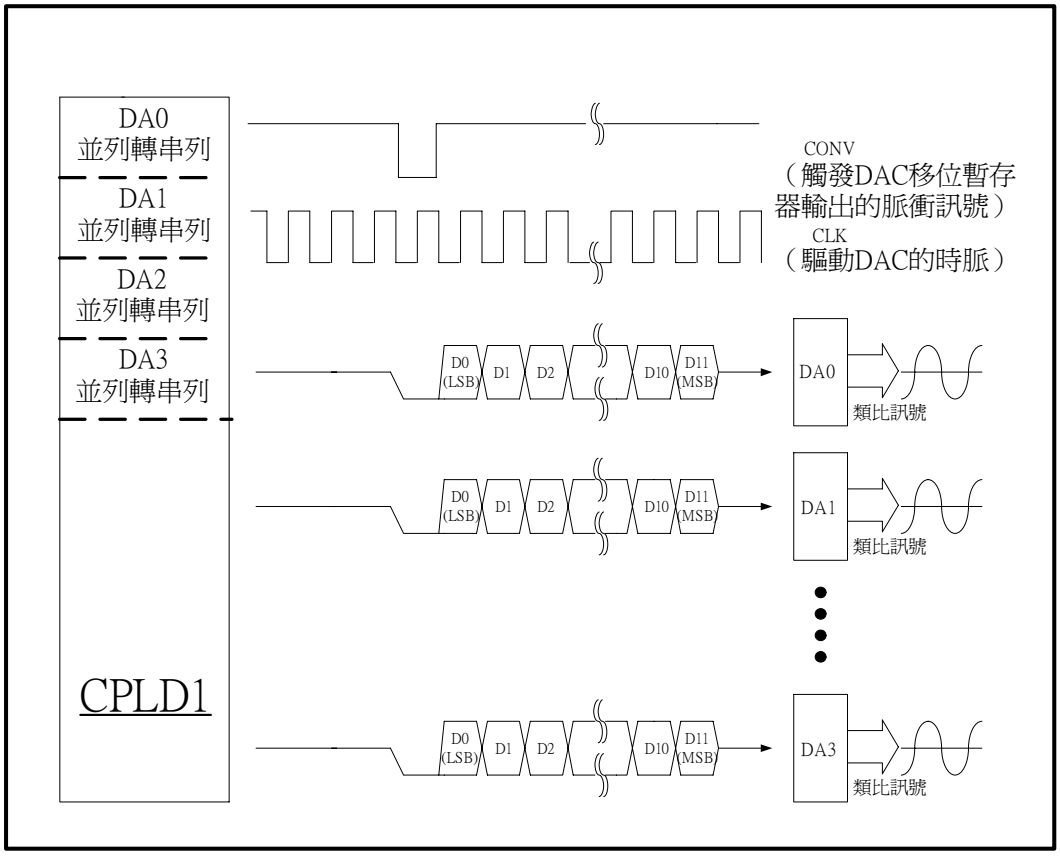

圖 3.7 CPLD1 傳輸欲觀測資料到 DAC 之流程圖

- 從數位訊號處理器到具有 PWM 功能的 CPLD2

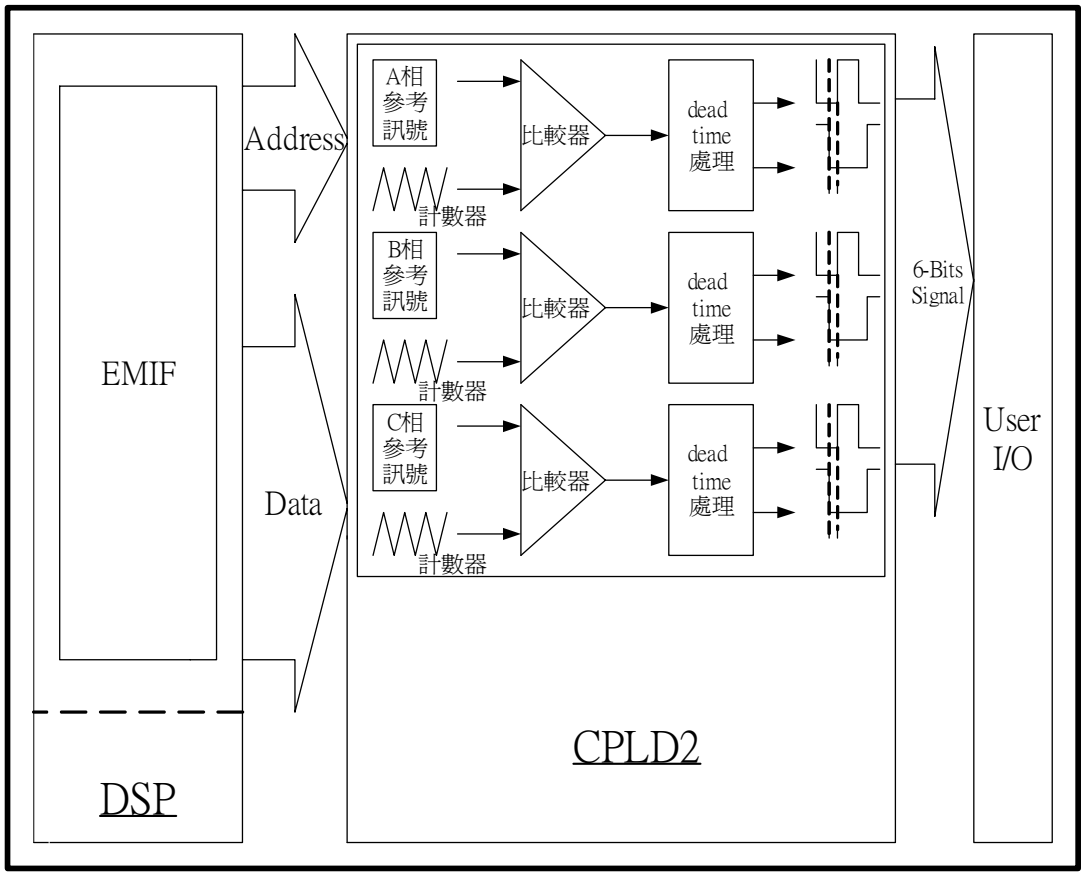

圖 3.8 DSP 輸出到具有 PWM 功能之 CPLD2 的訊號流程圖

本系統為了利於電力電子的應用，設計了一個整合 PWM 功能模組的 CPLD(為了不和本系統另一個 CPLD 混淆，根據圖 3.1 稱此 CPLD 為 CPLD2)，可供 DSP 透過外部記憶體介面，寫入三筆 16 位元的資料，作為三相 PWM 的參考(reference)訊號。這三個參考訊號與上下數計數器所產生的三角波比較之後，產生三組切換訊號。最後這三組訊號經過空白時間(blanking time)的處理，即可產生六組控制三臂六個開關的切換訊號，由電路板上的使用者輸出入埠(user I/O)輸出到一般電力電子開關元件的驅動器上，控制開關的動作。詳細的訊號流程如圖 3.8 所示。

以上這四個步驟，從 CPLD2 輸出取樣重置訊號給 CPLD1 開始，整個流程會在一個取樣週期內完成。使用者可以根據本身實際應用的需求和 DSP 程式的大小，來調整取樣週期，以獲得理想的實驗結果。

3.2 硬體描述語言 VHDL 簡介

本系統利用 CPLD 控制週邊元件的輸入、輸出，以及和 DSP 間的溝通。而規劃設計 CPLD 的方法有很多種，本系統採用 VHDL 硬體描述語言，來設計本系統中 CPLD 的功能。

所謂 VHDL 是 Very High Speed Integrated Circuit Hardware Description Language 的英文縮寫。VHDL 規範的目的，即是要提供一個高階且快速的設計工具。其中涵蓋了電路之描述、電路之合成與電路之模擬等三大電路設計工作。

在 1970 年到 1980 年間，美國國防部為了將電子電路的設計意涵，以文件的方式儲存起來，以便其他人能夠輕易地瞭解電路的設計意義，於是有了發展硬體描述語言(hardware description language, HDL)的念頭。早期的 VHDL 語言，係提供一種將電路模型化的方法，以利工程師來對該電路作實際模擬驗證的工作。在 1987 年的時候，VHDL 成為了 IEEE 的一種標準語言，稱為 IEEE-1076。在 1988 年時，美國國防部並規定所有官方的 ASIC 設計均要以 VHDL 為其電路描述語言。從此以後，VHDL 漸漸成為工業界相互流傳的標準硬體描述語言。IEEE 協會並於 1993 年將 IEEE-1076 的標準，經過一些增修之後，規範了另一個 VHDL 標準，稱之為 IEEE-1164。1996 年時，IEEE 並將電路合成的程式標準與規格，加入到 VHDL 電路設計語言之中，命名為 IEEE-1076.3。

以下歸納出幾點使用 VHDL 來當作電路設計工具的優點[6]：

- 功能強大：VHDL 電路設計語言的功能強大。包含了電路描述，電路合成與電路模擬等功能。就垂直整合而言，從 ASIC 設計到電路板設計，甚至於大型系統的設計，皆有使用 VHDL 的地方。
- 易於整合：VHDL 電路設計語言，將電子電路的設計方式，改變成以行為化描述的方式來設計。因此具有設計快速，更改及維護容易，犯錯機率降低和除錯容易等優點。除此之外，由於 VHDL

具有可流通性，故設計的內容，易為其他人所瞭解而採用，使得 VHDL 的設計整合性極佳。加之以模組化電路設計的趨勢，許多由 VHDL 所設計的模組電路，更易於做重新再組合的動作，使得設計的快速性產生了加成的效果。

- 設計靈活性高：用 VHDL 所寫的設計原始程式，可以透過不同的編譯器或合成器加以合成。所得到的內部接線(netlist)檔案，可以依照不同的邏輯元件和製程技術，將所設計的電路實現出來。
- 具有可攜性：因為 VHDL 是一種工業界標準的電路設計語言，其所提供的可攜性，使設計者可以利用不同的編譯軟體來編譯；也可以利用不同的模擬器來模擬，並且適用於任何種類的邏輯元件與半導體製程技術。

3.3 CPLD 設計介紹

本系統使用了兩顆 ispLSI2192VE 的 CPLD，CPLD1 作為控制輸出入資料和 DSP 間的溝通；CPLD2 作為 PWM 功能模組。本節將分別介紹設計這兩顆 CPLD 的工具、方法以及程式內容。

3.3.1 CPLD 發展軟體簡介

ispLSI2192VE 使用 Lattice 公司發行的 ispLEVER Starter，作為實現其設計和邏輯編程的發展軟體。ispLEVER Starter 除了供本系統所使用的 ispLSI2000VE 系列的晶片應用外，還支援 Lattice 公司所製造的其他系列，例如：ispMACH 4000V/B/C、ispLSI2000E 等。

ispLEVER Starter 可用來實現從設計到邏輯編程的 CPLD 規劃。該軟體包含 Synplicity 公司的 Synplify 發展程式和 Lattice 公司的 ispVM System 晶片邏輯編程程式。整個設計流程可以歸納如下：

- 專案管理

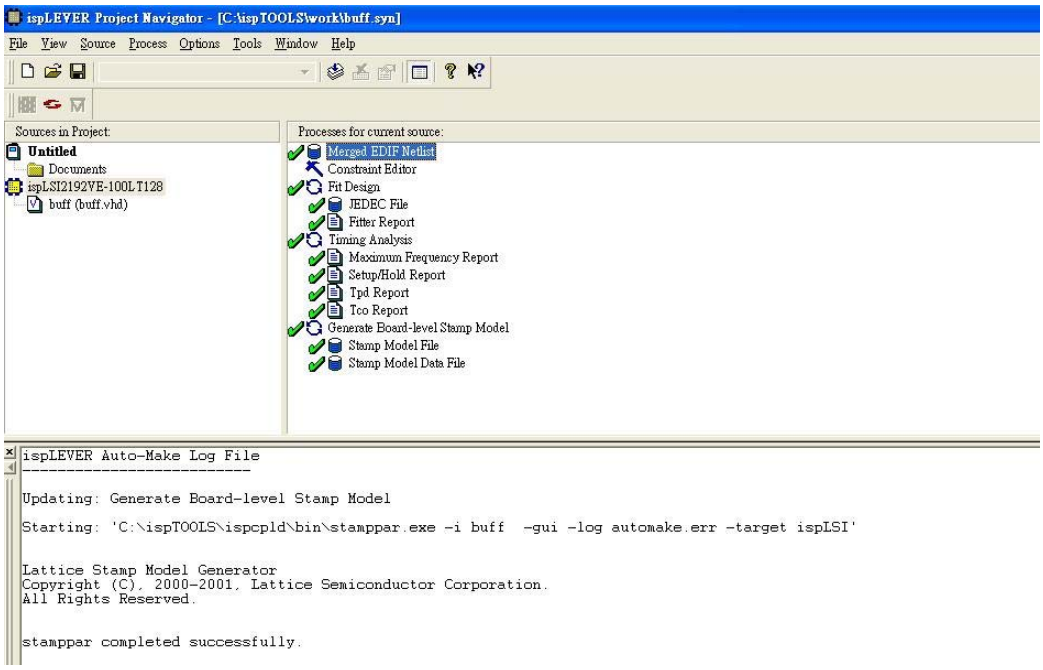

圖 3.9 Project Navigator

ispLever 的 Project Navigator 可以讓使用者對於自己的設計妥善地管理。同一個專案下，可以引入不同的程式進行規劃；並且有進度流程指標，讓使用者對於編譯過程一目瞭然，利於偵錯及更新。圖 3.9 為 Project Navigator 的視窗介面。

- 設計輸入

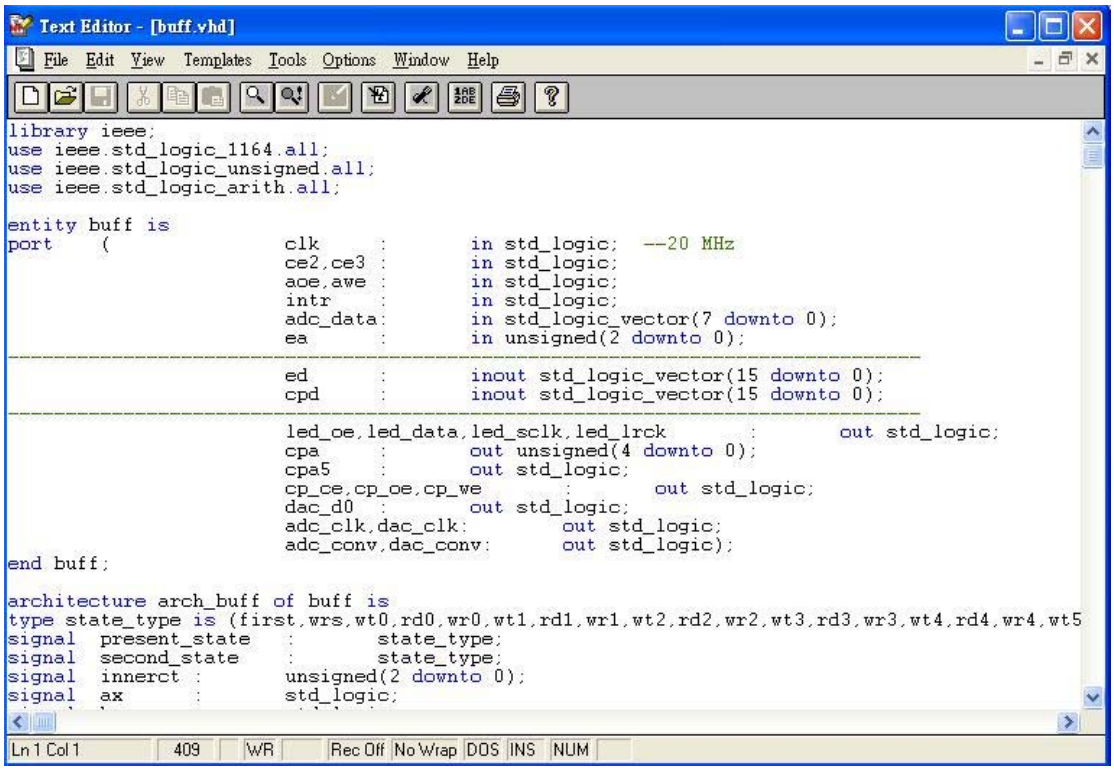

圖 3.10 程式編輯器

ispLEVER 支援 VHDL、Verilog 以及邏輯元件圖與 HDL 語言的混合輸入方式。設計者可以使用內建的程式編輯器，如圖 3.10 所示。或者是一般文書編輯軟體，來撰寫自己的程式。

- 功能模擬

ispLEVER 在程式編譯完成之後，可以依照自己需要的功能來輸入虛擬訊號，來做電路模擬。經過模擬後，就能判斷電路功能是否正確，如果一切無誤，就能下載到晶片去實際驗證。圖 3.11 表示功能模擬器的視窗介面。

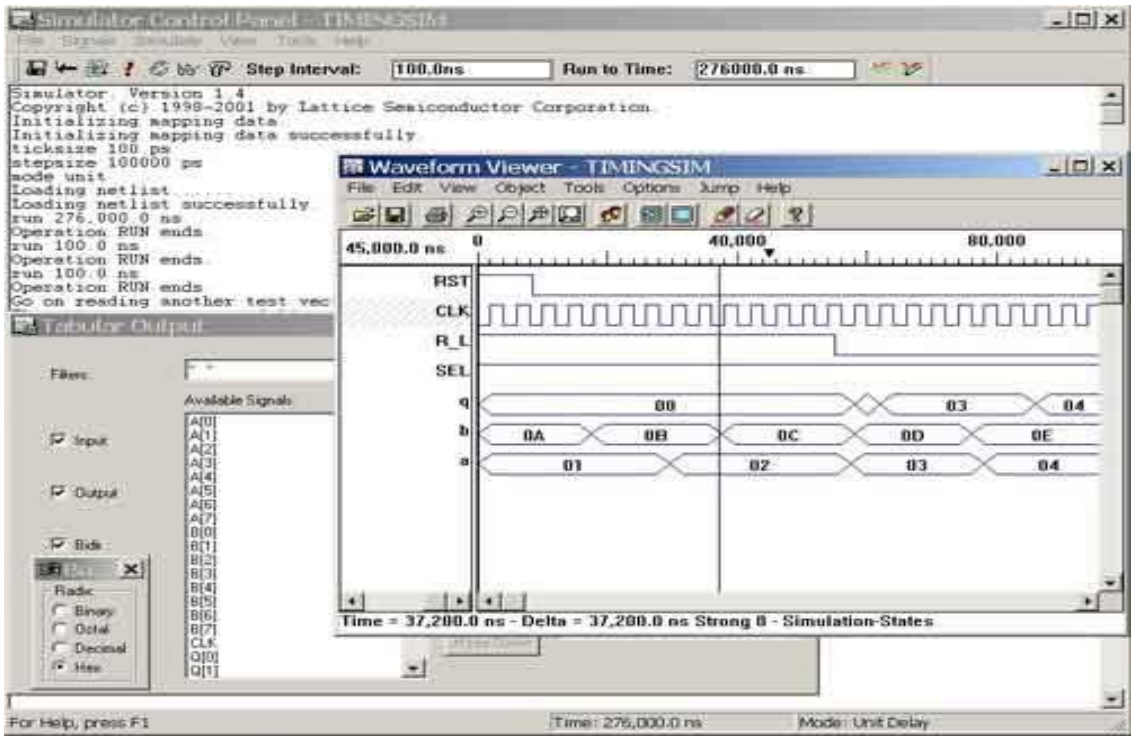

圖 3.11 功能模擬器

- 佈局與繞線

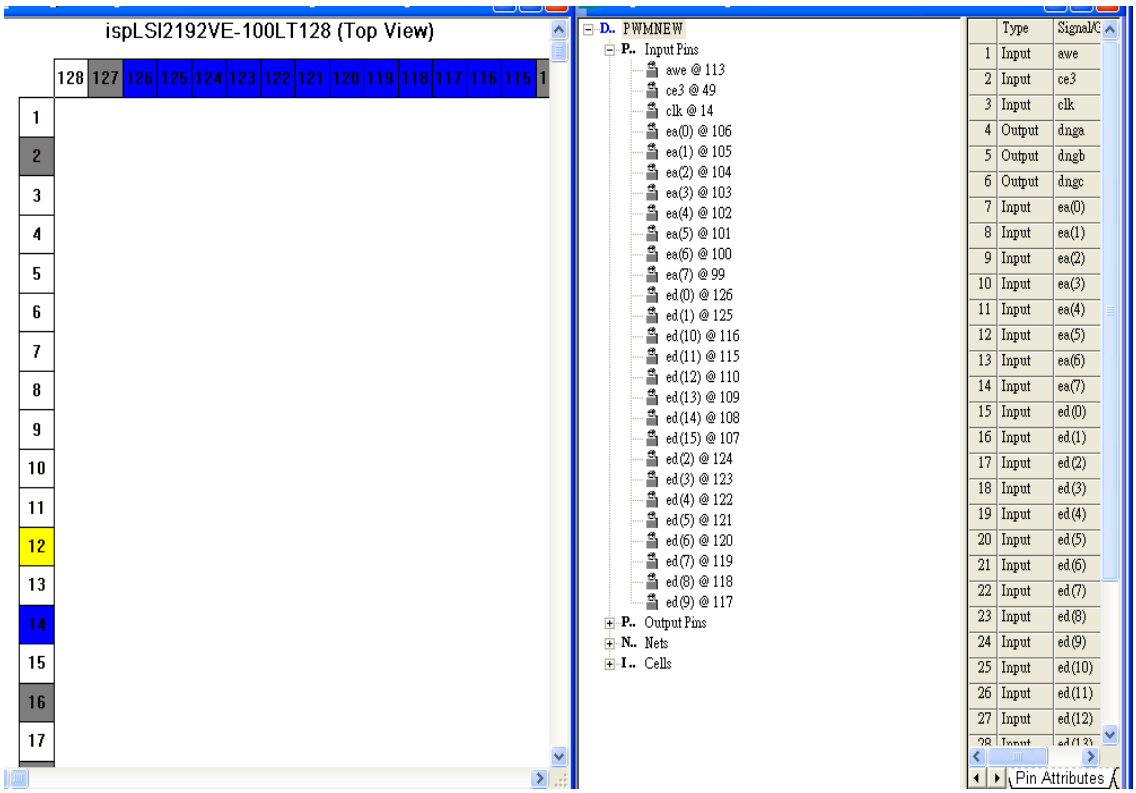

圖 3.12 腳位編輯器

ispLEVER 的腳位編輯器，如圖 3.12 所示，允許設計者自行規劃晶片的接腳功能，以符合所應用的系統電路，具有極大的彈性。再讓編譯器進行佈局與繞線。

- 時序模擬

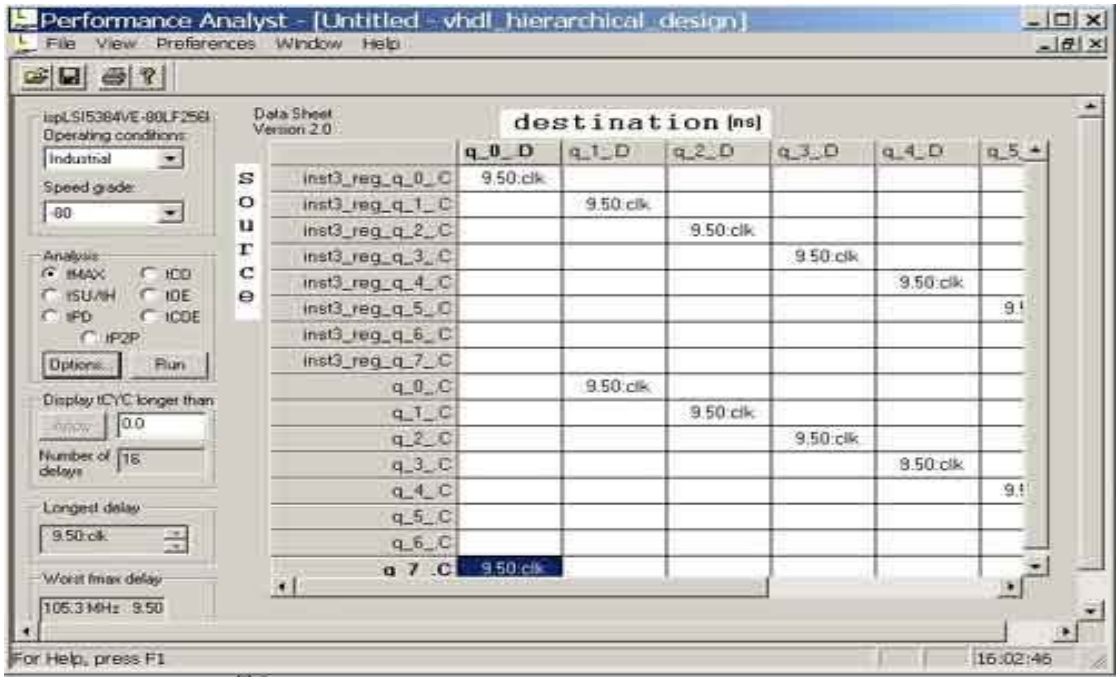

圖 3.13 時序模擬器

設計者可以配合功能模擬器檢查關鍵的延遲路徑，以利修改或更新設計。圖 3.13 表示時序模擬器的視窗介面。

- 邏輯編程

ispLEVER 包含了 ispVM System 邏輯編程硬體規劃軟體。這個軟體提供了一個簡單明瞭的介面，可以利用 .jed 檔直接通過電路板上的處理器，對板上的 CPLD 進行邏輯編程或再編程。讓使用者可以方便地修改 CPLD 的功能，符合實際需求。而這種便捷快速的開發流程，也是使得 CPLD 成為眾多邏輯電路設計者喜愛的原因。其作業視窗如圖 3.14 所示。

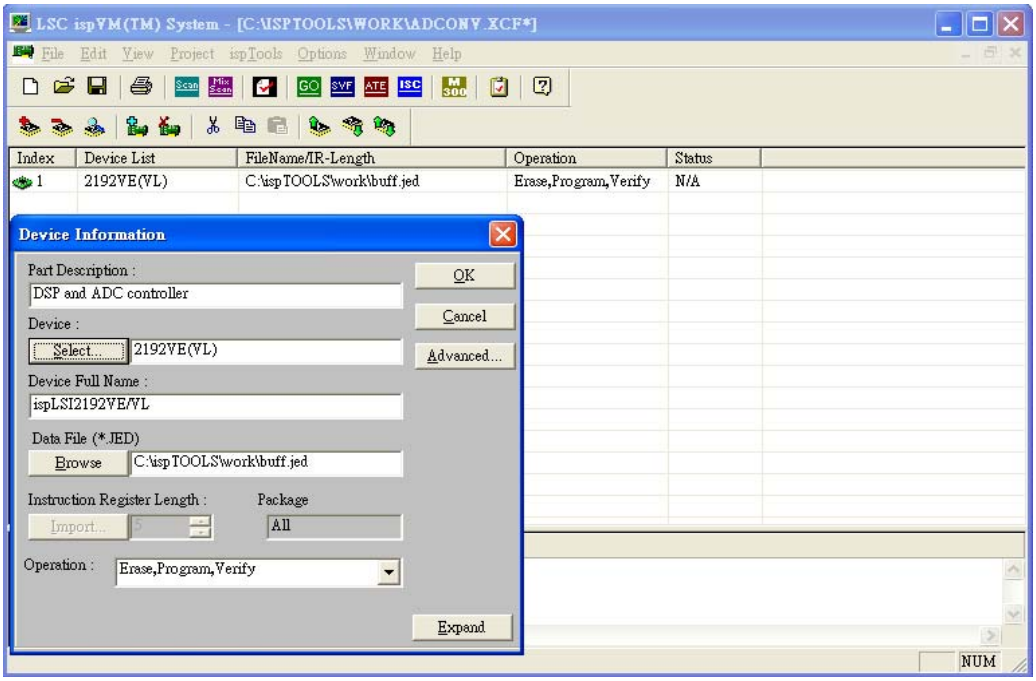

圖 3.14 ispVM System

3.3.2 A/D 與 D/A 訊號處理部分

這個 CPLD（即圖 3.1 的 CPLD1）主要是當作類比/數位轉換器、靜態隨機存取記憶體、數位/類比轉換器與 DSP 間訊號溝通的控制晶片。為了使這些訊號處理流程能夠循序進行，於是在 CPLD1 中設計了一個狀態機(state machine)來控制訊號的動作。狀態機是循序邏輯(sequential logic)電路中，最全面性的一種邏輯方式，但是狀態機通常用在比較特殊的循序邏輯電路設計中，例如一些無法簡單地歸類為移位暫存器或計數器的循序邏輯，皆可以設計一個狀態機來符合所要求之設計規格。

在 CPLD1 中所設計的狀態機，包含了除頻器(frequency divider)、編/解碼器(encoder/decoder)、緩衝器(buffer)及暫存器(register)這些功能，如圖 3.15 所示。

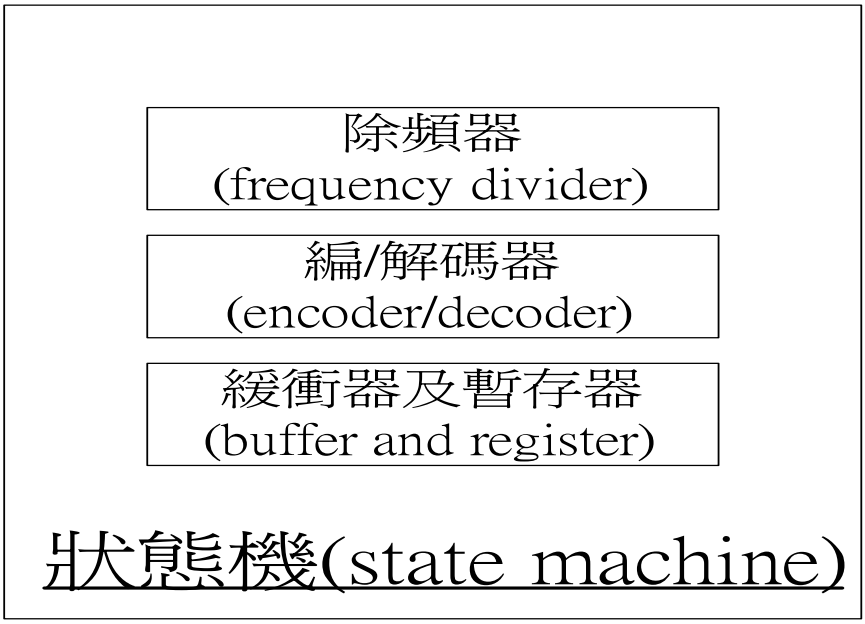

圖 3.15 設計在 CPLD1 中的狀態機所包含之功能方塊圖

除頻器將系統上由石英振盪器產生之 20MHz 的外部時脈（即系統上 DSP 的外頻）除頻成 5MHz，做為週邊類比/數位轉換器、數位/類比轉換器的驅動時脈；並且當作狀態機狀態更新的頻率。編/解碼器把從 DSP 所輸入的位址碼，轉換成對應的儲存位置；或是再輸出對應的位址碼來讀取 SRAM 中的資料。

緩衝器和暫存器是做為 DSP 和週邊元件進行訊號溝通時，資料暫時儲存的位置。整個狀態機共設計了 16 個狀態，從接收到 CPLD2 的取樣重置訊號開始，每隔 200ns（因為狀態更新頻率目前設定為 5MHz）更新到下一個狀態，直到第 16 個狀態才不再更新。必須等到下一次的取樣重置訊號輸入，才又從第一個狀態開始執行。由此可知，取樣重置訊號對於本狀態機而言，等同重置(reset)的功能，具有最大的執行權數(weight)。表 3.1 列出在每一個狀態時，所執行的動作內容，以及將執行的下一個狀態。

根據表 3.1 的內容，可以得知每一個狀態的執行順序和觸發條件。為了使整個狀態流程更加淺顯易懂，可畫出如圖 3.16 的狀態流程圖來表示。

表 3.1 狀態機每個狀態的執行工作表

狀態名稱	執行動作	下個狀態
接收到 取樣重置訊號	● 對 16 組 ADC 輸出取樣轉換訊號。	狀態一
狀態一	● 對 DAC 輸出欲觀測訊號之最高位元(MSB)。	狀態二
狀態二	● 對 DAC 輸出欲觀測訊號之第 11 個位元。	狀態三
狀態三	● 對 DAC 輸出欲觀測訊號之第 10 個位元。 ● 接收 16 組 ADC 所轉換數位資料之最高位元(MSB)，再寫入 SRAM 中的位址"000000"。	狀態四
狀態四	● 對 DAC 輸出欲觀測訊號之第 9 個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 11 個位元，再寫入 SRAM 中	狀態五

	的位址"000001"。	
狀態五	● 對 DAC 輸出欲觀測訊號之第 8 個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 10 個位元，再寫入 SRAM 中的位址"000010"。	狀態六
狀態六	● 對 DAC 輸出欲觀測訊號之第 7 個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 9 個位元，再寫入 SRAM 中的位址"000011"。	狀態七
狀態七	● 對 DAC 輸出欲觀測訊號之第 6 個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 8 個位元，再寫入 SRAM 中的位址"000100"。	狀態八
狀態八	● 對 DAC 輸出欲觀測訊號之第 5	狀態九

	個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 7 個位元，再寫入 SRAM 中的位址"000101"。	
狀態九	● 對 DAC 輸出欲觀測訊號之第 4 個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 6 個位元，再寫入 SRAM 中的位址"000110"。	狀態十
狀態十	● 對 DAC 輸出欲觀測訊號之第 3 個位元。 ● 接收 16 組 ADC 所轉換數位資料之第 5 個位元，再寫入 SRAM 中的位址"000111"。	狀態十一
狀態十一	● 對 DAC 輸出欲觀測訊號之第 2 個位元。 ● 接收 16 組 ADC 所轉換數位資料	狀態十二

	之第 4 個位元，再寫入 SRAM 中的位址"001000"。	
狀態十二	● 對 DAC 輸出欲觀測訊號之最低位元(LSB)。 ● 接收 16 組 ADC 所轉換數位資料之第 3 個位元，再寫入 SRAM 中的位址"001001"。	狀態十三
狀態十三	● 接收 16 組 ADC 所轉換數位資料之第 2 個位元，再寫入 SRAM 中的位址"001010"。	狀態十四
狀態十四	● 接收 16 組 ADC 所轉換數位資料之第 1 個位元，再寫入 SRAM 中的位址"001011"。	狀態十五
狀態十五	● 輸出脈衝訊號，使 DSP 產生外部中斷。	狀態十六
狀態十六	● 等待 DSP 的命令，執行讀取 SRAM 位址或儲存 DSP 輸出值的	保持在狀態十六，直到下

	動作。	個取樣重置訊號輸入。

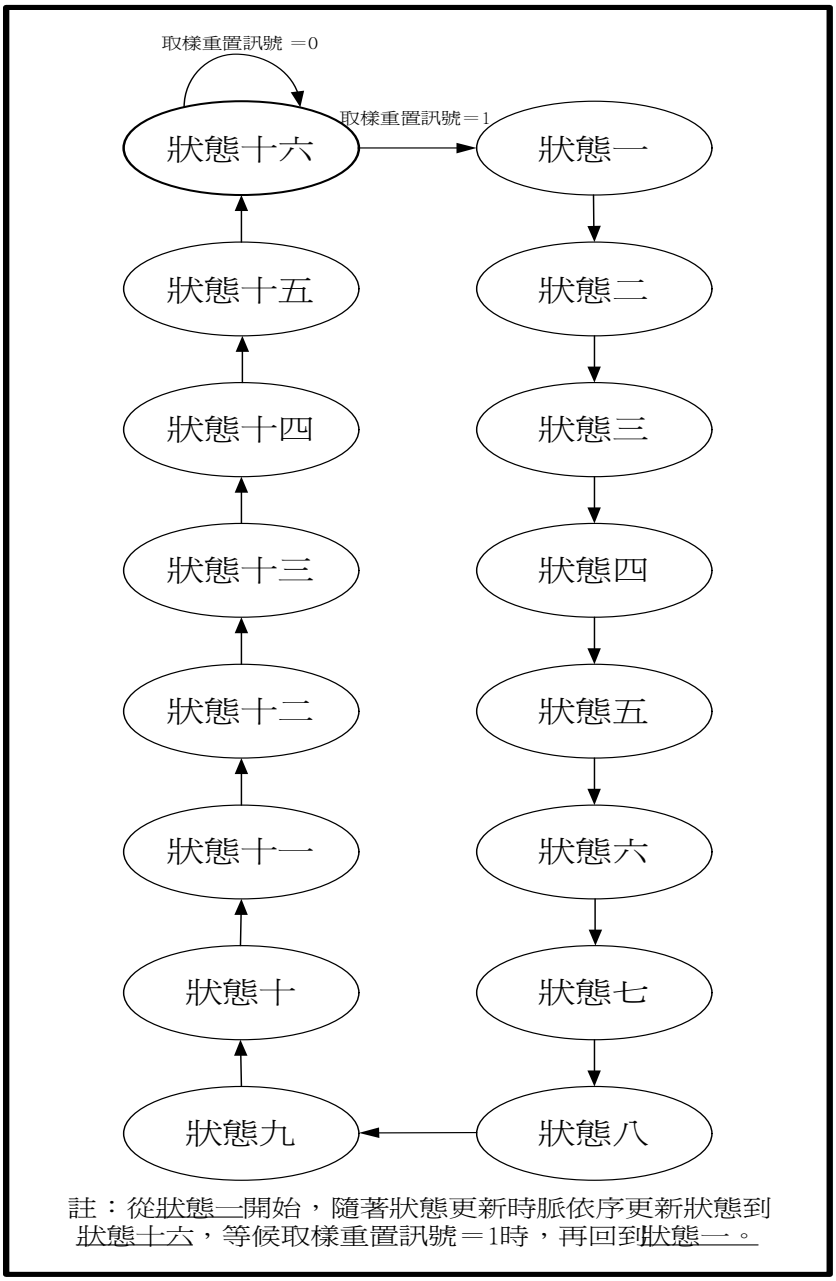

圖 3.16 狀態機的狀態流程圖

3.3.3 PWM 部分

本系統所設計的 PWM 功能模組是由一個 8 位元的上下數計數器，3 個比較器以及空白時間電路所組成。另外具有三個允許 DSP 寫入的 16 位元暫存器，作為 PWM 中和載波(carrier wave)比較的參考訊號。

在 CPLD2 中以一個上下數計數器模擬數位化的三角波，作為 PWM 中的載波。這個計數器利用系統上石英振盪器產生之 20MHz 的時脈訊號進行除頻，得到一個 5MHz 的時脈作為該計數器的輸入。計數器從 0x0 開始，依照 5MHz 的時脈每次上緣觸發則上數 0x01。直到計數器等於 0xFE 時，則改變成下數的狀態，再隨著時脈每次的上緣觸發下數 0x01。等到計數器又為 0x0 時，再變成上數的狀態，如圖 3.17 所示。根據以上的動作原理，可以算出計數器的週期 T_{tri} 為：

$$T_{tri} = \frac{1}{5MHZ} \times (255 + 253) = 101.6\mu s$$

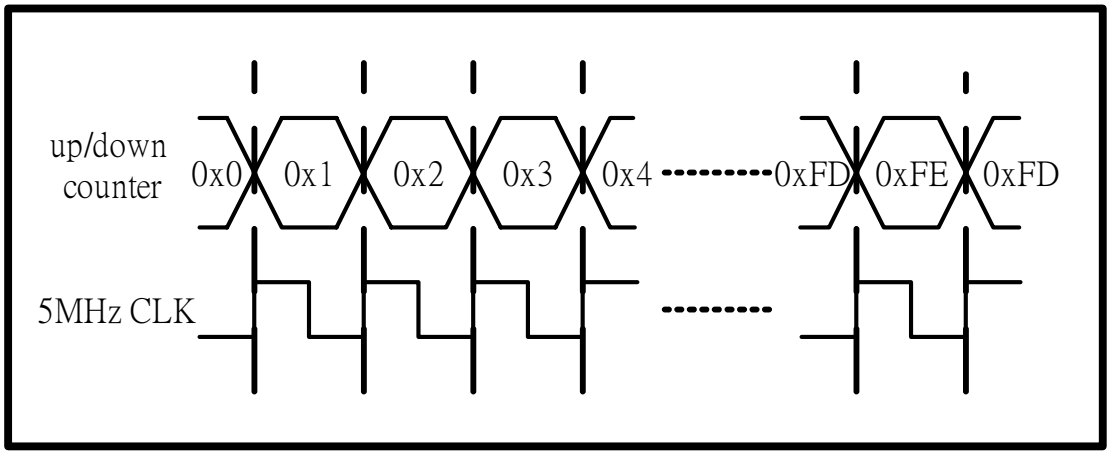

圖 3.17 上下數計數器產生之原理示意圖

由於 DSP 寫入 CPLD2 之參考訊號為 16 位元的數位資料，而且採用 2 補數(2's complement binary)的編碼方式。和 CPLD2 中 8 位元

計數器採用的直接編碼(straight binary)不同。2 補數編碼和直接編碼所代表之十進位值如表 3.2 所示。

從 DSP 輸出的三組參考訊號寫入 CPLD2 後，必須加上 0x8000 以轉換成直接編碼格式，再存入暫存器中。而每個時脈下的計數器數值則必須先左移八位元，變成 16 位元的格式之後，才可以和參考訊號輸入比較器比較，如圖 3.18 所示。另外，參考訊號的更新頻率取決於系統中給予 DSP 的中斷頻率，為了避免每一次資料更新時的暫時錯誤碼，本系統設計以計數器為 0x0 時，CPLD2 所讀取的參考訊號值，來和轉換成 16 位元資料格式的計數器數值進行比較。圖 3.19 表示在 CPLD2 中的三組參考訊號如何和三角波進行比較，產生 PWM 的脈衝訊號。

表 3.2 16 位元直接編碼和 16 位元 2 補數編碼與十進位值之關係

直接編碼	2 補數編碼	代表的十進位值
──	0x8000	-32768
──	0xFFFF	-1
0x0	0x0	0
0x7FFF	0x7FFF	32767
0xFFFF	──	65535

在比較器中，當參考訊號大於三角波，則輸出高電位訊號；否則，輸出低電位訊號。由於有三組參考訊號輸入，所以也會有三組 PWM 脈衝訊號產生。理想上，只要再將這三組訊號各經過反相器(not gate)，即可得到六組控制三相反流器(inverter)的開關訊號，達成以 PWM 控制電力電子轉換器的功能。

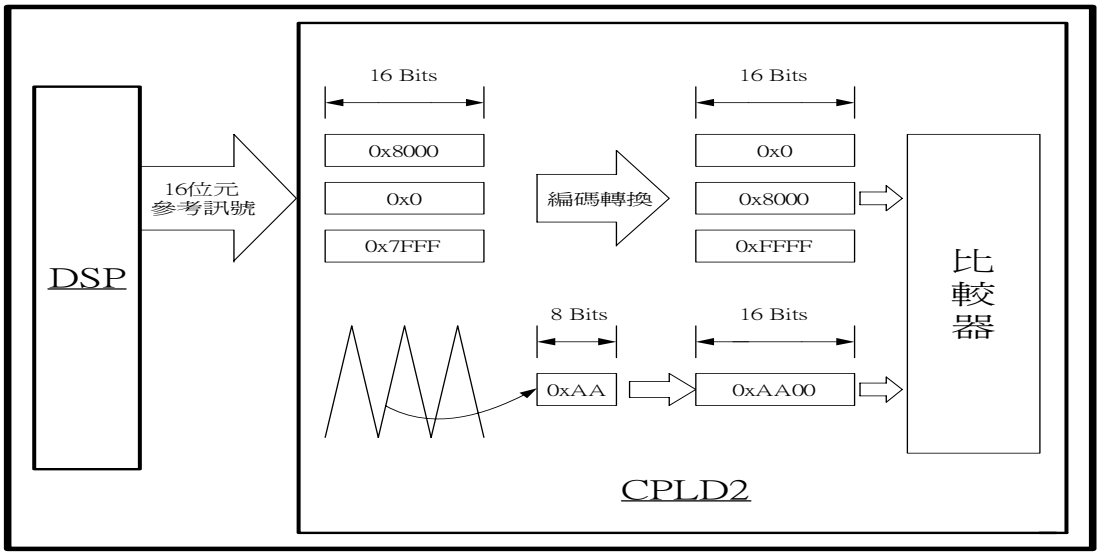

圖 3.18 參考訊號及三角波在比較器前級格式轉換示意圖

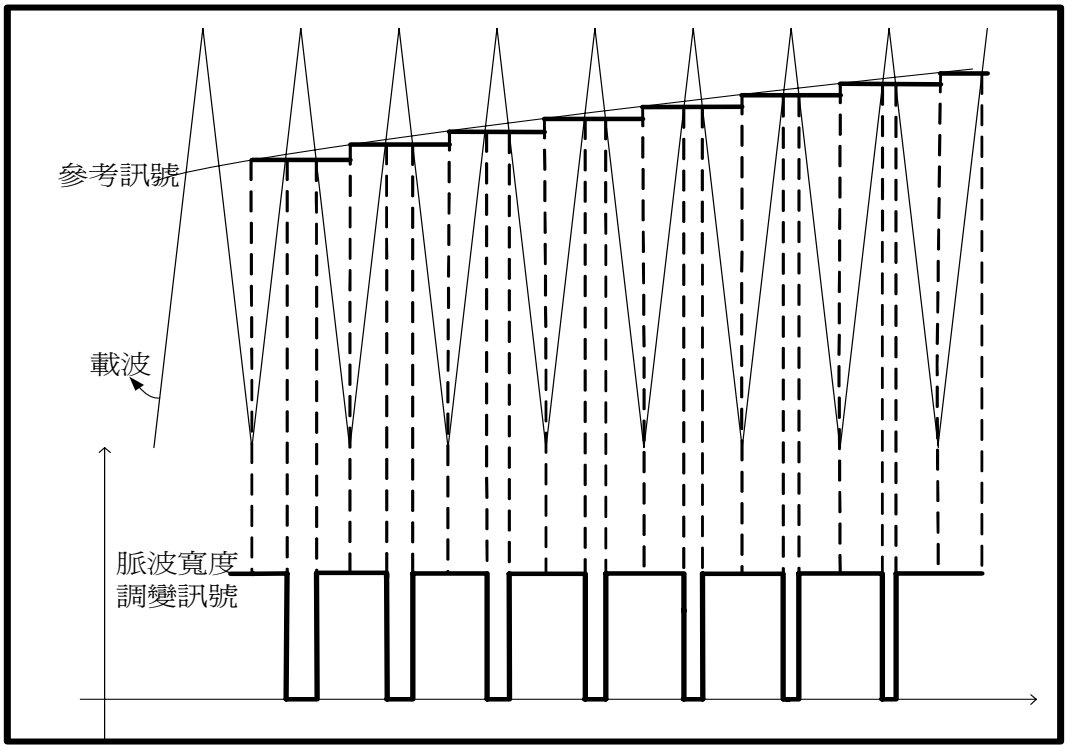

圖 3.19 CPLD 中參考訊號與載波訊號做 PWM 方法示意圖

實際上值得注意的是，在三相反流器中任一臂的上下兩個開關，絕對不允許同時導通；否則將造成短路，引起大電流燒壞半導體開關

元件。為了防止上述的情形發生，任一臂的上下兩個開關驅動訊號，必須做空白時間處理。圖 3.20 為本系統設計的空白時間邏輯電路圖。S1 是某一組參考訊號和三角波經過比較器之後，產生的 PWM 脈衝訊號；S2 是 S1 經過一個 D 型正反器(flip-flop)後的輸出訊號；S3 是 S2 經過一個 D 型正反器後的輸出訊號。取 S2 與 S3 輸入及閘(and gate)可得 SU，為上臂的開關訊號；取 S2 與 S3 輸入反或閘(nor gate)可得 SD，為下臂的開關訊號。這樣的邏輯處理，會使得空白時間發生在每個脈衝訊號上緣觸發的時候。圖 3.21 為經過空白時間處理後的實際訊號波形。空白時間電路中使用 D 型正反器的作用為將訊號延遲一個時脈週期。以本系統為例，驅動 D 型正反器的時脈週期為 3.2μs，則上下臂開關訊號的空白時間即為 3.2μs。

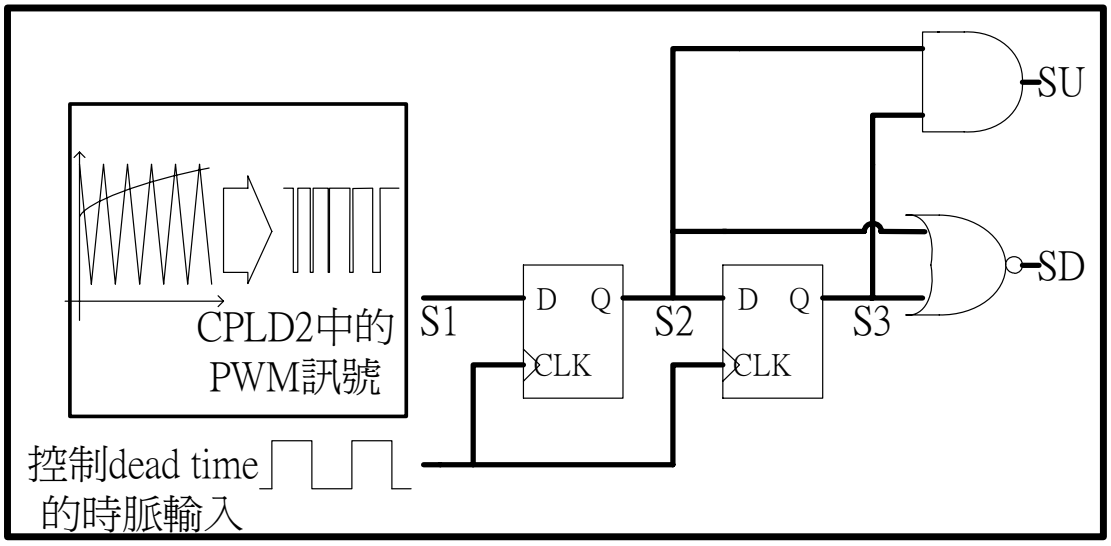

圖 3.20 空白時間邏輯電路圖

經過以上步驟，輸出的六組 PWM 訊號就可以實際應用在驅動外部電力電子開關，作為控制反流器或轉換器(converter)的方法，達成本系統設計的目的。

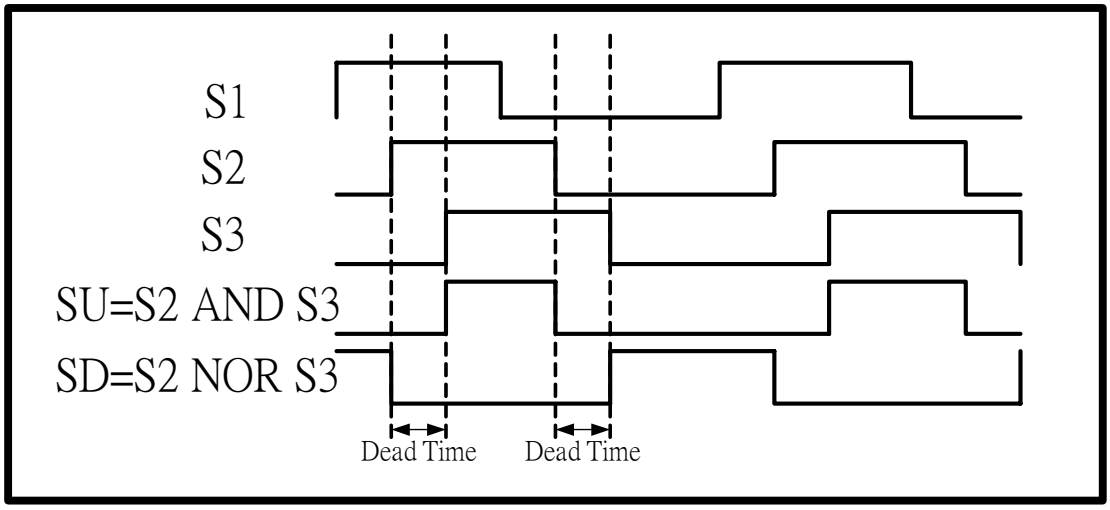

圖 3.21 經過延遲時間處裡的 PWM 訊號波形

3.4 數位訊號處理器程式設計

本系統使用德州儀器公司製造的 TMS320C6711B 晶片，作為數位訊號處理的工作核心，並且以 C 語言來規劃內部介面與設計程式。在 DSP 內部程式發展與除錯階段，還使用德州儀器公司的 Code Composer 搭配 TMS320C6X 系統發展模擬器─XDS510PP 進行程式的修改與模擬。以下兩個小節，將簡介 TMS320C6X 的整合式發展介面 Code Composer 與數位訊號處理器內部的程式設計。

3.4.1 DSP 整合式發展介面─Code Composer

Code Composer Version-2 提供了一個包含軟體編輯工具的 DSP 整合式發展環境(integrated development environment, IDE)。在這個整合環境中，可以對 C 語言、組合語言進行編譯或組譯，產生執行檔。並且在編譯完成後，提供了一個功能模擬和即時偵錯的介面，利於使用者使用[14]。

副檔名.c 的 C 語言程式碼經過 C 編譯器(C compiler)編譯之後，會產生副檔名.asm 的組合語言程式碼。組合語言程式碼再經過組譯器(assembler)後，產生副檔名.obj 的機器碼。最後，連接器(linker)再把機器碼檔案和機器碼資料庫進行連結產生副檔名.out 的可執行檔。所謂可執行檔是指一種連結到一般機器碼的檔案形式(common object file format, COFF)，可以應用在 Unix-based 的系統和大部分的數位訊號處理器。而 C6711B 處理器也是利用這種形式的可執行檔直接下載執行。圖 3.22 顯示使用 Code Composer 進行 DSP 的程式發展流程。

Code Composer 將程式透過 XDS510PP 型的 JTAG(joint team action group)下載至 DSP 晶片中，並且在程式執行時，可以利用個人電腦對 DSP 的動作進行監控、即時分析和偵錯。善用 Code Composer 所提供的即時控制介面，有助於簡短程式發展時間，加快研發腳步。

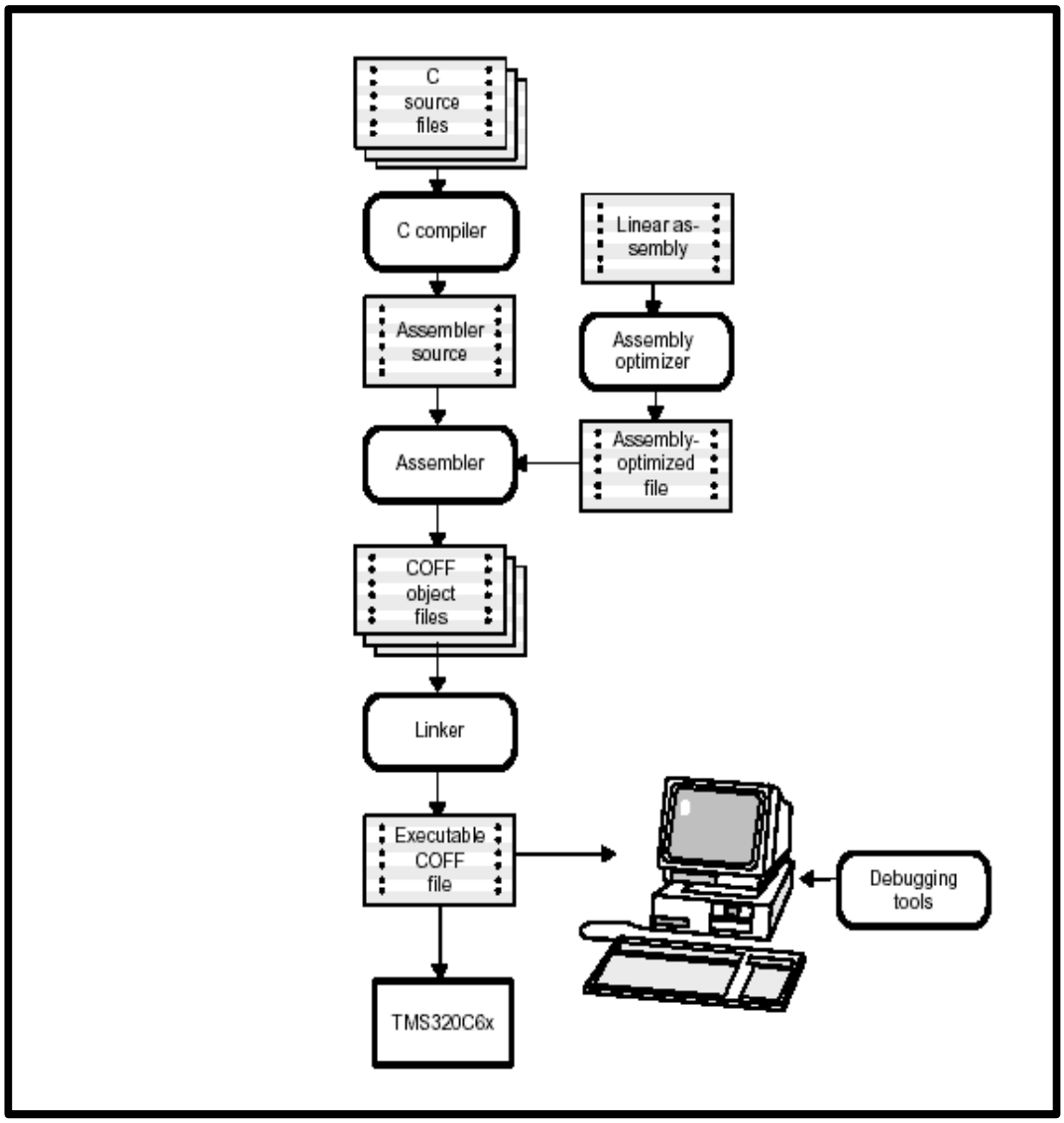

圖 3.22 TMS320C6X 軟體發展流程圖[15]

3.4.2 內部程式簡介

在 TMS320C6711B 系統中發展程式，德州儀器公司提供了完整的軟體發展環境。在編譯器方面，從高階的 C 語言編譯器到低階但較容易撰寫的線性組合語言編譯器，以及程式執行效率最高，但不易撰寫的一般組合語言編譯器，都已兼備。可以視使用者的喜好，選擇

以何種程式語言進行設計。

本論文所發展的數位訊號處理系統，採用 C 語言來進行設計。C 語言是一種高階語言，當初是以模擬人類的語言及邏輯思考方式為設計目的，讓程式設計者可以藉由自己熟悉的語言來設計程式。所以使用 C 語言來設計 DSP，可以提升程式的可讀性，以便日後偵錯與維護。雖然 C 語言比較容易學習及設計，但卻沒有組合語言執行效率高。不過隨著 DSP 的運算速度愈來愈高，編譯器具有最佳化功能，C 語言函式庫的完善，使得 C 語言仍是 DSP 設計者喜愛的程式語言之一。

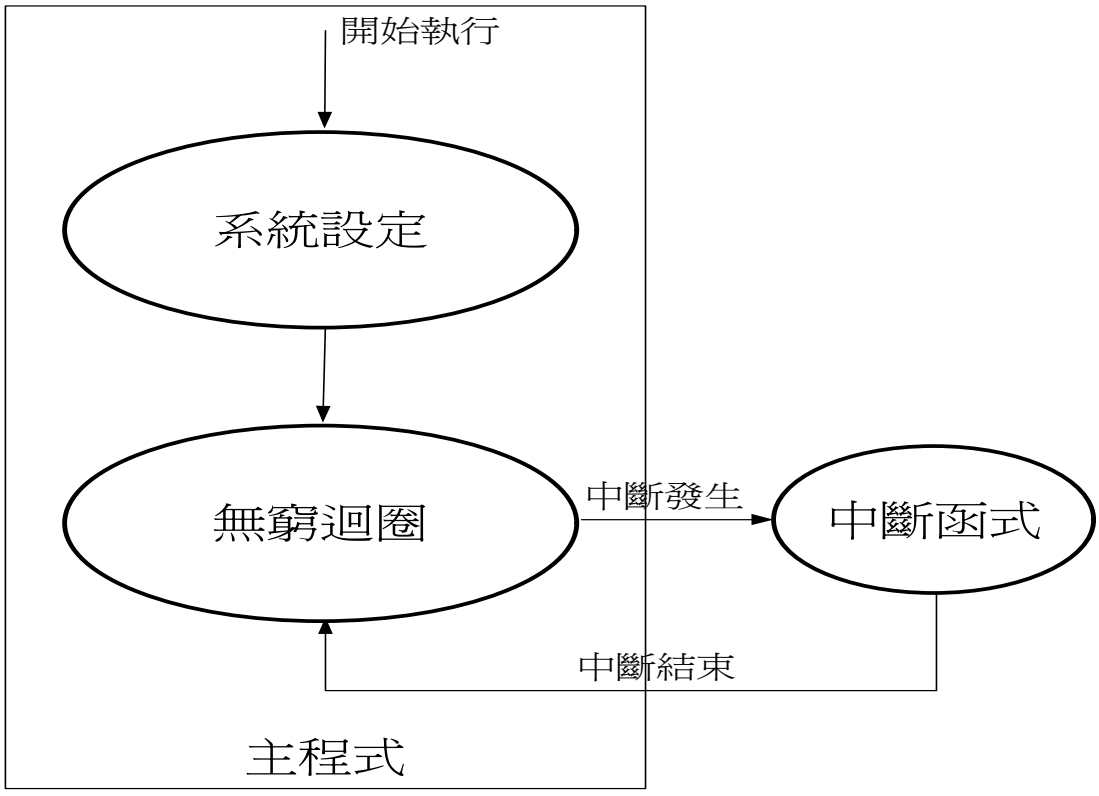

圖 3.23 DSP 中主程式和中斷函式執行順序圖

本系統的 DSP 程式主要分為兩大部分：主程式及中斷函式。主程式包含 TMS320C6X 的週邊介面設定、記憶體規劃和一個等待中斷

訊號的無窮迴圈。中斷函式可以有 12 個，包括 4 個外部中斷、4 個 DMA 中斷、2 個內部計時器中斷、1 個外部記憶體介面的 SDRAM 計時器中斷和 1 個主處理器對此 DSP 的中斷。使用者根據系統設計的中斷訊號，設計對應的中斷函式來撰寫 DSP 程式。主程式和中斷函式的執行順序，如圖 3.23 所示[16]。

以本系統為例，在主程式中需要對於 DSP 的外部記憶體介面(EMIF)進行初始設定，以使 DSP 可以透過 CPLD1 讀取 SRAM 中儲存的資料；或者是將參考訊號寫入 CPLD2 中進行脈波寬度調變。所以必須利用 TMS320C6x 週邊支援函式庫(TMS320C6x peripheral support library)做以下的設定：

```
#include<emif.h>

emif_init(0x3f78, 0xffffff23, 0xffffff13, 0x20e28322, 0x20e28322,
0x248f000, 0x0);
```

emif_init 這個函式的用途就是用來定義控制 EMIF 的幾個暫存器參數。其函式內容定義於 emif.h 中。

在主程式中另外一個初始設定就是中斷控制。中斷控制函式是用來設定 DSP 的各種中斷及其對應的中斷服務函式，皆包含於 intr.h 之中。本系統目前僅利用一個外部中斷，其對應的 DSP 中斷對照號碼為 4。以下為 DSP 中斷控制設定的程式內容：

```
#include<intr.h>

intr_init();

intr_map(CPU_INT4, ISN_EXT_INT4);

intr_hook(c_int04, CPU_INT4);

INTR_GLOBAL_ENABLE;

INTR_ENABLE(CPU_INT_NMI);

INTR_ENABLE(CPU_INT4);
```

經過主程式的初始設定後，最後就進入一個等待中斷的無窮迴圈，等待中斷的產生，以進入對應的中斷函式執行所設計的程式內容。一般而言，設計者都將希望 DSP 執行的演算法和控制策略撰寫在中斷函式中。不過，值得注意的是，如果系統中只設計了單一中斷，執行中斷函式的時間要小於該中斷週期，以免該中斷函式無法完成，造成錯誤的訊號輸出。本系統規劃 DSP 的 C 語言程式架構如下所示[17]：

```
#include "regs.h"
#include "emif.h"
#include "math.h"
#include "intr.h"
變數宣告
interrupt void c_int04()    //interrupt service routine
{
    中斷後執行的程式內容
}
void main()
{
    DSP 初始設定函式
    while(1);                    //infinite loop
}
```