FPGA中的DSP核心設計

唐佩忠

交通大學電子研究所博士(1983)
加拿大Syscor R&D公司硬體部門主管(2000~迄今)
俊原科技公司技術顧問(1998~迄今)
交通大學電機與控制工程系副教授(1985~2000)

2002年11月

FPGA中的DSP核心設計

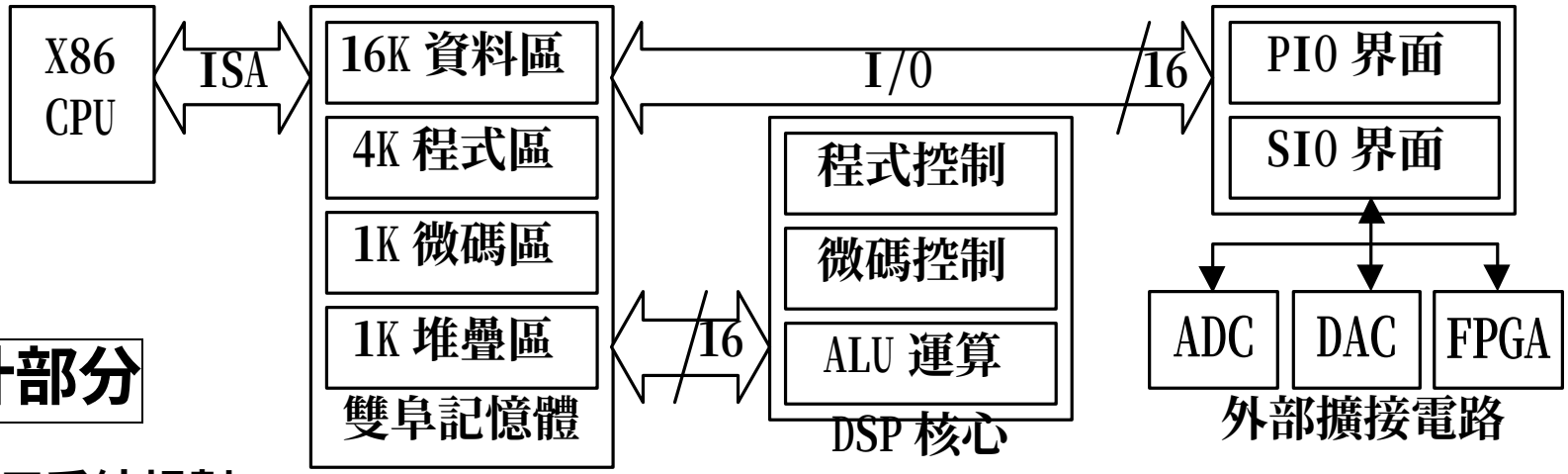
應用系統規劃

控制程式規劃

組合語言規劃

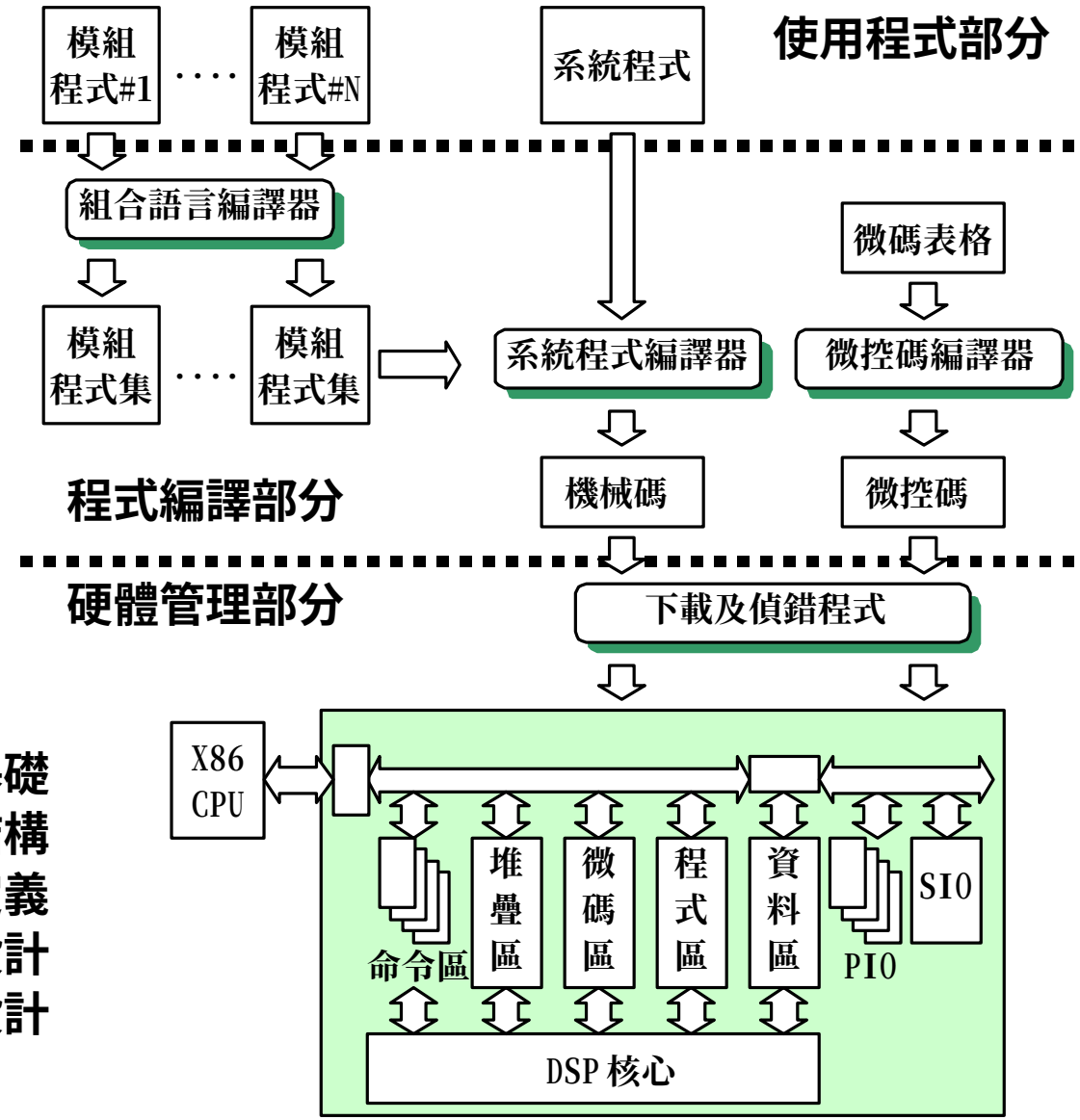
DSP核心設計

教學實驗規劃



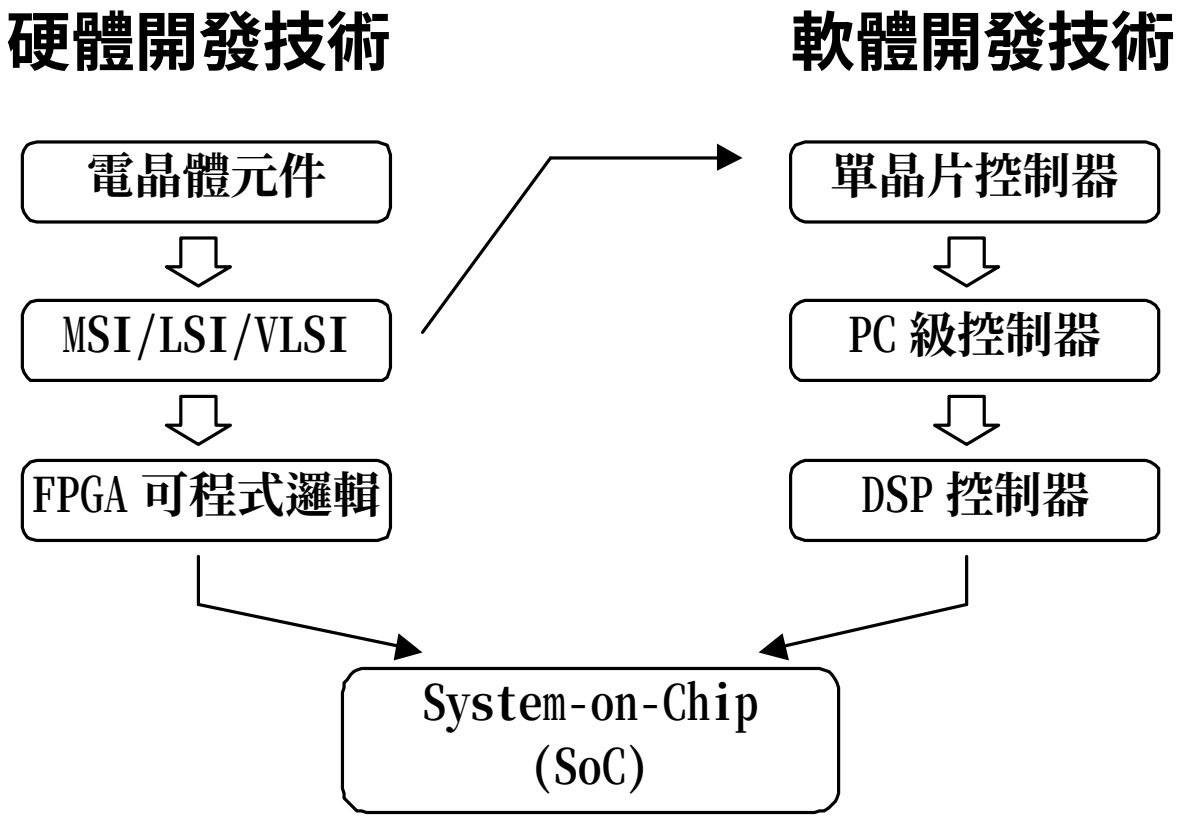
硬體設計部分

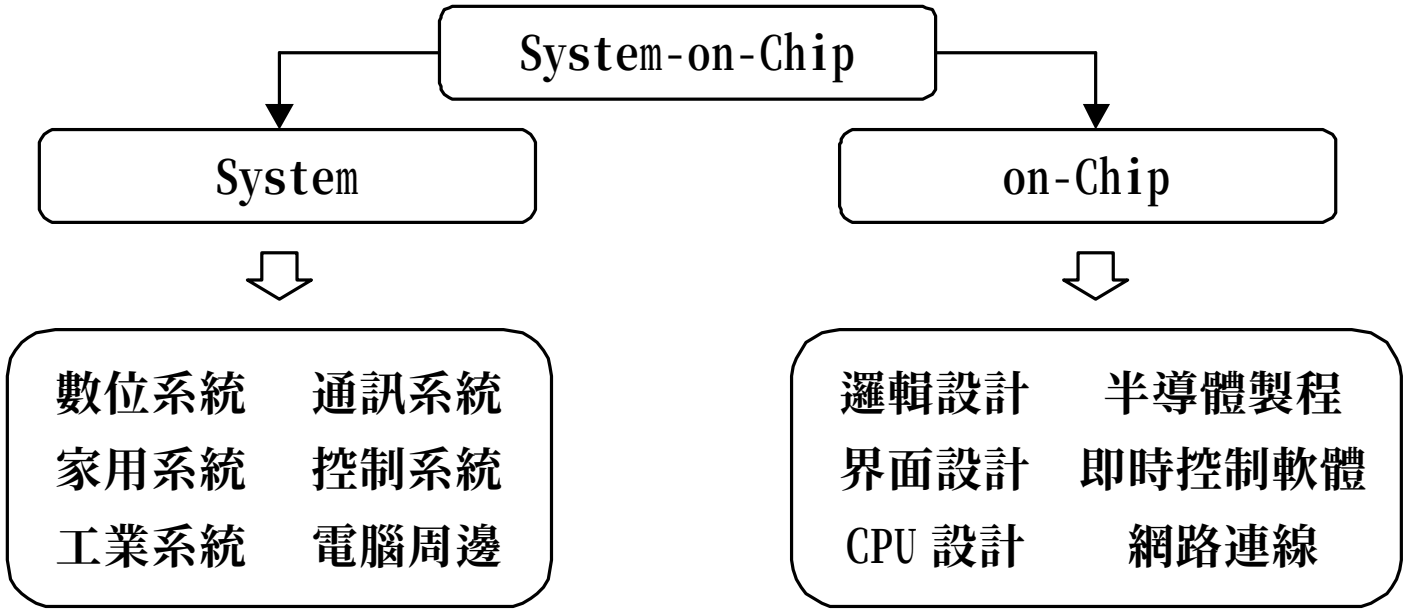
第一章：應用系統規劃
第二章：DSP核心設計
第三章：微控碼設計
第四章：CPU界面設計
第五章：I/O界面設計



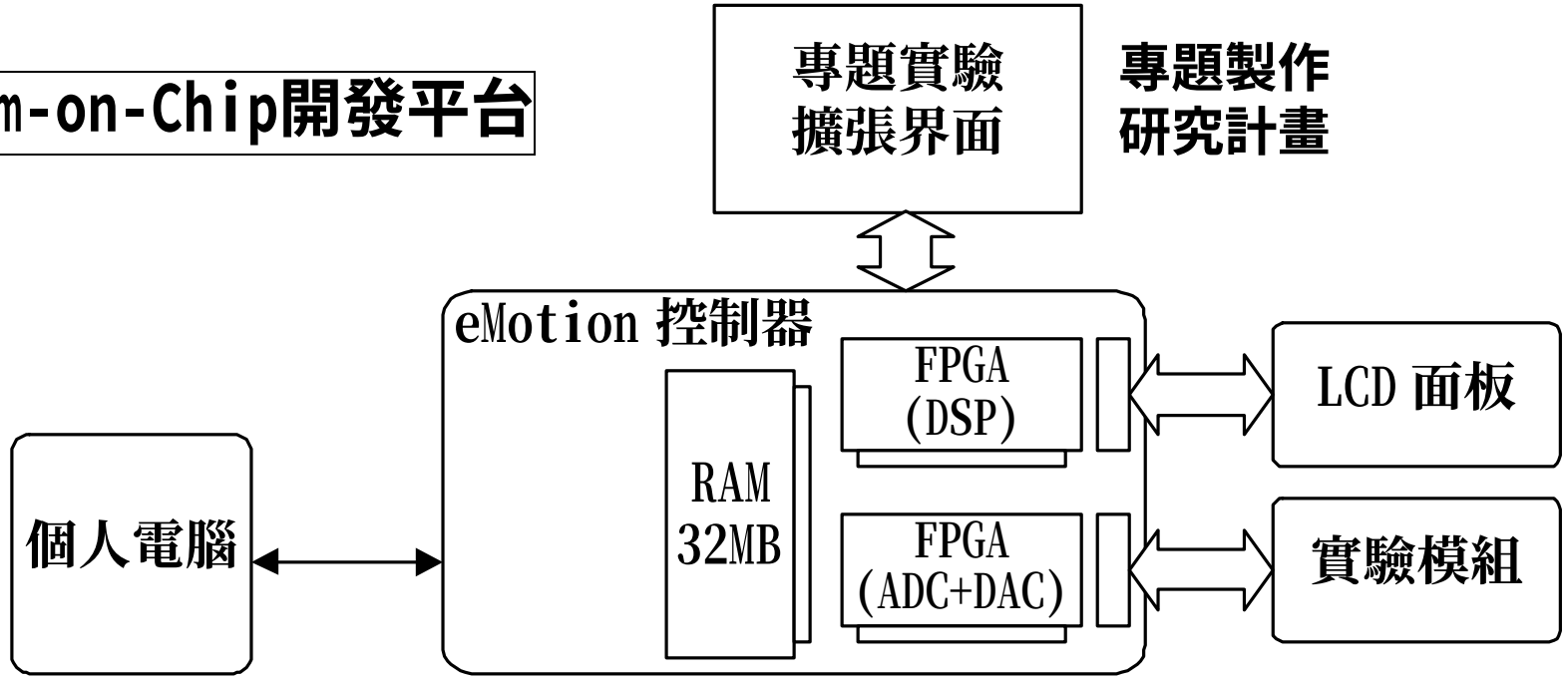
軟體設計部分

第六章：組合語言基礎
第七章：組合語言結構
第八章：組合語言定義
第九章：模組程式設計
第十章：系統程式設計


硬體開發技術
軟體開發技術
電晶體元件
MSI/LSI/VLSI
FPGA 可程式邏輯
單晶片控制器
PC 級控制器
DSP 控制器
System-on-Chip
(SoC)


System-on-Chip
System
on-Chip
數位系統 通訊系統
家用系統 控制系統
工業系統 電腦周邊
邏輯設計 半導體製程
界面設計 即時控制軟體
CPU 設計 網路連線

System-on-Chip開發平台



VHDL開發環境	Linux作業環境	邏輯電路設計	邏輯實驗
C/Java開發環境	TCP/IP網路界面	DSP核心設計	通訊實驗
MatLab開發環境	DSP軟體編譯程序	即時軟體開發	控制實驗
遠端網路監控	測試與監控環境	周邊界面電路	電機實驗

應用系統規劃

硬體設計

模組配置

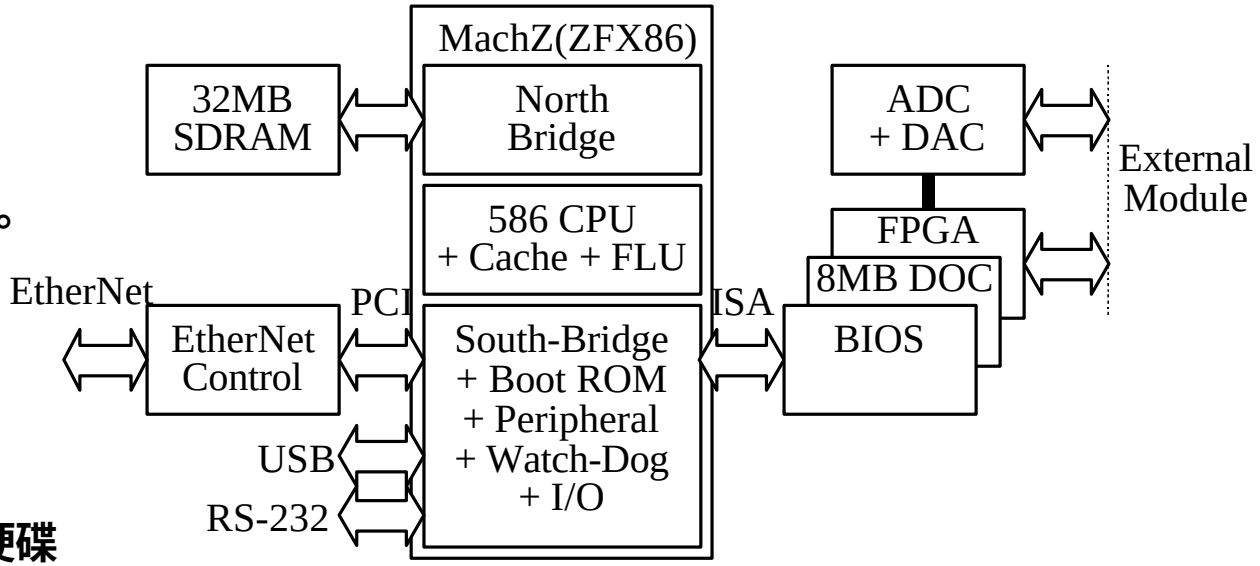
軟體規劃

應用規劃

內藏式PC控制器

ZF-Linux公司的MachZ(ZFX86)

1. 100MHz的586CPU，耗電0.5W。
2. 現成的BIOS碼和參數設定工具。
3. 外加元件極少。
4. 研發過程中可利用RS-232、USB及Ethernet界面連接。
5. 模組中併入FPGA和ADC/DAC。
6. 全部電路板尺寸約10cm*20cm。
7. 程式和資料儲存在8MB的單晶硬碟(Disk-On-Chip，簡稱DOC)中。



FPGA電路

Xilinx公司Virtex-II系列的XC2V250：

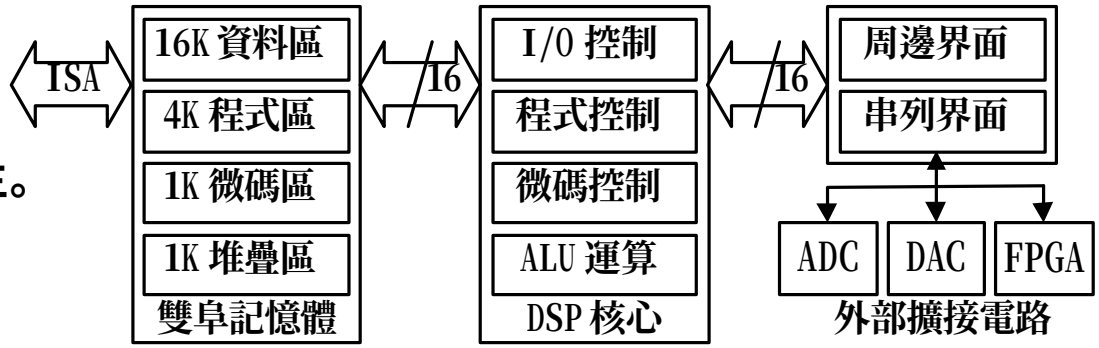
1.閘數(Gate Count)為250K，內部電源為1.5V，可允許420MHz的脈波頻率。
2.邏輯閘單位為CLB，每個CLB包含四組Slice，每個Slice中都有一組D型正反器和4位元可程式邏輯。
3.XC2V250中有24*16個CLB，相當於1536個Slice。
4.擁有獨立的雙阜RAM，容量為18Kbit或是2Kbyte。XC2V250中共有24個雙阜RAM，容量共48KB。
5.擁有硬體乘法器，為18*18=36的整數乘法器。XC2V250中共有24個硬體乘法器。
6.XC2V250中最多可同時擁有16組的公用Clock，具備硬體連線。
7.就繞線能力極佳，幾乎不會繞線失敗。
8.擁有200個I/O點，每個I/O點都可規劃成不同型式的邏輯準位。

Xilinx公司Spartan系列的XCS05XL：

1.閘數(Gate Count)為5K，內部電源為3.3V，可允許80MHz的脈波頻率。
2.邏輯閘的單位為CLB，每個CLB包含兩組D型暫存器和4位元的可程式邏輯。
3.XCS05中有10*10個CLB，相當於200組暫存器。
4.最多可同時擁有4組的公用Clock，具備硬體連線。
5.擁有72個I/O點，每個I/O點都可規劃成3.3V的LVTTL邏輯準位。

內藏式DSP核心

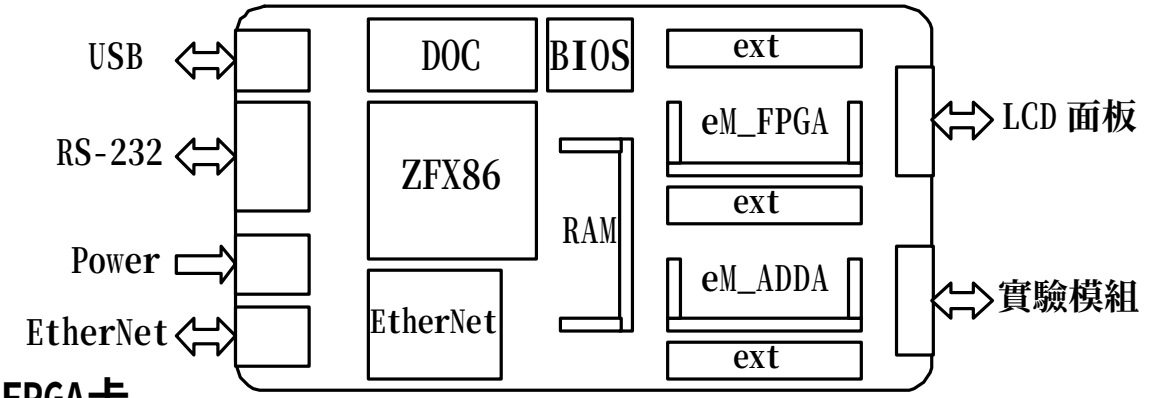
1.不參考任何現成的設計。
2.組合語言規劃以簡單、易讀為目標。
3.DSP必須搭配586CPU，以即時控制為主。
4.DSP和586CPU以雙阜記憶體連接。
5.目標是執行32KHz的即時控制程式。



功能上：

1.就586CPU而言，記憶體佔用ISA-bus的8KB位址，總共48KB的雙阜記憶體規劃成24頁。
2.就DSP而言，雙阜記憶體規劃成資料區、程式區、微碼區和堆疊區四個部分，除了微碼區為32-bit外，其他部分都是16-bit(1-word)寬度。
3.不論是DSP或586CPU都不准直接控制I/O界面。所有I/O界面都以DMA方式和記憶體連接。
4.外部擴接電路(包括ADC、DAC和另一顆FPGA)都是由I/O控制器透過串列界面連接。

eMotion控制器



配置包括：

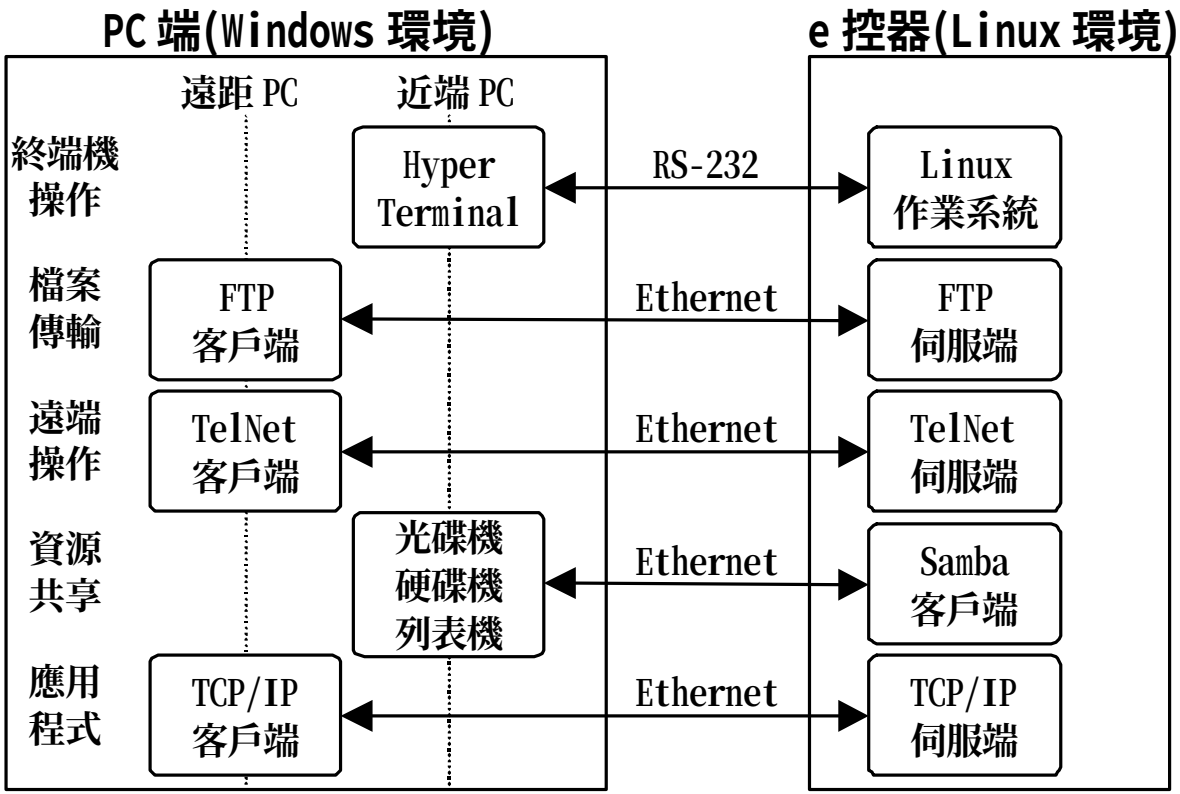
- **ZFX86：** 586CPU
- **EtherNet：** 網路以PCI-bus連接CPU
- **DOC：** 8MB單晶硬碟。
- **BIOS：** 256KB的容量。
- **RAM插槽：** 插上64MB的RAM卡
- **DSP插槽：** 插上內含DSP核心的eM_FPGA卡
- **ADDA插槽：** 插上包括ADC/DAC的eM_ADDA卡
- **ext擴張座：** 3組50P的插座，可作為彈性擴張用，以處理不同的專題實驗。
- **實驗接頭：** 2組26P的插座，可連接兩組的實驗模組，通常一組固定為LCD面板。

軟體開發環境

e控器：提供硬體控制和軟體管理等控制功能。

近端PC：提供軟硬體開發過程中的開發工具和作業環境。

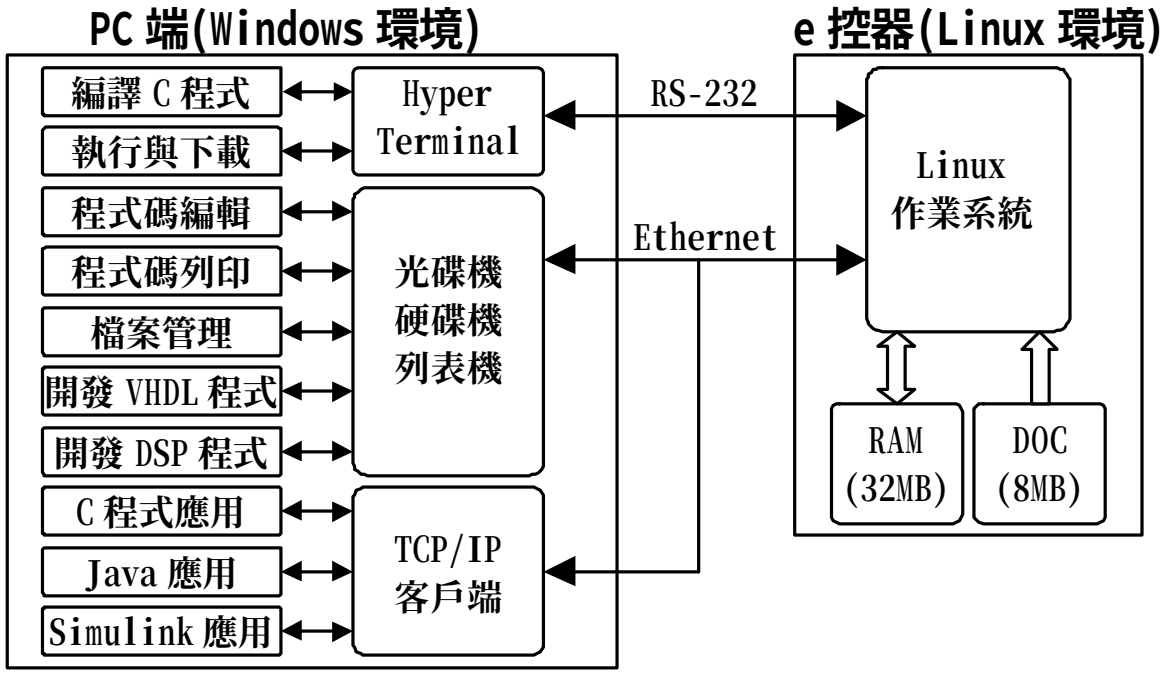
遠端PC：提供叫學和產業應用時的遠端監控功能。



視窗操作環境

1.執行上採用Linux系統
2.操作上採用視窗作業環境

開機程序：
1.DOC檔案解壓縮並下載到RAM中
2.檔案的讀寫或管理都在RAM中
3.DOC儲存的檔案都是唯讀檔案
4.使用者程式儲存在PC端的硬碟
5.PC端在視窗環境直接處理檔案
6.光碟和印表機在視窗下處理。

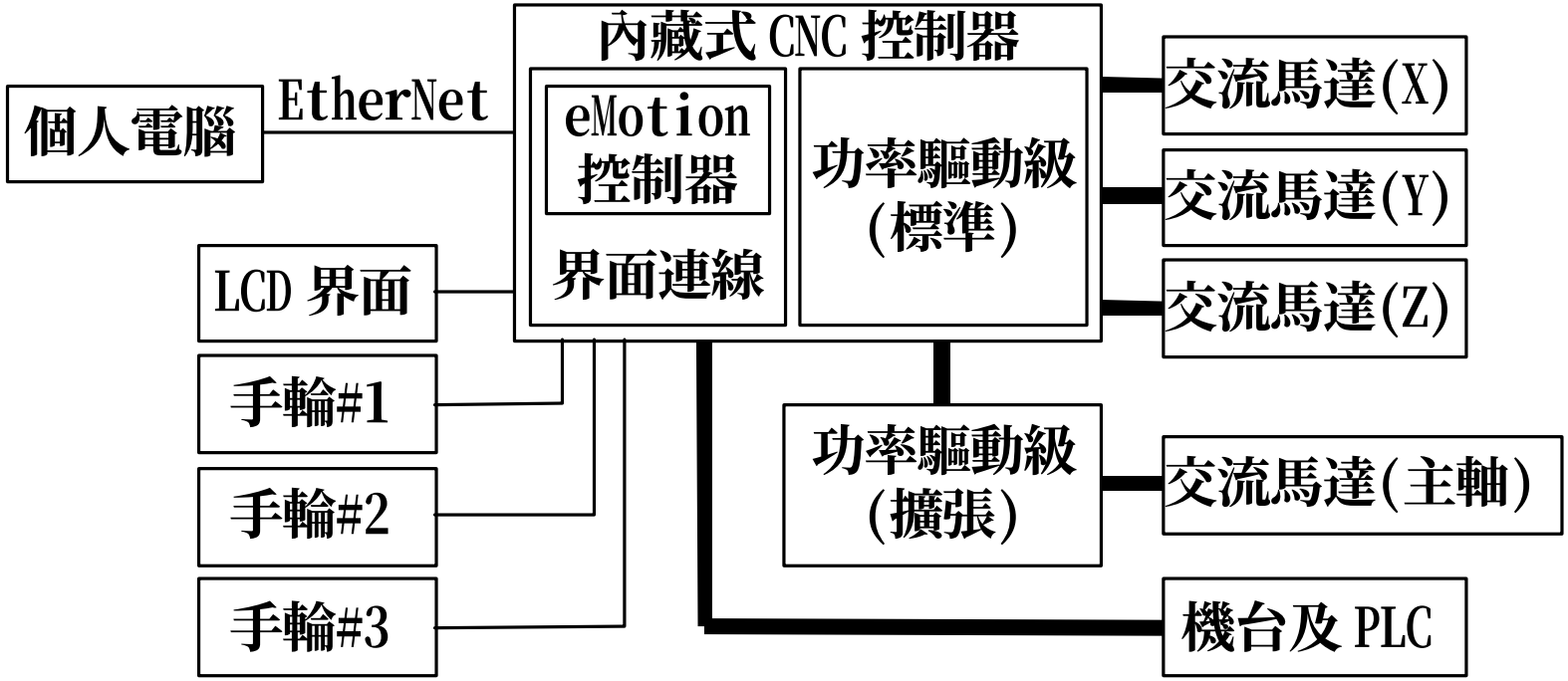


項 目	說 明
C程式開發 (e控器部分)	1. 直接在視窗中編輯程式碼 2. 在終端機中下達編譯指令 3. 在終端機中執行下載指令
DSP程式開發 (e控器部分)	1. 直接在視窗中編輯程式碼 2. 在終端機中下達編譯指令 3. 在終端機中下載並執行
VHDL程式開發 (e控器部分)	1. 直接在視窗中編輯程式碼 2. 直接在視窗中下達編譯指令 3. 在終端機中下載並執行
C程式開發 (PC端部分)	所有程序都可在視窗環境處理
JAVA程式開發	所有程序都可在視窗環境處理
Matlab開發	所有程序都可在視窗環境處理
Simulink開發	所有程序都可在視窗環境處理
應用程式執行	所有程序都可在視窗環境處理

即時控制環境

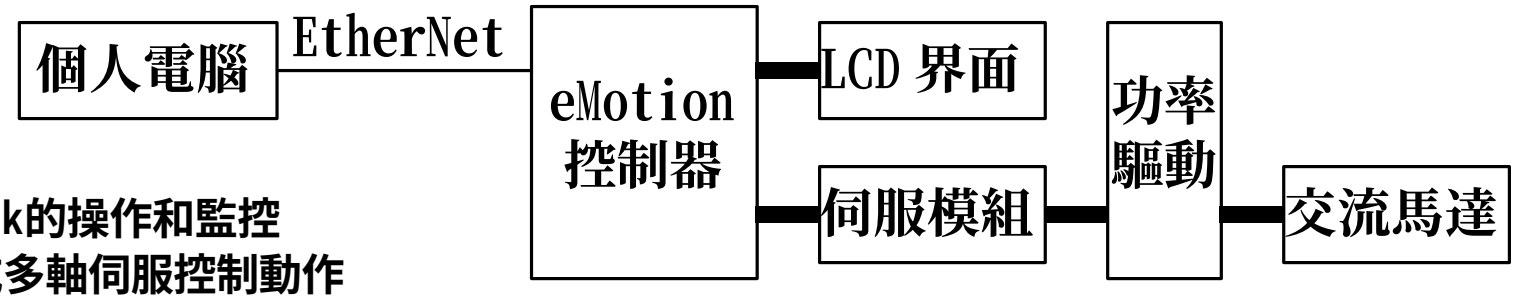
1.DSP核心部分負責即時控制程式，抽樣控制頻率可高達32KHz，完全不受Linux作業系統干擾。
2.Linux作業系統下的586CPU，掌管主控程式和TCP/IP通訊程式，只能執行100Hz左右的抽樣控制。
3.所謂100Hz左右，是指基本頻率是100Hz，但是多少會受到其他程式負載所影響，不能完全保證。
4.586CPU和DSP透過雙阜記憶體共享所有的參數和變數。ISA-bus的傳訊速度限制可忽略不計。
5.PC端應用程式可執行人機圖控和TCP/IP通訊界面，執行遠距操作的功能。
6.PC端和586CPU之間透過網路做連線。EtherNet在輕負荷下可執行100Hz的資料傳輸(200-byte左右)
7.若是透過網際網路做遠距連線，則只能傳遞資料而不能保證傳訊速度了。

四軸CNC控制器



1.PC端負責操作界面和遠端監控
2.e控器負責CNC的整個控制動作，即時控制負載由586CPU和DSP核心來分擔。
3.e控器直接整合在功率驅動級中，形成單一模組的內藏式CNC控制器。
4.可連接工業級PC做標準型的CNC控制器，也可連接簡單的LCD面板做簡易操作型的CNC控制器。

電動機控制實驗



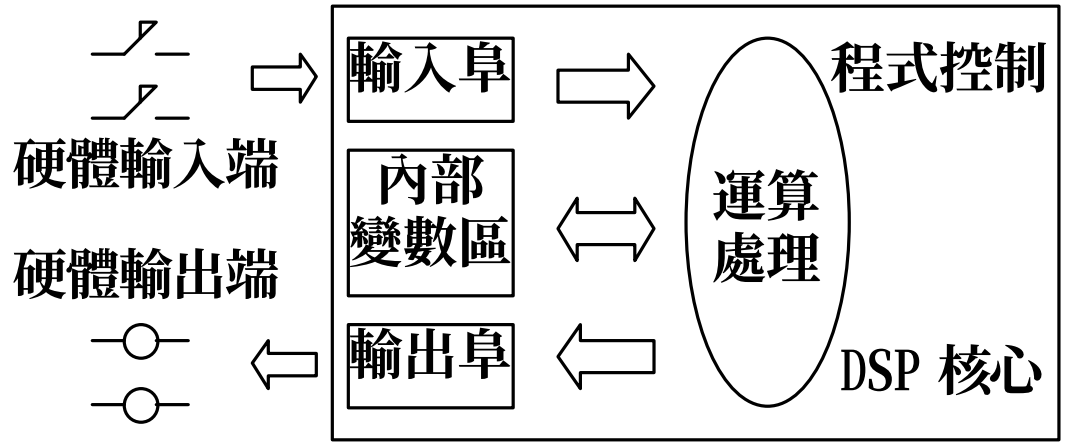
1.PC端負責Simulink的操作和監控
2.e控器負責單軸或多軸伺服控制動作
3.586CPU只負責網路資料的傳遞，所有控制都以模組方式架構在DSP核心中。

變頻器模式實驗　(開環路電壓控制)
步進馬達模式實驗(閉環路電流控制)
直流馬達模式實驗(閉環路扭力控制)
速度控制模式實驗
位置控制模式實驗
加減速控制與前置補償
電流過載與PID補償效應

控制程式規劃

系統程式規劃
模組程式規劃

程式控制

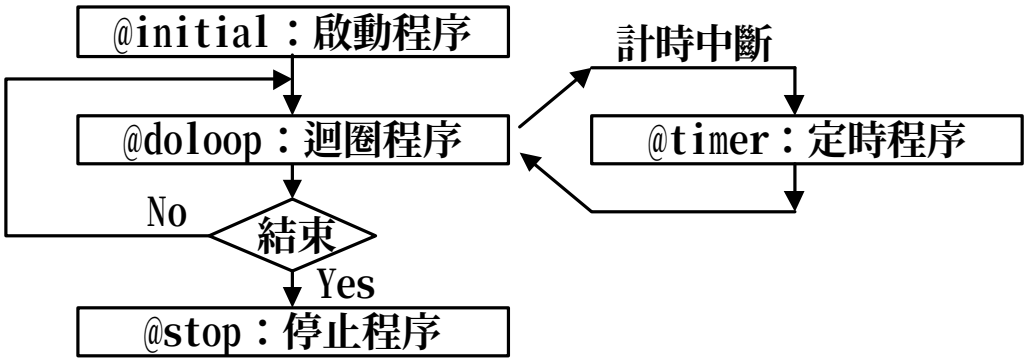


程式控制的基本架構：

1.DSP核心以輸入阜連接硬體輸入端。輸入阜是特定記憶體，可將硬體輸入端的狀態反應在記憶體中。
2.DSP核心以輸出阜連接硬體輸出端。輸出阜也是特定的記憶體，可將對應的硬體輸出端更新。
3.程式控制，就是處理輸入阜和輸出阜之間的運算關係。根據輸入阜的狀態，改變輸出阜的狀態。
4.為了執行這些運算處理，往往需要額外的記憶體來儲存其內部變數，就是其中的內部變數區。

換句話說：DSP核心的程式控制，就是記憶體的特殊處理程序。

即時控制



即時控制的程序包括：

- @initial程序：準備啟動條件
- @stop程序　：準備安全的停止條件
- @doloop程序：較慢或不重要的控制程序，以迴圈方式執行。
- @timer程序：較快以及重要的控制程序，以計時中斷執行。

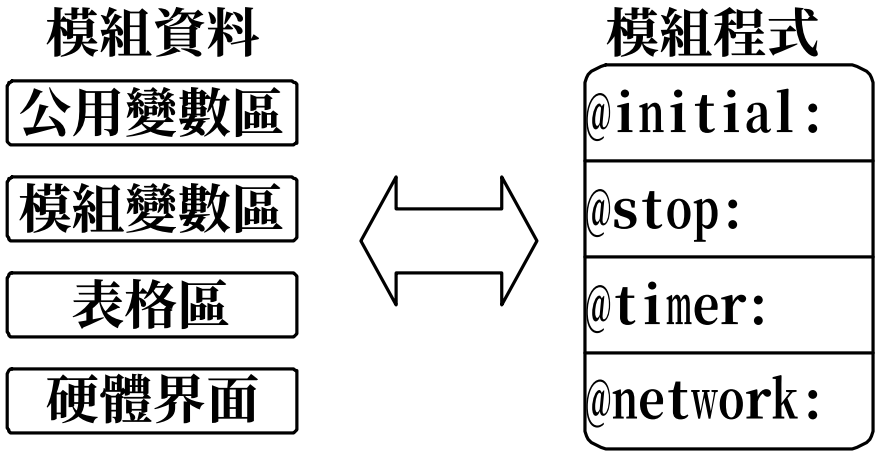
就時間軸而言:

- @timer程序是以固定頻率執行
- @doloop程序則是利用剩下的時間空檔來執行

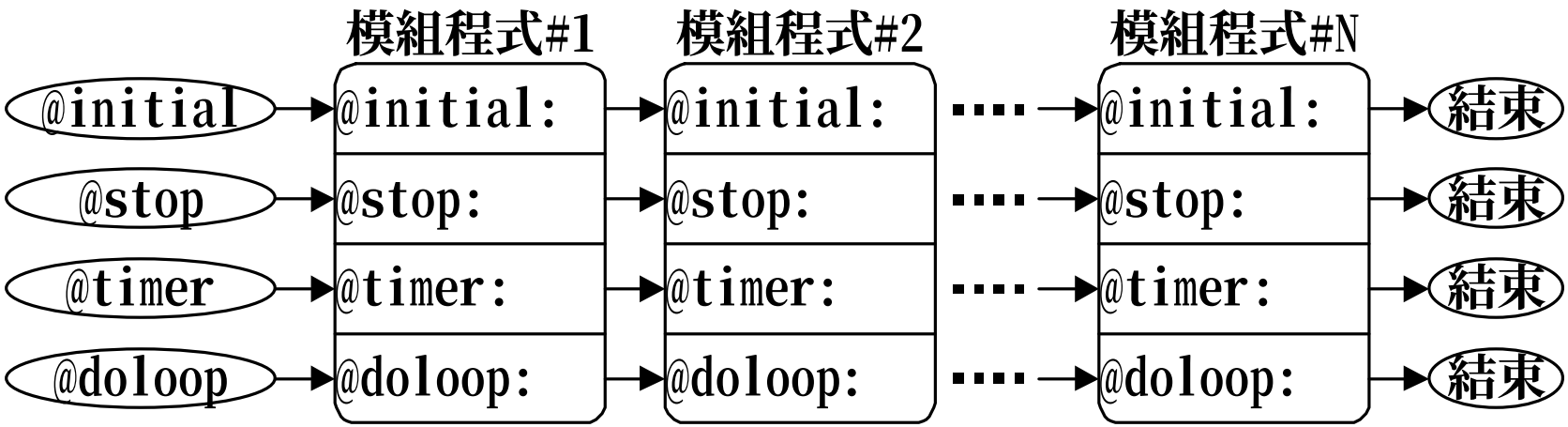
模組架構

就控制架構而言，模組程式的系統方塊圖如上：

1.模組程式可允許多組的輸入點，可以用來連接其他模組的輸出點。
2.模組程式可允許多組的輸出點，可以用來連接其他模組的輸入點。
3.模組程式中可以擁有內部變數和表格，用以執行指定的計算或是控制工作。

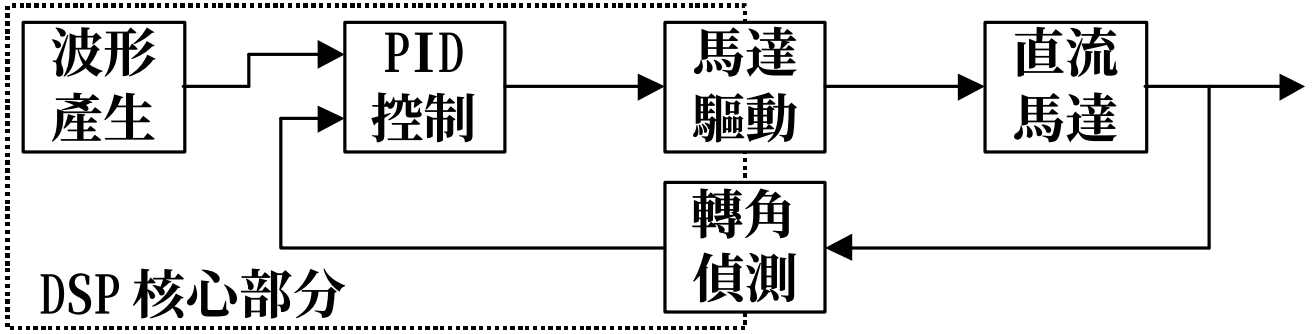


系統執行程序

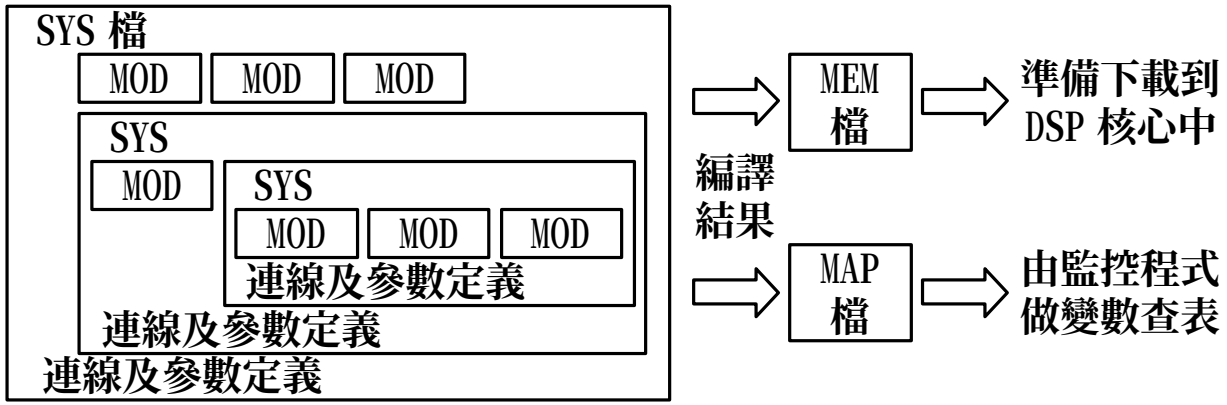


1.每個程序都是副程式結構，而以RET指令做結束。
2.當系統程式連結多個模組時，就會自動的依序呼叫各模組中的相對程序。
3.以@timer程序為例，當DSP核心週期性的產生@timer需求時，
系統程式就會先執行模組程式#1中的@timer程序，
接著再依序執行其他模組程式中的@timer程序。
最後在系統程式中結束，並將控制權交還給其他程序。

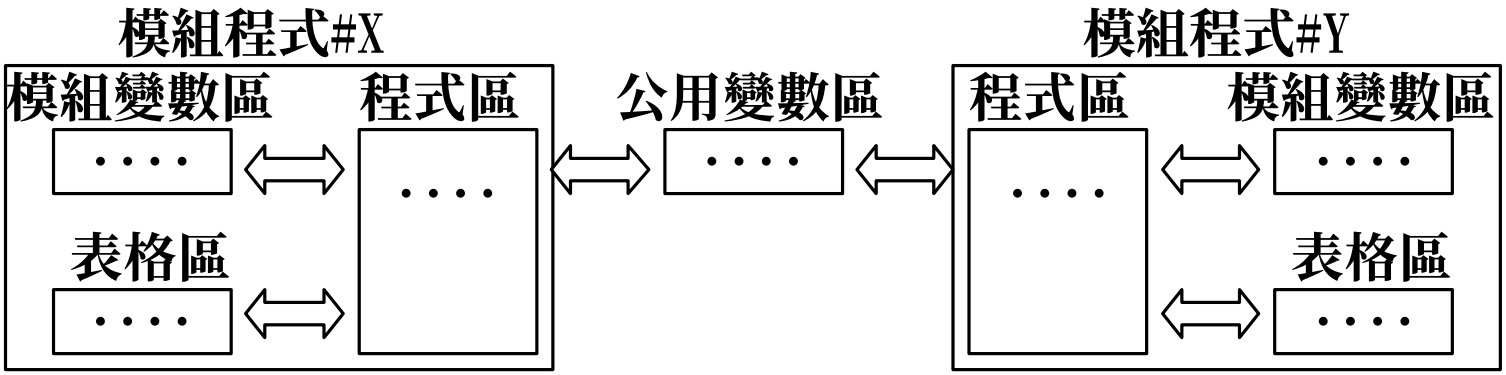
控制方塊圖



1.控制系統通常以方塊圖的形式來表示。
2.整個系統中依照功能和連接關係，通常又細分成許多較小的模組。
3.每個模組本身有輸入點和輸出點，通常也有一些內部變數和參數。
4.將各模組的輸入輸出連接清楚後，
系統方塊圖即定義完成。



模組的資料結構



模組程式的資料結構包括：

1.每個模組程式都有自己專屬的模組變數區，其中的變數只有這個模組程式能讀寫。

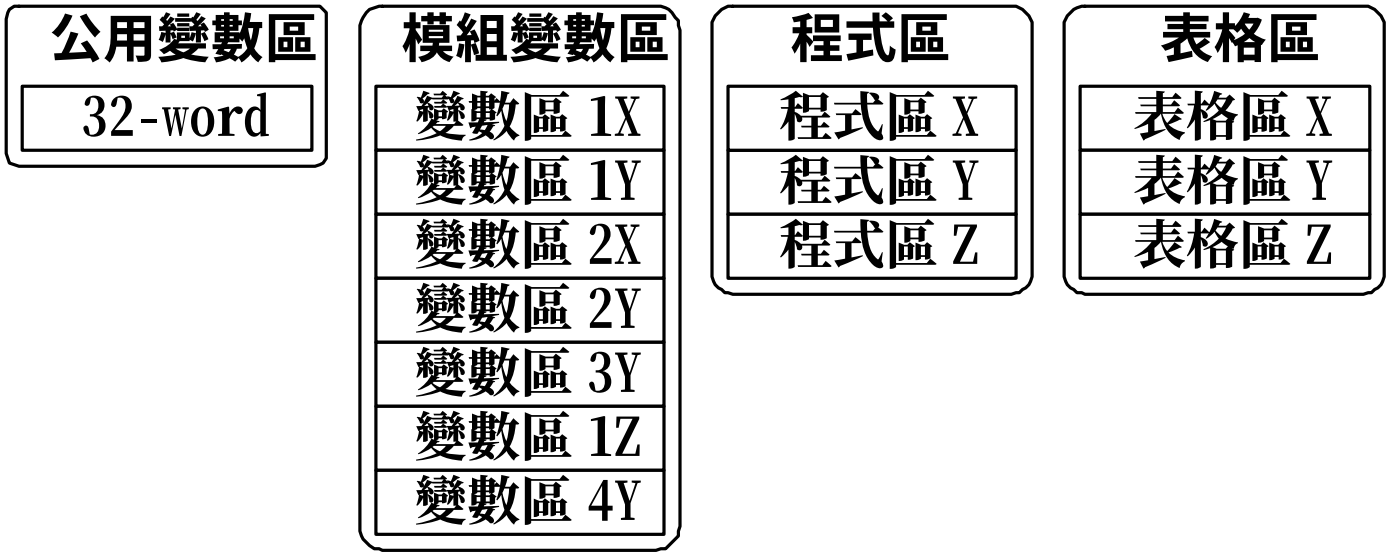
2．模組程式中也有自己專屬的表格區，其中的資料也只有這個模組程式讀寫。

3．模組程式還可讀寫公用變數區的資料，公用變數區由所有模組程式所共享，任何模組都可使用。

4．另外，模組程式包括@initial、@stop、@timer和@doloop四個程序，由系統程式的呼叫來執行。

系統的資料結構

1.系統程式可以重複的使用各個模組。
2.就模組變數區而言，每個模組都有獨立的模組變數區，不論這個模組是否為重複使用。
3.就程式區而言，每個模組只儲存一組。
4.就表格區而言，每個模組只儲存一組。



系統程式(sys檔案)

系統程式是以文書檔的型式存在，副檔名固定為".sys"，程式架構如下：

```
@module    .....;                    //定義系統中所需的模組
@sequence  .....;                    //定義系統中模組的執行順序
@variable  .....;                    //定義系統中的新變數
<var> = <express>;                   //定義各參數和變數的內容
```

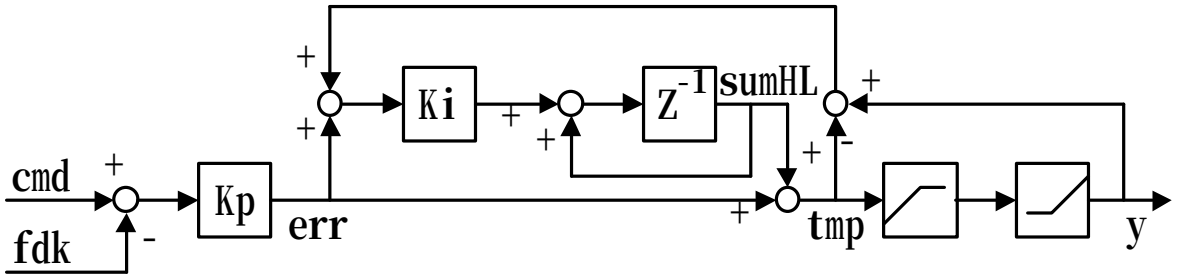
1.以@module指令定義系統中所需要的模組。
2.以@sequence指令定義各模組的執行順序。
3.以@variable指令定義系統中的新變數，提供監控程式或高階層檔案做運用。
4.以var=express的格式設定各參數和變數。

模組程式(asm檔案)

模組程式是以文書檔的型式存在，副檔名固定為".asm"，程式架構如下：

```
@global   .....;              //定義公用變數
@local    .....;              //定義模組變數
@data     .....;              //定義模組變數區的長度
@table    .....;              //定義表格區的長度和內容
@initial: .....               //@initial副程式，設定啟始值
          RET;                //以RET指令結束副程式
@stop:    ....                //@stop副程式，設定安全的停止狀態
          RET;                //以RET指令結束副程式
@timer:   ....                //@timer副程式，週期性的即時控制程式
          RET;                //以RET指令結束副程式
@doloop: ....                 //@doloop副程式，較不重要的控制程式
          RET;                //以RET指令結束副程式
```

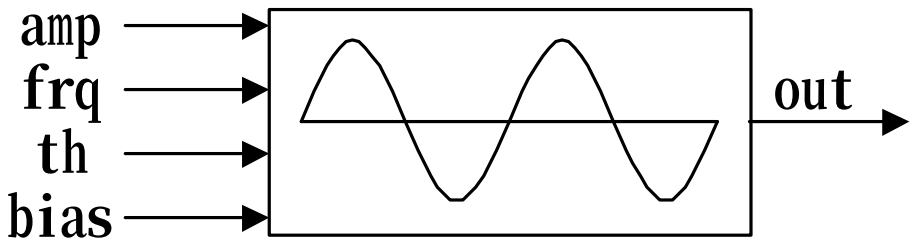
PI(比例積分)控制器



模組程式範例：PI_CONTROL.asm

```
....                                                        //宣告公用變數和模組變數
@initial:  .... RET;                                        //啟動程序，以RET指令結束
@timer:                                                     //即時控制程序
     RD.0 seq; XOR axis;  AND #3; JNZ b9;                   //1KHz抽樣頻率模式
     RD (cmd); SUB (fdk); MUL kp; SFR 12; WR err;           //err=(cmd-fdk)*kp/4096
     ADD sumH; WR  tmp;                                     //tmp=err+sumH
     MIN high; MAX low;   WR out;                           //out=limit(tmp,high,low)
     SUB tmp;  ADD err;   MUL ki; SFR 12;                   //ACC=(out-tmp+err)*ki/4096
     SGN;      ADD sumL;  WR sumL;                          //sumHL+=ACC
     RD.H;     ADC sumH;  WR sumH;
b9:  RET;
```

弦波產生器



模組程式範例：SIN_WAVE.asm

```
....                                                    //宣告公用變數和模組變數
@initial:  .... RET;                                    //啟動程序，以RET指令結束
@timer:                                                 //即時控制程序(8KHz)
     RD  ang;   ADD (frq); WR ang;                      //ang+=freq
     SFR 8;     ADD (th);  AND #255;                    //ACC=sin(ang/256+th)
     ADD tab;   WR.H;      RD.M;                        //table-look-up
     MUL (amp); SFR 12;;   ADD (bias); WR out;          //out=ACC*amp/4096+bias
     RET;
@table tab 256;                                         //sin[256]表格區，1.0=4096
....                                                    //宣告表格內容
```

組合語言規劃

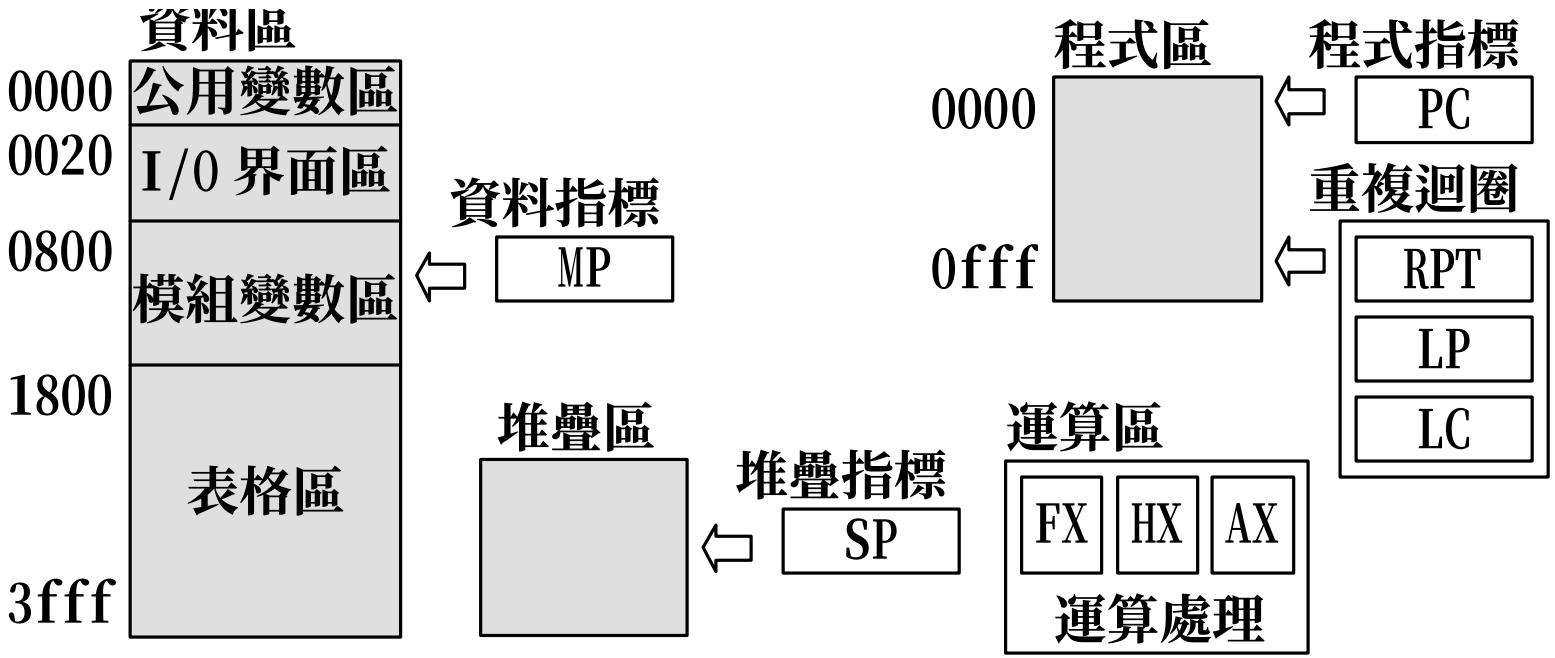
程式管理指令
資料搬移指令
數值計算指令

組合語言指令

DSP核心中的組合語言以簡短易讀為目標，所以指令數目只有29組。

資料設定指令	RD	WR		
數值運算指令	ADD	ADC	SUB	SBC
數值運算指令	MUL	NEG	ABS	SGN
移位運算指令	SFR	SFL		
邏輯運算指令	AND	OR	XOR	COM
運算保護指令	MAX	MIN	PRT	
堆疊控制指令	PUSH	POP		
程式控制指令	JMP	JR	CALL	RET
程式控制指令	NOP	HALT		
迴圈控制指令	RPT	LOOP		

DSP核心結構



1.記憶體分成資料區、堆疊區和程式區三部分，資料區為16K-word。
2.堆疊區為1K-word，由指標SP來控制。
3.程式區為4K-word，程式的執行位址由指標PC來控制。
4.為應付重複迴圈的需求」，另外提供了三個系統變數：RPT值、LP值和LC值。
5.運算區共有AX暫存器、HX暫存器和FX暫存器。

跳躍指令

跳躍指令群只有一個JR指令，但是卻包括了許多不同的比較條件，包括：

JR.C	JC			JR.C 指令
JR.NC	JNC			JR.NC指令
JR.Z	JZ	JR.EQ	JEQ	JR.Z 指令
JR.NZ	JNZ	JR.NE	JNE	JR.NZ指令
JR.N	JN	JR.LT	JLT	JR.N 指令
JR.P	JP	JR.GE	JGE	JR.P 指令
JR.V	JV			JR.V 指令
JR.NV	JNV			JR.NV指令
		JR.LE	JLE	JR.LE指令
		JR.GT	JGT	JR.GT指令

JR指令相當於C程式中的if指令，例如：

```
     RD x; SUB y; JGT tend; RD z; WR x;    //相當於C程式的 if (x<=y) x=z;
tend:                                      //繼續執行下一步程式
```

跳躍指令

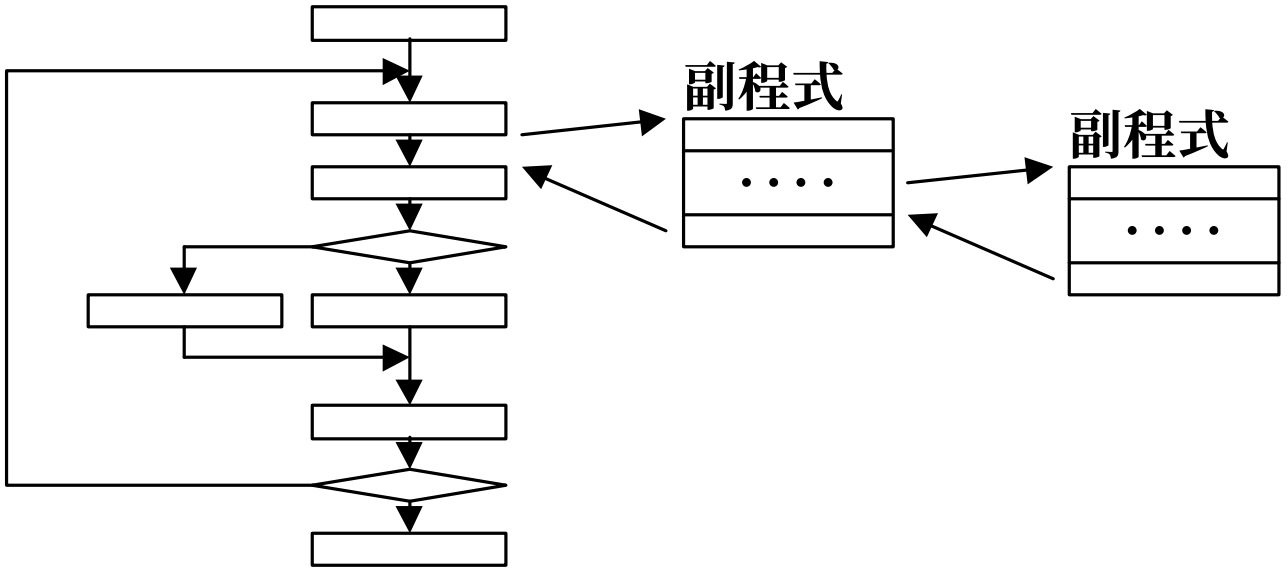
JR.H指令專門處理查表跳躍工作，像C程式中的switch指令。

```
     switch (idx)
     {
     case 0 : .....; break;          //若idx=0則執行此部分
     case 1 : .....; break;          //若idx=1則執行此部分
     ...
     case N : .....; break;          //若idx=N則執行此部分
     }
```

組合語言雖然無法執行像switch()這麼高階的指令，但是也可以做些近似的動作。例如：

```
     RD idx; MUL #5; WR.H; JR.H tab;  //跳到tab+(idx*5)的位址繼續執行
tab: ***; ***; ***; ***; JR tend;     //若idx=0則執行此部分，長度固定為5-word
     ***; ***; NOP; NOP; JR tend;     //若idx=1則執行此部分，長度不足時以NOP補齊
     .....
     ***; ***; ***; ***; JR tend;      //若idx=N則執行此部分，長度固定為5-word
tend:
```

副程式指令



1.一般而言，程式是依序執行的，不需要特別的指定。
2.當程式要選擇性執行，就必須執行程式跳躍(JR)的動作。
3.有時需要呼叫副程式，這時就需要跳到副程式中執行；副程式執行完畢才跳回主程式中繼續執行。
4.副程式中還可以呼叫其他副程式，形成階層式的程式結構。
5.副程式以CALL指令執行呼叫，以RET指令結束。

單指令迴圈

C程式中的迴圈指令非常好用，例如：

```
for (i=0,sum=0; i<20; i++) sum+=x[i];      //sum為x(0~19)的累加結果
```

組合語言中雖然沒有for()那麼高階的指令，我們也設法加入近似的功能。

```
RD #0; RPT #20; ADD x; WR sum;              //sum為x(0~19)的累加結果
```

1.RPT #k指令是將下一筆程式自動的執行k次。
2.如果只是單純的將ADD x執行20次，那麼只是sum=x*20的動作，沒有必要。
3.所以實際上並不是執行sum=x*20的動作，而是執行sum=x(0)+...+x(19)的動作。

ADD x指令的細部執行動作如下：

```
ADD    reg;  #1.PC++; DP=MP+LC+reg+0x800    LC為迴圈參數，預設=0
clk=1+n      #2.AX=AX+(DP)                  實際運算動作
             #3.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

多指令迴圈

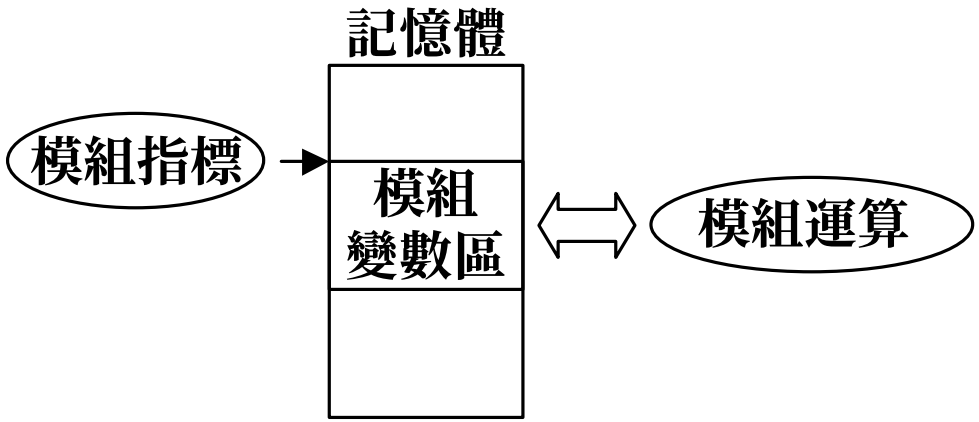
迴圈中只有一個指令時可用RPT指令。如果迴圈中有兩個以上的指令時，就必須用LOOP指令了。

```
WR.LP #4;          //設定迴圈要執行四次
.....              //真正的迴圈執行區域
LOOP;              //迴圈結束點，k值不用特別定義，由編譯程式自動計算
```

為了使得每次迴圈執行時的參數有所不同，LOOP指令並不是將指令執行N次而已。詳細變化說明如下：

1. 在DSP核心中，為了配合LOOP指令，搭配了一個LC計數器(Loop Counter)。
2. LC計數器的初值固定為(LC=0)，每次LOOP迴圈執行後會自動加一(LC++)。
3. 同時將DP指標的計算公式定義為(DP=MP+LC+x+0x800)。
4. 一旦加上LOOP迴圈後，因為LC的遞增結果，每次LOOP執行時就不同了。
5. 例如(ADD x)指令，
   在LOOP第一圈時執行(DP=MP+0+x+0x800)的動作，
   在第二圈時執行(DP=MP+1+x+0x800)的動作，依此類推。
   相當於模組變數x之後的連續區域處理。
6. 只要LOOP迴圈一結束，立刻重設(LC=0)，以免影響迴圈之外的指令。

模組變數區的讀寫



為了使組合語言更精簡，在DSP核心中定義了模組變數區，其中：

1.每一個程式模組都有自己的模組變數區，也只能運作這個模組變數區的資料。

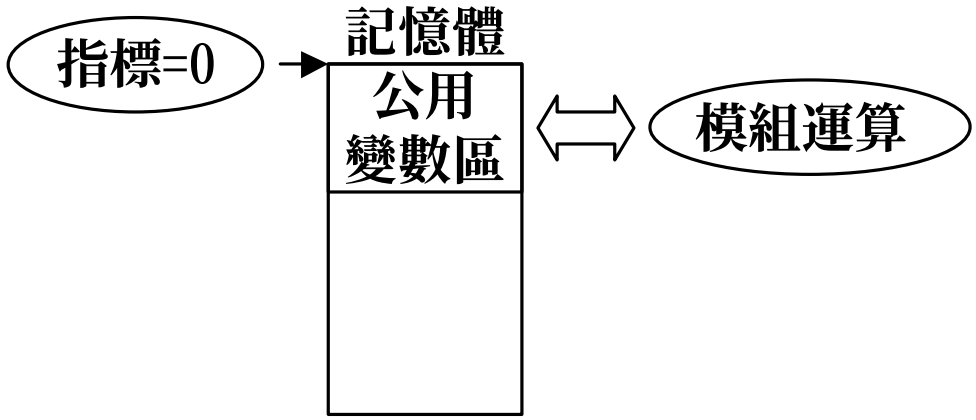
2.和整個記憶體的容量相比，模組變數區的容量是小多了，在DSP核心中：

記憶體容量是16384-word，需要14-bit的地址來選擇其中的變數。

模組變數區的容量是32-word，只需要5-bit的地址來選擇其中的變數。

3.模組變數區以模組指標(MP)來定義在整個記憶體中的特定區段。只要MP指標改變，就會對應到不同的模組變數區。公式為(DP=MP+LC+reg+0x800)。

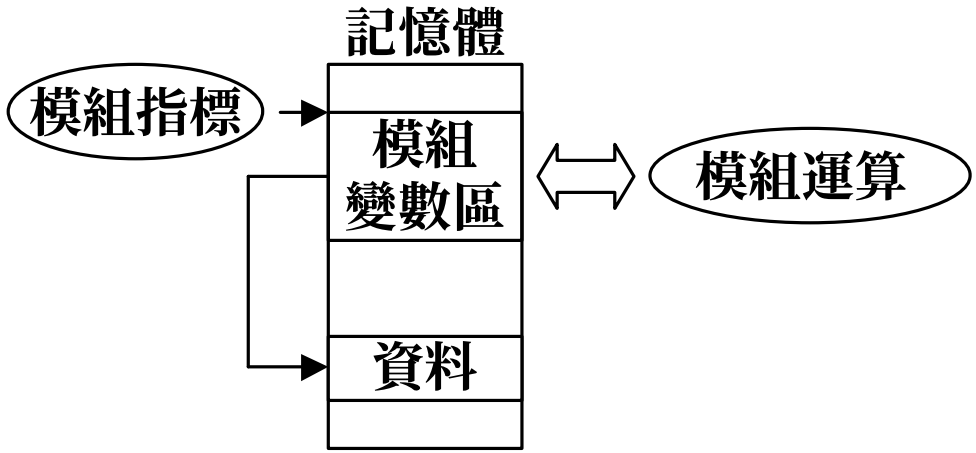
公用變數區的讀寫



在DSP核心中也定義了公用變數區。組合語言只需要5-bit就能定義公用變數區中的變數。其中：

1.所有程式都可運用公用變數區，其指標固定為0，就是放在記憶體的最前端。
2.公用變數區的容量是32-word，只需要5-bit的地址來選擇其中的變數。
3.公用變數區中通常儲存了系統變數，供所有的模組使用。例如系統時鐘，儲存目前的執行次數，提供給所有模組作參考。
4.公用變數區的計算公式為(DP=reg)。

間接讀寫模式



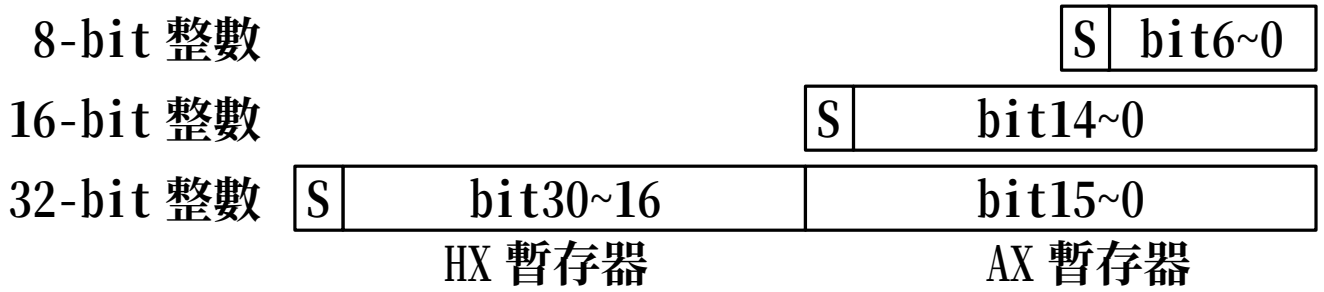
當要處理模組變數區之外的資料時，可以間接處理模式來執行。說明如下：

1.將模組變數區的變數視為指標，指到特定的記憶體位址，在將這個位址的資料取出作處理。
2.在DSP核心中，變數的長度是16-bit，可以選擇的記憶體範圍為65536-word，超過記憶體範圍。
3.而在組合語言中，只需佔用5-bit的機械碼空間就能選擇所有記憶體中的資料。
4.間接讀寫的用途包括：
   A.模組的輸入點連線。
   B.表格的讀寫。
   C.硬體界面的絕對位址讀寫。

數值運算指令

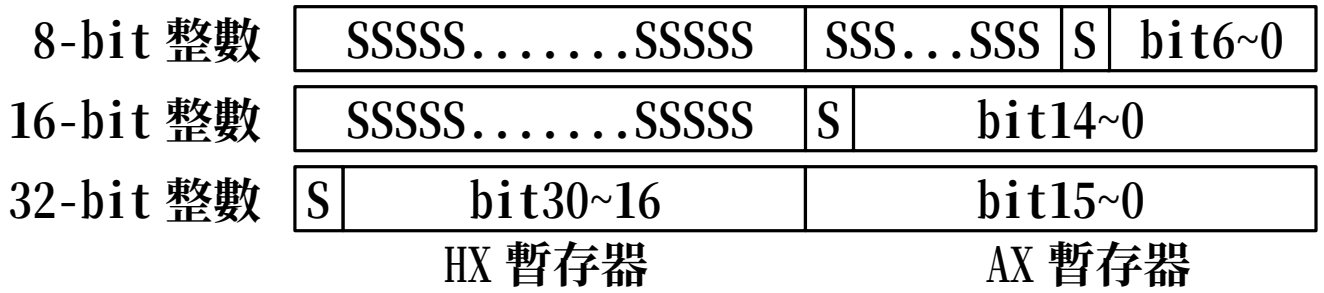
指令	分 類	運算結果						指 令 說 明
		HX	AX	VF	NF	ZF	CF	
AND	邏輯運算	--	◎	--	◎	◎	--	AX= AX.and.R
OR	邏輯運算	--	◎	--	◎	◎	--	AX= AX.or. R
XOR	邏輯運算	--	◎	--	◎	◎	--	AX= AX.xor.R
COM	邏輯運算	--	◎	--	◎	◎	--	AX= not.AX
ADD	加法運算	--	◎	◎	◎	◎	◎	AX= AX+R
ADC	加法運算	--	◎	◎	◎	◎	◎	AX= AX+R+CF
SUB	減法運算	--	◎	◎	◎	◎	◎	AX= AX-R
SBC	減法運算	--	◎	◎	◎	◎	◎	AX= AX-R-CF
MUL	乘法運算	◎	◎	--	◎	◎	--	HXAX= AX*R
SFL	移位運算	--	◎	--	◎	◎	--	AX= AX<<bit
SFR	移位運算	--	◎	--	◎	◎	--	AX= HXAX>>bit
MAX	比較運算	--	◎	--	◎	◎	--	AX= max(AX,R)
MIN	比較運算	--	◎	--	◎	◎	--	AX= min(AX,R)
ABS	絕對運算	--	◎	--	◎	◎	--	AX= abs(AX)
NEG	負數運算	--	◎	--	◎	◎	--	AX= -AX
SGN	延伸運算	◎	◎	--	◎	◎	--	HXAX=sign(AX,bit)
PRT	過溢保護	--	◎	--	--	--	--	AX=prot(VF)

符號延伸指令



符號延伸指令：

```
SGN   bit;      ;HXAX=sign(AX,bit)的正負號延伸指令，其中bit的範圍為1~15
```



過溢保護

所謂過溢保護，就是在計算錯誤的狀況下，尋求最佳的妥協之道。
例如(20480+24576=45056)，其結果的45056已經超出+/-32767的正整數極限。

妥協之道有兩個方式：
1.顯示過溢警報並將程式停掉，以免使用錯誤結果繼續計算而一錯再錯。
2.不顯示過溢警報而程式也繼續執行，但是卻將計算結果改成32767的極限值。

過溢保護指令：

PRT.F	：一旦VF=1則依NF的結果重設AX暫存器，但過溢時NF的指示錯誤，所以 若NF=0則設定AX=0x8001 (相當於-32767)，而 若NF=1則設定AX=0x7FFF (相當於+32767)。
PRT.O	：一旦VF=1則重設AX暫存器為零(0x0000)。

0x50001234+0x60000000=0xB0001234：因為過溢現象必須更正結果為
0x7FFF0000

乘法運算

對於16位元的加減運算，只要對過溢現象稍加處理，即可維持16位元的計算結果。
但是16位元的乘法運算，即使在正常狀況下也會造成32位元的計算結果。

這時有兩種選擇：
1.放棄其中的16-bit，仍然以16-bit儲存結果。
2.保留所有的32-bit。

前種選擇造成精度的嚴重損失，所以好像第二種方式較為正確，但是實際上有困難。
因為以32位元儲存結果後，接下去的所有運算(不論加減乘除)都要變成32位元。
要是再碰到32位元乘法時，難道要將結果再擴張成64位元嗎？

所以放棄其中的16位元，而用剩下的16位元來儲存結果，才是真正實用的選擇。
問題是：該放棄那些位元？而該留下那些位元呢？這就是乘法運算中要面臨的慎重選擇了。

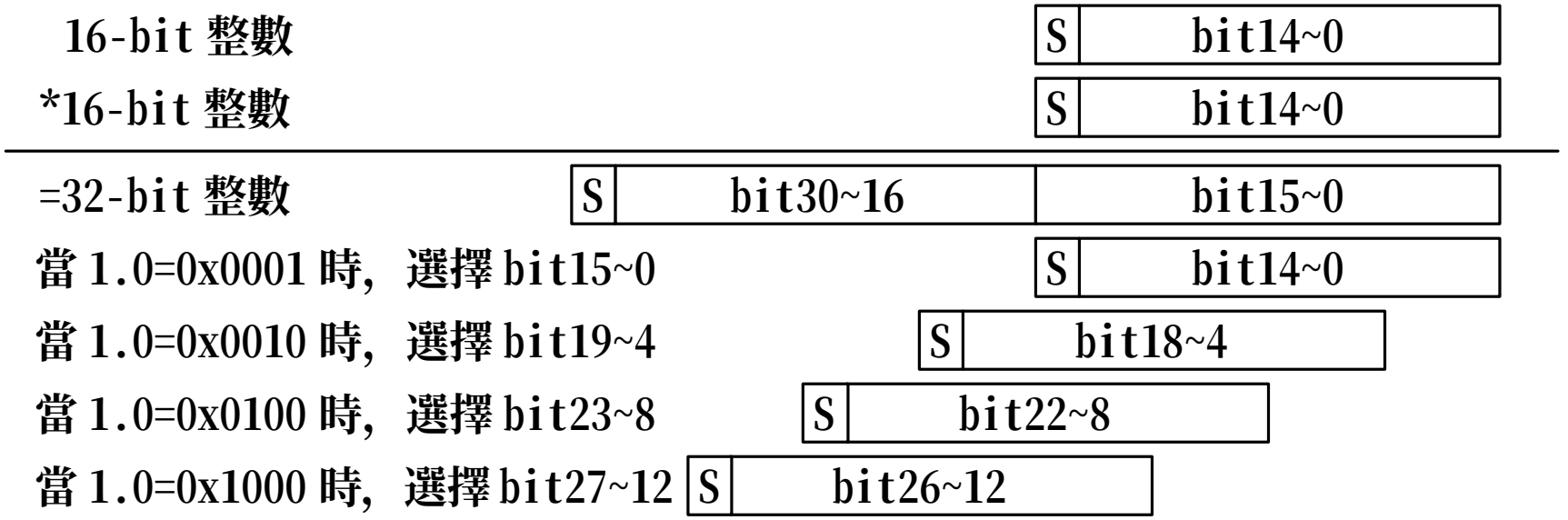
小數計算

基本上，整數運算並不是浮點運算，看起來並沒有小數的問題。例如

1.若小數點在bit0之後，則0x1000.+0x2000.=0x3000.代表了4096+8192=12288。
2.若小數點在bit4之後，則0x100.0+0x200.0=0x300.0代表了 256+ 512= 768。
3.若小數點在bit8之後，則0x10.00+0x20.00=0x30.00代表了 16+ 32= 48。

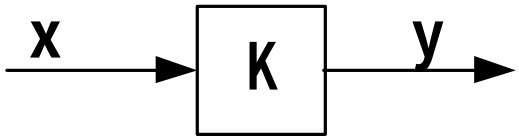
不論認定小數點放在哪裡，加減法的整數計算結果都正確無誤。
但是乘法運算就不同了：
1.首先，乘法運算的乘數和被乘數的小數點不需一致，不像加減法必須將小數點對齊。
2.就乘法計算而言，唯一要遵守的就是(X*1.0=X)的單位原則。
換句話說，乘法運算中的1.0有絕對的意義，不能任意變更。
3.若小數點在bit0 之後，則X*0x0001.=X，代表了1.0=0x0001。
4.若小數點在bit4 之後，則X*0x001.0=X，代表了1.0=0x0010。
5.若小數點在bit8 之後，則X*0x01.00=X，代表了1.0=0x0100。
6.若小數點在bit12之後，則X*0x1.000=X，代表了1.0=0x1000。

乘法運算



1.首先執行標準的16-bit乘法，其結果為32-bit，分別放置在HX和AX暫存器中。
2.接著對HX和AX暫存器執行右移的運算，
若右移 0-bit，即是選擇乘法結果的bit15~0， 相當於1.0=0x0001的狀況。
若右移 4-bit，即是選擇乘法結果的bit19~4， 相當於1.0=0x0010的狀況。
若右移 8-bit，即是選擇乘法結果的bit23~8， 相當於1.0=0x0100的狀況。
若右移12-bit，即是選擇乘法結果的bit27~12，相當於1.0=0x1000的狀況。
3.換句話說，由右移的位元數即可決定小數點的位置。

乘法運算



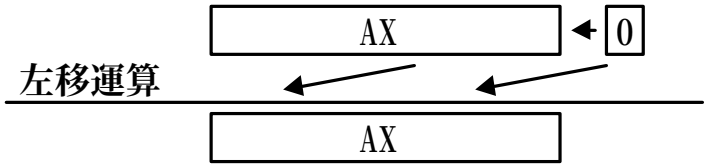
小數點在乘法計算上非常重要。上圖是一個簡單的增益，其中x訊號放大k倍後放到y中。
根據小數點的不同，K值的範圍和精度都會不同。例如：

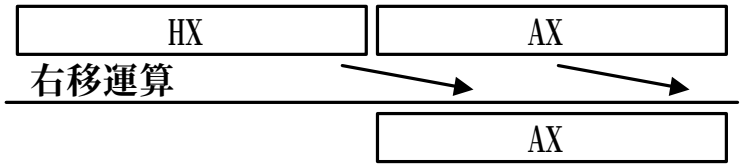
1.當1.0=0x0001時，K值範圍+/-0x7FFF.，相當於+/-32767，而精度單位為1。
2.當1.0=0x0010時，K值範圍+/-0x7FF.F，相當於+/-2048，而精度單位為0.06。
3.當1.0=0x0100時，K值範圍+/-0x7F.FF，相當於+/-258，而精度單位為0.004。
4.當1.0=0x1000時，K值範圍+/-0x7.FFF，相當於+/-8，而精度單位為0.0002。

小數點越高位時的K值範圍越小，但精度越高。我們可以依據系統需求決定增益的範圍和精度。
乘法和右移運算的定義如下：

MUL r ：HXAX=AX*R， 即16-bit乘法結果為32-bit，而置於HX和AX暫存器中。
SFR bit：AX=HXAX>>bit，即32-bit做右移，而將16-bit結果置於AX暫存器中。

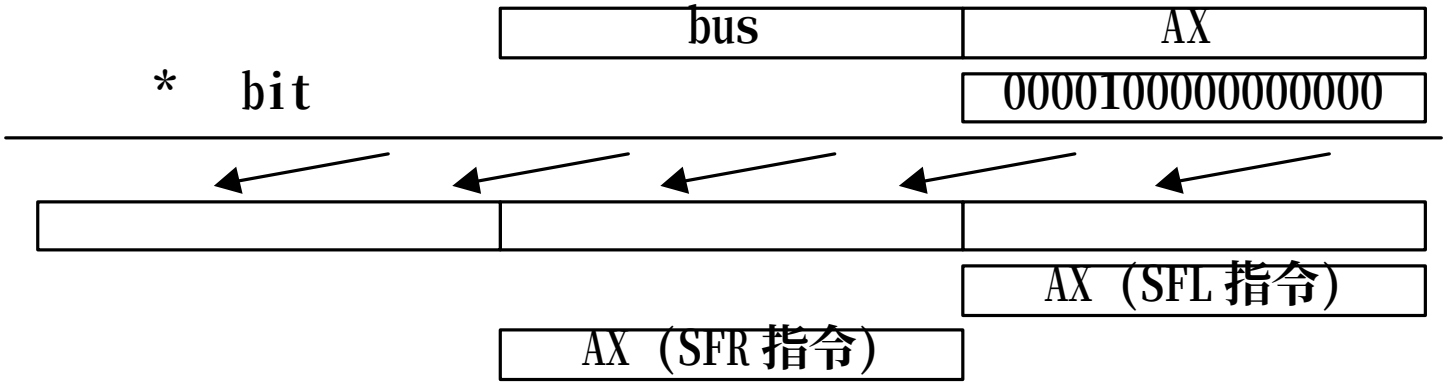
移位運算指令



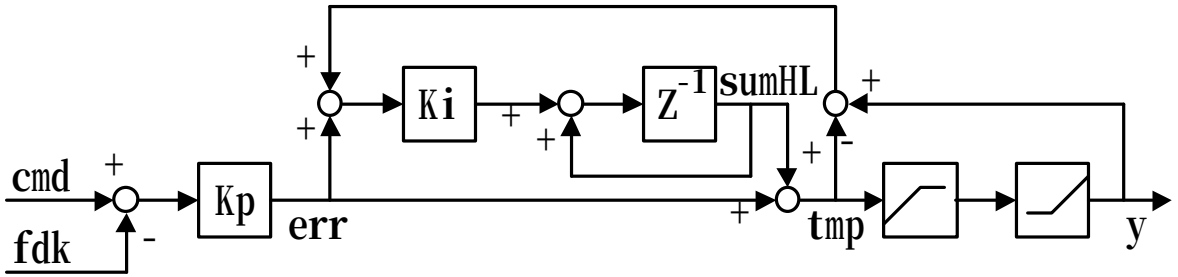


移位動作指令：

```
SFL   bit;     ;AX=   AX<<bit的左移指令，其中bit的範圍為1~15
SFR   bit;     ;AX=HXAX>>bit的右移指令，其輸入為HX和AX的32位元組合結果。
```



PI(比例積分)控制器



模組程式範例：PI_CONTROL.asm

```
....                                                        //宣告公用變數和模組變數
@initial:  .... RET;                                        //啟動程序，以RET指令結束
@timer:                                                     //即時控制程序
     RD (cmd); SUB (fdk); MUL kp; SFR 12; WR err;           //err=(cmd-fdk)*kp/4096
     ADD sumH; WR  tmp;                                     //tmp=err+sumH
     MIN high; MAX low;   WR out;                           //out=limit(tmp,high,low)
     SUB tmp;  ADD err;   MUL ki; SFR 12;                   //ACC=(out-tmp+err)*ki/4096
     SGN;      ADD sumL;  WR sumL;                          //sumHL+=ACC
     RD.H;     ADC sumH;  WR sumH;
b9:  RET;
```

DSP核心設計

記憶體的規劃

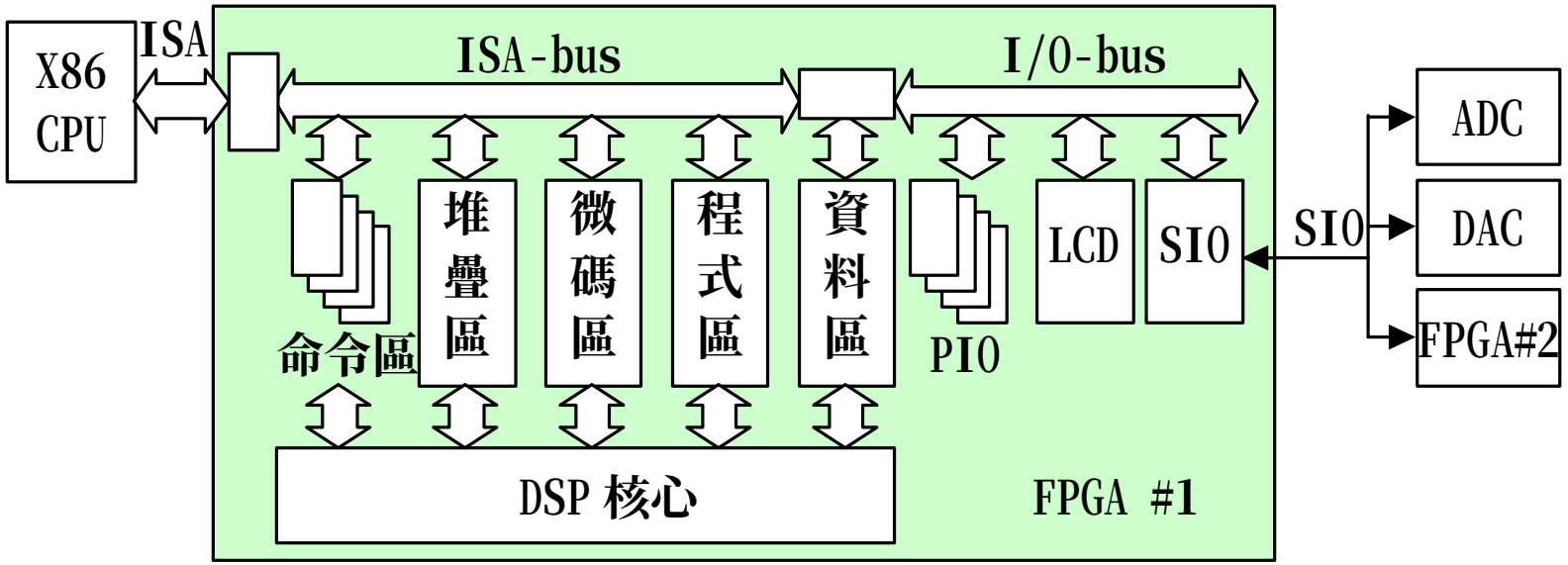
程式的執行

資料的處理

運算單元設計

硬體配置

DSP核心只是FPGA#1晶片中的一部份電路，藉著雙阜記憶體和其他的周邊溝通，包括ISA界面和I/O界面。

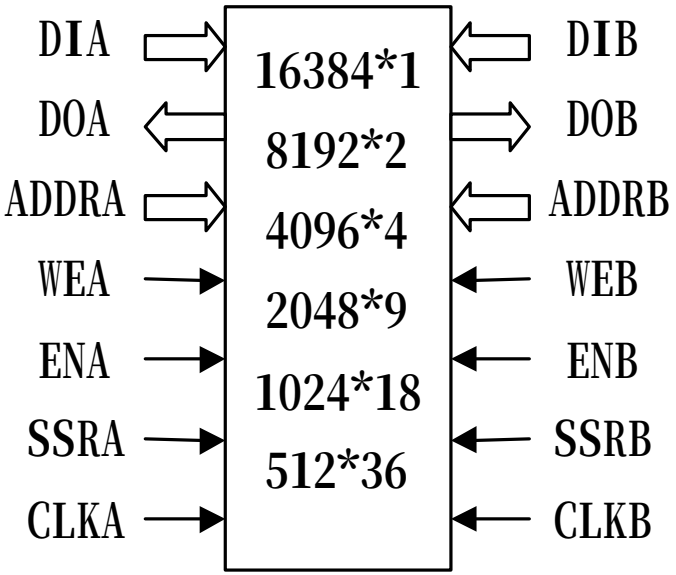


記憶體結構

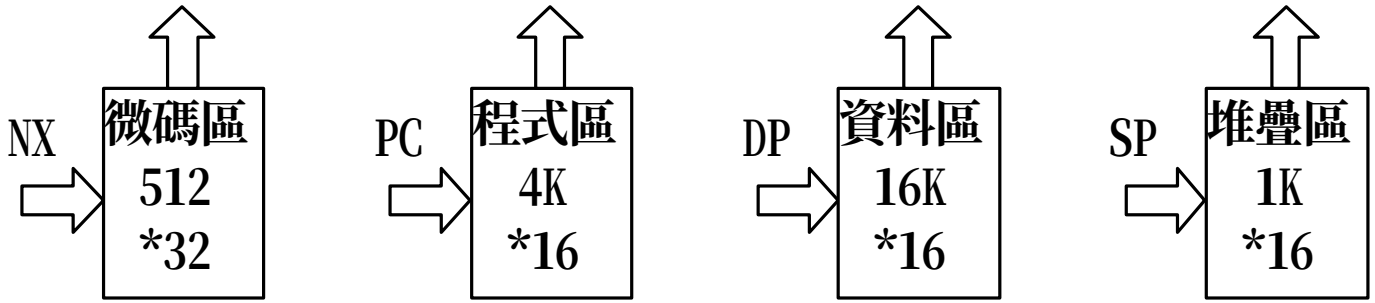
雙阜記憶體是Virtex-II系列中的標準元件。規格包括：

1.雙阜記憶體以18K-bit(2K-byte)為單位。
2.XC2V250有24個雙阜記憶體，共計48K-byte。
3.雙阜記憶體分成A和B兩個部分，兩部分控制訊號完全獨立。
4.字寬有1/2/4/9/18/36-bit等不同選擇，A/B兩部分完全獨立。
5.記憶體的讀寫為同步方式，由CLK控制，在升緣或降緣觸發。
6.WR動作由WEx=1且ENx=1控制，Data由DIx送入記憶體。
7.RD動作由ENx=1控制，Data由DOx送出記憶體。
8.DOx輸出並不是三態界面，所以不能直接連到匯流排，
而必須再串接三態緩衝器才行。
9.無論是ENx、WEx和SSRx，都可以自由選擇正向或反向觸發。
10.同步記憶體在寫入和讀出時都會有延遲現象，設計時必須特別小心。

雙阜記憶體



記憶體配置



就DSP核心而言，所謂程式控制只是RAM資料的管理而已，非常單純。RAM分成四個獨立部分，包括：

1.微碼區的容量為512*32，由NX(Next)控制。負責執行機械碼中的一個細節步驟。
2.程式區容量為4K*16，由PC(Program Counter)控制。負責執行DSP程式中的一個機械碼指令。
3.資料區容量為16K*16，由DP(Data Pointer)控制，DP值由機械碼的程式控制。
4.堆疊區容量為1K*16，由SP(Stack Pointer)控制，SP值由機械碼的程式控制。

程式的執行

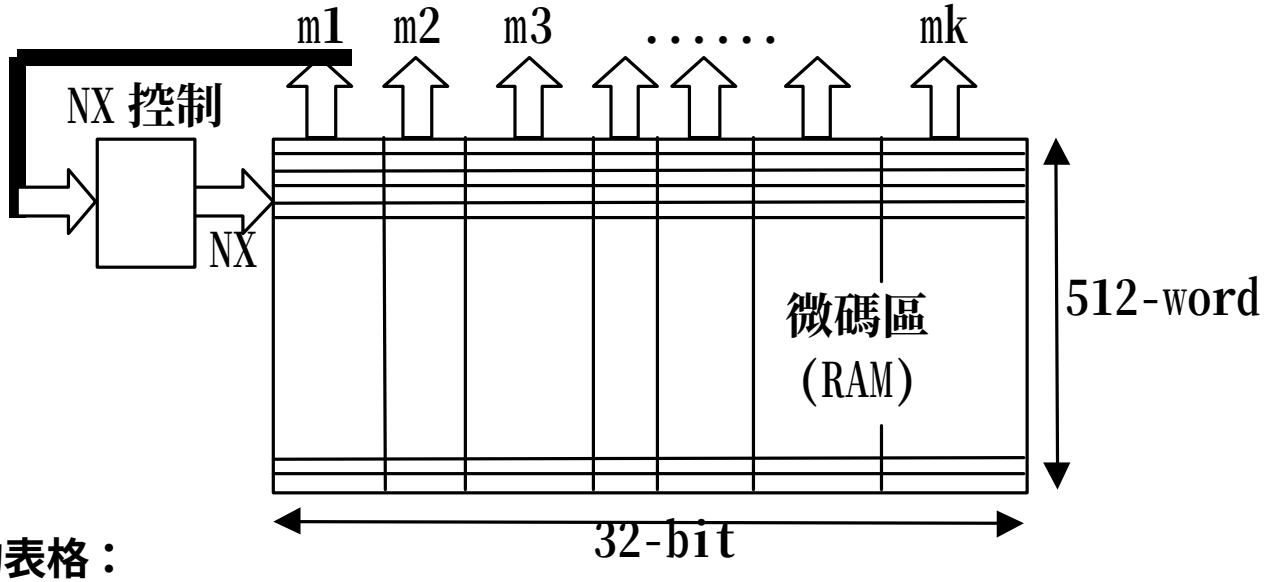
DSP程式的執行包括了三個部分：

1.微碼區的控制：
將一個機械碼再細分成多個微控碼步驟，每個步驟都佔用一個CLK週期。不同的機械碼所對應的微碼動作不同，也就造成不同的執行目標。

2.程式區的控制：
程式區儲存DSP的機械碼程式，依序的一步步執行，就能完成DSP程式所指定的動作。

3.X86命令控制：
不論是程式的啟動或停止，都需要接受X86指令的控制。因此必須建立適當的連線控制關係。

微控碼的管理



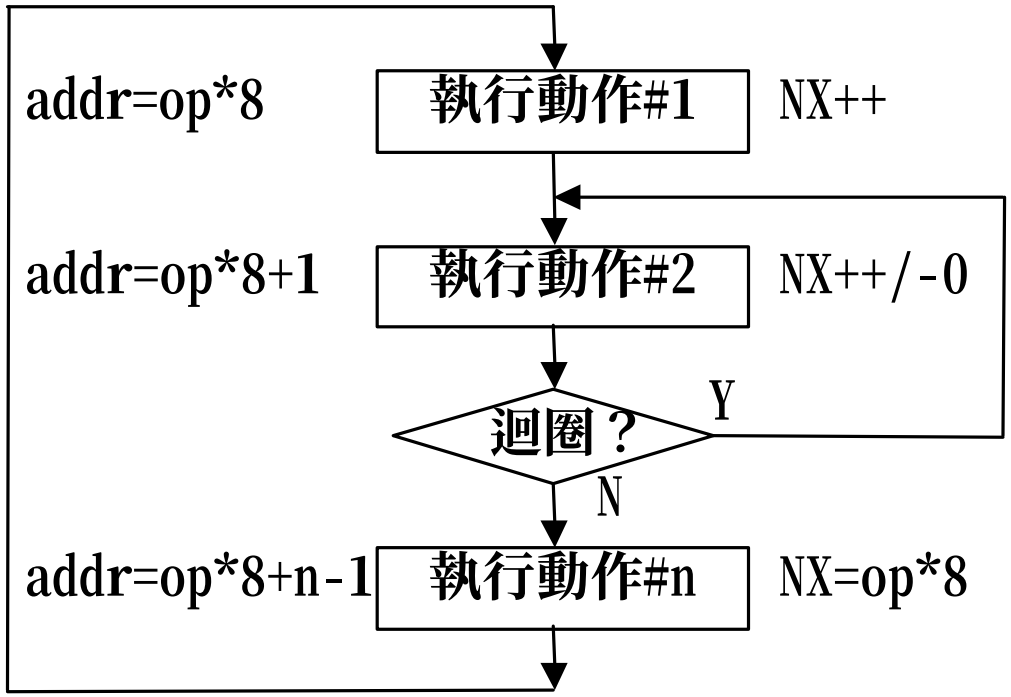
微控碼是儲存在RAM(512*32)中的表格：

1.表格的寬度為32位元，其中分成多個欄位。每個欄位代表了DSP中某個硬體部分的控制動作。
2.表格的長度為512-word，由NX做位址選擇，指向表格中正在執行的步驟。
3.每個CLK中NX都更新一次，而NX的變化就形成執行時的每個步驟。
4.每個機械碼都包括了一個或多個步驟。而每個步驟都佔用1-word的微控碼。

微控碼的執行

微控碼的執行流程：

1. 每個機械碼佔用微碼區的8-word，每個機械碼最多只有8個順序動作。
2. 512-word微碼區可放置64種機械碼
3. 根據機械碼的分類(op值，0~63)，從不同的位址(NX=op*8)開始執行。
4. 每個動作執行完畢後就將位址遞增(NX++)，自動執行下一步驟。
5. 最後步驟執行完畢時，自動將位址指向下一筆機械碼的表格(NX=op*8)
6. 微控碼也有迴圈的動作。當迴圈條件成立時，執行迴圈動作。當迴圈條件不成立時，執行下一個步驟。
7. 每筆執行動作都需要一個CLK，所需CLK的總數就是該機械碼的執行時間。



mNX欄位的控制

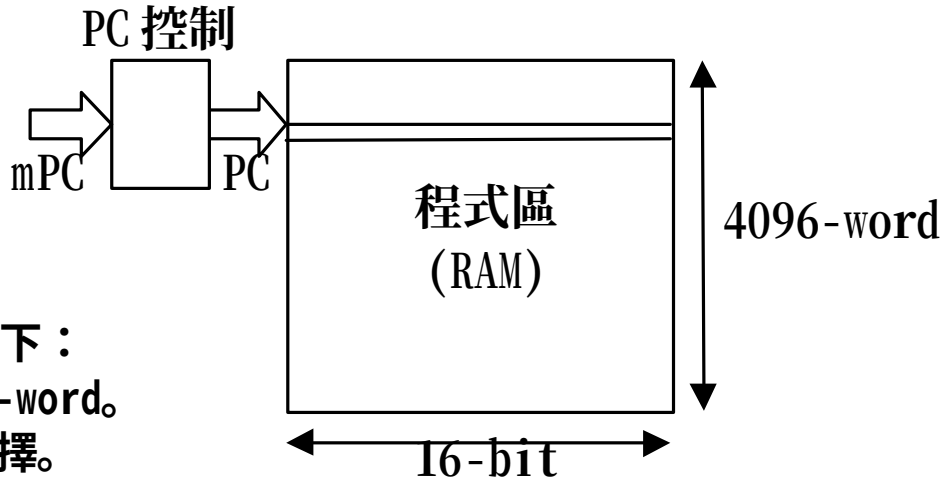
NX的控制是機械碼中控制步驟的一部份，NX控制也屬於微碼區中的一個欄位，稱之為mNX[4]。
其中2-bit定義迴圈結束時的執行狀況，而另外2-bit定義迴圈執行時的狀況。定義如下：

mNX	NX動作		定 義 說 明
	迴圈結束時	迴圈執行時	
00XX	NX=0	--	程式結束，進入HALT狀況，NX=0
01XX	NX++	--	繼續下一個步驟，NX=NX+1
10XX	NX=op*8	--	開始下一筆機械碼，NX=op*8
XX00	--	NX-=0	當迴圈條件成立時，NX=NX-0
XX01	--	NX-=1	當迴圈條件成立時，NX=NX-1
XX10	--	NX-=2	當迴圈條件成立時，NX=NX-2
XX11	--	--	沒有迴圈，固定執行迴圈結束指令

微控碼的分類

op值	控制指令	op值	運算指令	op值	RD指令群	op值	WR指令群
00-0000	HALT	01-0000	ADD.0 r	10-0000	RD.0 r	11-0000	WR.0 r
00-0001	--	01-0001	ADD.0 (r)	10-0001	RD.0 (r)	11-0001	WR.0 (r)
00-0010	--	01-0010	ADD r	10-0010	RD r	11-0010	WR r
00-0011	NOP	01-0011	ADD (r)	10-0011	RD (r)	11-0011	WR (r)
00-0100	RET	01-0100	ADD.H	10-0100	RD.H	11-0100	WR.H
00-0101	JR.H #k	01-0101	ADD.M	10-0101	RD.M	11-0101	WR.M
00-0110	PRT.0	01-0110	ADD.S r	10-0110	RD.S r	11-0110	PUSH.H
00-0111	PRT.F	01-0111	ADD.S (r)	10-0111	RD.S (r)	11-0111	PUSH.F
00-1000	JR #k	01-1000	ADD.X r	10-1000	RD.X r	11-1000	PUSH.A
00-1001	JMP #k	01-1001	ADD.X (r)	10-1001	RD.X (r)	11-1001	PUSH r
00-1010	CALL #k	01-1010	ADD.S	10-1010	RD.X	11-1010	PUSH (r)
00-1011	WR.MP #k	01-1011	--	10-1011	--	11-1011	POP.H
00-1100	RPT #k	01-1100	ADD #k	10-1100	RD #k	11-1100	POP.F
00-1101	WR.LP #k	01-1101	ADD #k,bit	10-1101	RD #k,bit	11-1101	POP.A
00-1110	LOOP #k	01-1110	ADD r,bit	10-1110	RD r,bit	11-1110	POP r
00-1111	--	01-1111	ADD r,#k	10-1111	WR r,#k	11-1111	POP (r)

程式區的管理



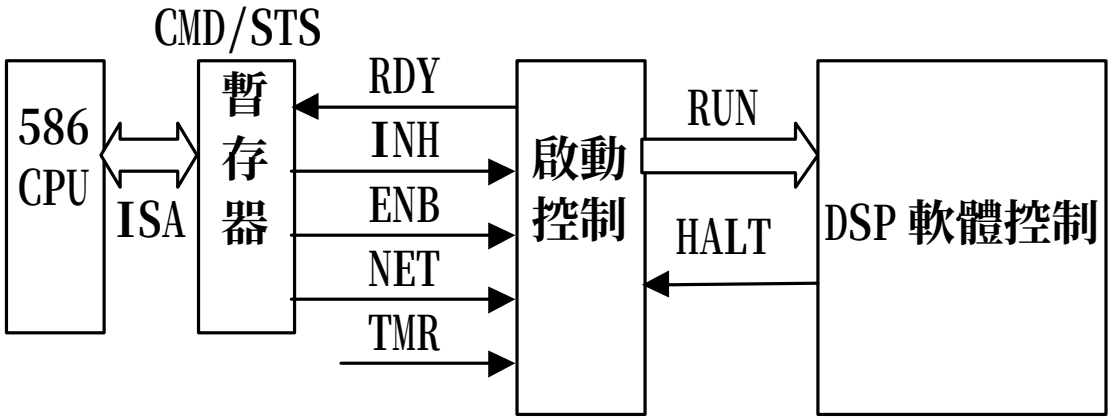
程式區是儲存在RAM(4096*16)中的表格，說明如下：

1. 程式區的寬度為16位元，每筆機械碼佔有1-word。
2. 程式區的長度為4096-word，由PC做位址選擇。
   PC就是下一筆程式的位址。
3. 對於PC的更新動作是由微控碼來控制，其欄位為mPC[3]。

mPC欄位的控制

mPC	PC	定 義 說 明
000	nop	不處理
001	PC++	將PC指向下一筆程式
010	PC+=bus	將PC移動相對值，如JR、CALL指令
011	PC=bus	將PC設為絕對值，如JMP、RET指令
100	PC=run	程式啟動時由硬體設定PC值， 此時有PC=0/1/2/3的四種選擇
101	PC++/	有條件的執行PC++動作，通常配合NX迴圈控制。當迴圈結束時執行PC++，否則不處理
110	PC+=bus/	有條件的將PC移動相對值，如JGE、JC、LOOP指令 當條件成立時執行PC+=bus的動作，否則不處理
111	bus=PC	讀出目前的PC值，如CALL指令

X86控制界面



對於DSP的控制，
不論是啟動或停止，
都是由X86CPU透過ISA-bus來控制。
其中的暫存器定義如下：

位址	WR	RD	定 義 說 明
0x1000	PAGE[4]	PAGE[4]	選擇2KW的記憶體，包括資料區、程式區、微碼區、堆疊區
0x1002	CMD[4]	STS[6]	TMR/ENB/INH/RDY
0x1004	PRD[3]	--	prd=0：由X86的TMR命令來控制週期 prd=1~6：控制週期固定為1KHz~32KHz

1.以INH命令強制DSP停止，依序下載微碼區、程式區和資料區。
2.設定PRD以選擇即時控制的週期。
3.以ENB命令管理DSP的即時控制，啟動@initial和@stop程序。

機械碼的分類

DSP核心中的機械碼固定以16位元表示，格式如下：

	15	14	13	9	8	4	0	
OP #k	0	0	0	func	k=+/-255			K0 類群
OP #k,bit	0	0	1	func	bit	k=0~31		K1 類群
OP r,bit	0	1	0	func	bit	reg		K2 類群
OP r,#k	0	1	1	func	k=0~15	reg		K3 類群
OP (r)	1	0	0	func	mode	reg		K4 類群
OP (r)	1	0	1	func	mode	reg		N5 類群
JR #k	1	1	00	cond	k=+/-127			J6 類群
JMP #k	1	1	XX	k=0~4095				J7 類群

func	K0類群	K1類群	K2類群	K3類群	K4類群	N5類群	J6類群	J7類群
0000	AND	AND	AND	AND	AND	HALT	JR.C	--
0001	OR	OR	OR	OR	OR	--	JR.NC	--
0010	XOR	XOR	XOR	XOR	XOR	--	JR.Z	--
0011	MUL	MUL	--	MUL	MUL	NOP	JR.NZ	--
0100	MAX	MAX	MAX	MAX	MAX	RET	JR.N	--
0101	MIN	MIN	MIN	MIN	MIN	--	JR.P	--
0110	ADD	ADD	ADD	ADD	ADD	PRT.0	JR.V	--
0111	ADC	ADC	ADC	ADC	ADC	PRT.F	JR.NV	--
1000	SUB	SUB	SUB	SUB	SUB	--	JR.LE	--
1001	SBC	SBC	SBC	SBC	SBC	--	JR.GT	JMP
1010	--	--	--	ABS	ABS	--	JR	CALL
1011	--	--	--	NEG	NEG	--	JR.H	WR.MP
1100	RPT	--	SGN	COM	COM	--	--	--
1101	WR.LP	--	SFR	--	--	--	--	--
1110	LOOP	--	SFL	--	WR	--	--	--
1111	RD	RD	RD	WR	RD	--	--	--

機械碼的解碼

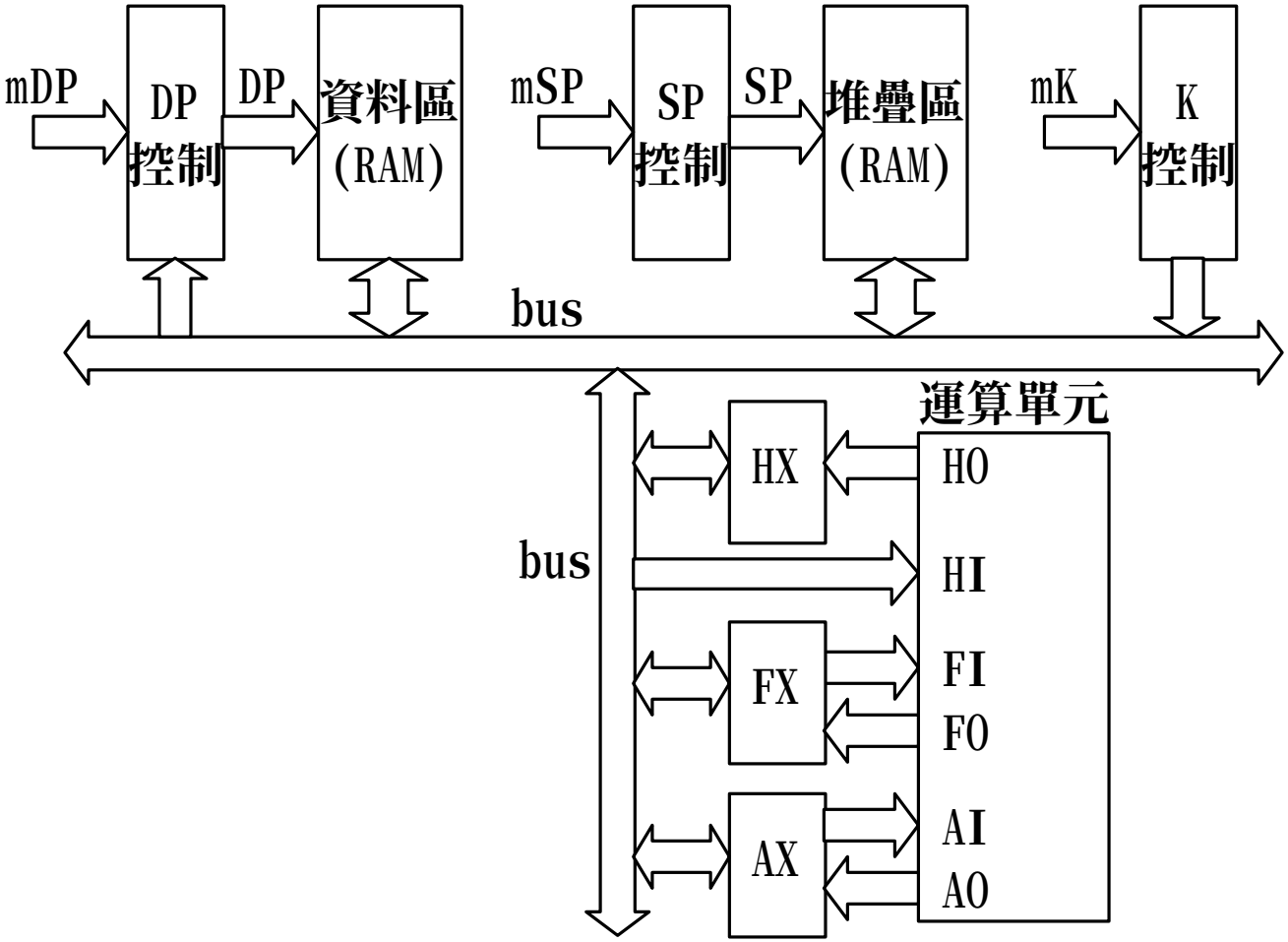
在機械碼的解碼過程中，實際上包含了兩組解碼動作：

1.首先是PC值更新，接著立刻讀出機械碼並解碼。
2.在同一個PC週期中，有些解碼結果會被立刻使用，像是op值和reg值。
3.其他解碼後的參數就必須再延遲一個PC週期，到下一次PC值改變時才會運用。

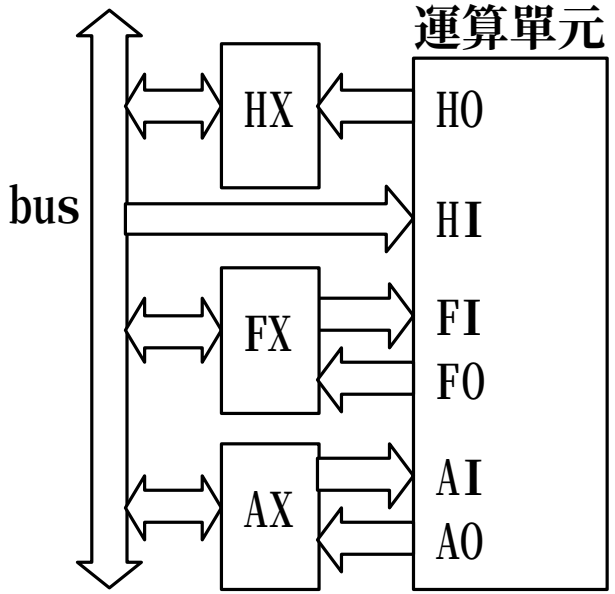
從機械碼中可以解出的參數包括：

op值:　做為微控碼的表格控制，　為6位元，不需延遲。
reg值: 解出機械碼中的變數位址，為14位元，不需延遲。
k值:　解出機械碼中的常數，　為16位元，需要延遲一個PC週期。
alu值: 解出機械碼中的運算指令，為4位元，需要延遲一個PC週期。
bit值: 解出機械碼中的移位數，　為4位元，需要延遲一個PC週期。
cnd值: 解出JR指令中的條件選擇，為4位元，需要延遲一個PC週期。

資料的處理


mDP
DP
控制
DP
資料區
(RAM)
mSP
SP
控制
SP
堆疊區
(RAM)
mK
K
控制
bus
運算單元
HX
HO
bus
HI
FX
FI
FO
AX
AI
AO

運算單元設計



運算單元在整個DSP中負責主要的運算處理工作：

1. 運算單元負責處理(HO,AO,FO)=alu(HI,AI,FI)的動作，和其他模組無關。
2. 運算所需的暫存器有三組：其中
   - AX :為16-bit暫存器，為主要運算儲存位置。
   - HX :為16-bit暫存器，做為輔助儲存和32-bit運算。
   - FX :為4-bit的旗標暫存器，包括了CF、ZF、NF和VF。
3. 運算單元的輸入包括AI、FI和HI，其中AI和FI直接是AX和FX暫存器的輸出，而HI並非HX的輸出，而是直接接到匯流排上。
4. 運算單元的輸出包括AO、FO和HO，三者都直接接到AX、FX和HX的暫存器。

pALU	運算種類	輸出	旗標更新	指令說明
0000	AO=AI.and.HI	AO	NF,ZF	AND指令，邏輯運算
0001	AO=AI.or. HI	AO	NF,ZF	OR 指令，邏輯運算
0010	AO=AI.xor.HI	AO	NF,ZF	XOR指令，邏輯運算
0011	AO=~AI	AO	NF,ZF	COM指令，邏輯運算
0100	AO=AI-HI	AO	VF,NF,ZF,CF	SUB指令，加減運算
0101	AO=AI-HI-CF	AO	VF,NF,ZF,CF	SBC指令，加減運算
0110	AO=AI+HI	AO	VF,NF,ZF,CF	ADD指令，加減運算
0111	AO=AI+HI+CF	AO	VF,NF,ZF,CF	ADC指令，加減運算
1000	AO=MAX(AI,HI)	AO	VF,NF,ZF	MAX指令，比較運算
1001	AO=MIN(AI,HI)	AO	VF,NF,ZF	MIN指令，比較運算
1010	AO=ABS(AI)	AO	0,NF,ZF	ABS指令，比較運算
1011	AO=0-AI	AO	0,NF,ZF	NEG指令，比較運算
1100	AO= (AI)<<bit	AO	NF,ZF	SFL指令，移位運算
1101	AO=(HI,AI)>>bit	AO	NF,ZF	SFR指令，移位運算
1110	(HO,AO)=sign(AI,bit)	HO,AO	NF,ZF	SGN指令，補位運算
1111	(HO,AO)=AI*HI	HO,AO	NF,ZF	MUL指令，乘法運算

運算控制

mREG	整體動作	AX	HX	FX	bus	ALU	說　　明
0000	nop	--	--	--	--	--	不處理
0001	HX=AX	--	HX=AI	--	--	--	WR.H　動作
0010	AX=HX	AX=HI	--		bus=HX	--	RD.H　動作
0011	AX<=>HX	AX=HI	HX=AI		bus=HX	--	RD.X　動作
0100	HX=AX=bus	AX=HI	HX=AI		--	--	RD.X r 動作
0101	AX=bus<<bit	AX=HI	--		--	--	RD r<<bit動作
0110	bus=HX	--	--	--	bus=HX	--	PUSH,H 動作
0111	HX=bus	--	HX=bus	--	--	--	POP,H　動作
1000	bus=AX	-	-	-	bus=AX	-	PUSH,A 動作
1001	AX=bus	AX=bus	--		--	--	POP,A　動作
1010	bus=FX	--	--	--	bus=FX	--	PUSH,F 動作
1011	FX=bus		--	FX=bus	--	--	POP,F　動作
1100	HA=alu=bus	AX=AO	HX=HO	FX=FO	--	pALU--	運算執行(#1)
1101	HA=alu=HA	AX=AO	HX=HO	FX=FO	bus=HX	pALU	運算執行(#2)
1110	HA=sfr	AX=AO	HX=HO	FX=FO	bus=HX	SFR	運算執行(#3)
1111	AX=bus/	AX=bus	--	--	--	--	PRT動作，但是只有當VF=1時才執行

教學實驗規劃

電機控制實驗
數位通訊實驗
數位邏輯實驗
高等邏輯實驗

電動機控制實驗

變頻器模式實驗　(開環路電壓控制)
步進馬達模式實驗(閉環路電流控制)
直流馬達模式實驗(閉環路扭力控制)
速度控制模式實驗
位置控制模式實驗
加減速控制與前置補償
電流過載與PID補償效應

數位通訊實驗

基本原理

實驗 1 ：認識類比訊號(正弦波/AM/FM/PM)
實驗 2 ：認識數位訊號(NRZ/Manchester/AMI)
實驗 3 ：數位濾波器設計(IIR)
實驗 4 ：數位濾波器設計(FIR)
實驗 5 ：相鎖迴路設計(VCO/PLL)

通訊介質

實驗 6 ：UART 界面設計(RS232/RS422)
實驗 7 ：紅外線界面設計(NEC 通訊規格)
實驗 8 ：無線電界面設計(HT12x 通訊規格)
實驗 9 ：光纖界面設計(SPDIF 通訊規格)
實驗 10：電話界面設計(DTMF 通訊規格)

調變與解調

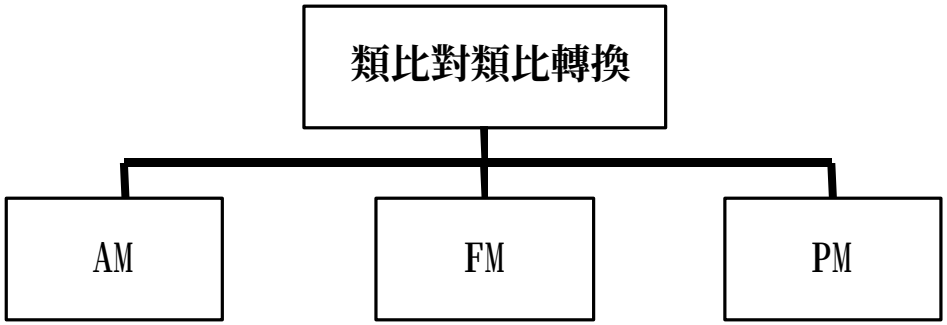
實驗 11：PCM 的調變與解調
實驗 12：ASK 的調變與解調
實驗 13：FSK 的調變與解調
實驗 14：PSK 的調變與解調
實驗 15：QAM 的調變與解調

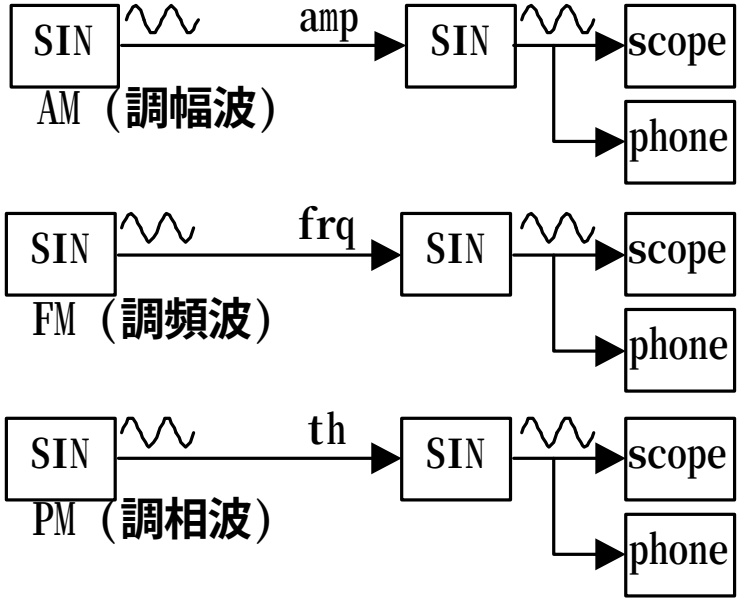
專題應用

實驗 16：載波還原設計
實驗 17：CRC 編碼與檢測
實驗 18：TDM 多工器設計
實驗 19：2400MODEM 設計
實驗 20：9600MODEM 設計

數位通訊實驗

實驗一：認識類比訊號





數位邏輯實驗

實驗一：基本架構
實驗二：訊號定義
實驗三：邏輯處理
實驗四：序列處理
實驗五：數值處理
實驗六：基本暫存器
實驗七：計數處理
實驗八：計時處理
實驗九：程序處理
實驗十：系統設計
實驗十一：電子鐘專題
實驗十二：密碼鎖專題
實驗十三：步進馬達專題
實驗十四：運動控制專題
實驗十五：溫度控制專題

高等邏輯實驗

實驗一：記憶體設計
實驗二：ISA-bus界面設計
實驗三：乘法器設計
實驗四：運算單元設計
實驗五：資料管理設計
實驗六：微控碼設計
實驗六：機械碼解碼設計
實驗七：LCD界面設計
實驗八：SIO界面設計
實驗十：系統整合