Programmable Peripheral Application Note 042

Four Axis Stepper Motor Control Using a Programmable PSD5XX MCU Peripheral from WSI, Inc.

By Nasser Pooladian, Data Card Corp.

Introduction

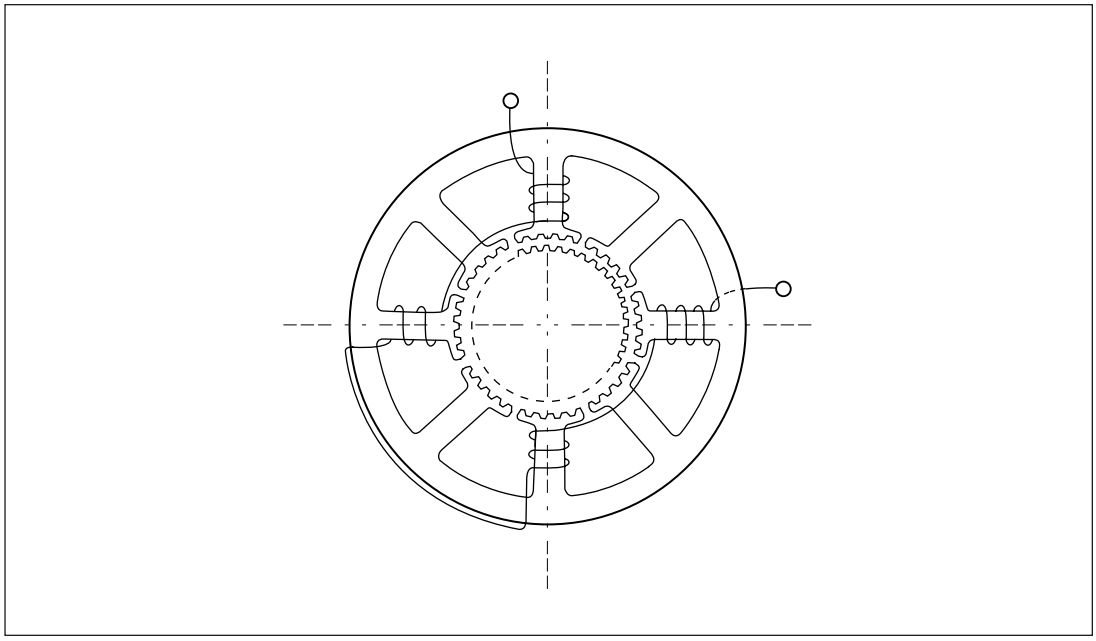
The design of a stepper motor control requires various timers and electronic controls. This application note explains the basic operation of a stepper motor. It also presents the theory, implementation and electronic control of a four axis stepper motor control using the PSD5XX family of products from the WSI Inc. The PSD5XX, as a field programmable microcontroller peripheral device, provides a high degree of integration on the embedded controller design. Configuration of the memory, ease of interface to various different microcontroller buses, interrupt handling, I/O ports, and four sixteen bit counter/timers make this device a great candidate for embedded applications.

Stepper Motor Operation

A stepper motor is basically a rotational actuator which rotates a fixed anglewhen excited. A stepper motor can be directly controlled electronically without the need for a feedback element (encoder, tachometer feedback, etc.) as required in servo applications. The simpler drive and control electronics needed by a stepper motor makes it a good candidate for a positioning actuator in many different motion control applications. Several different types of stepper motors are used in the industry.

A hybrid stepping motor is used in this application. The rotor and stator are multi-toothed in a hybrid stepping motor and the rotor is magnetized in the axis of the rotor shaft. When properly driven, a hybrid stepping motor will step 1.8 degrees in the full step mode and 0.9 degrees in the half step mode. Figure 1 shows a typical hybrid motor.

Figure 1. Hybrid Stepping Motor



Stepper Motor Operation (Cont.)

The stator windings in a Hybrid stepper motor are distributed in 90-degree quadrants around the motor case. See Figure 1 for the phase winding distribution of the hybrid motor. Different methods are used for the excitation of a stepper motor. In this application a bipolar drive circuit is used for the power stage. The motor windings are connected 90 degrees apart such that the stepper motor looks like a two-phase motor. In this case there are four motor leads to be powered from the amplifier stage. Each phase of this stepper motor is powered by an H bridge. Figure 2 shows a typical H bridge that drives a stepper motor and Figure 3 illustrates the driving waveforms.

Figure 2. Two H Bridges for Driving a Two Phase Stepper Motor.

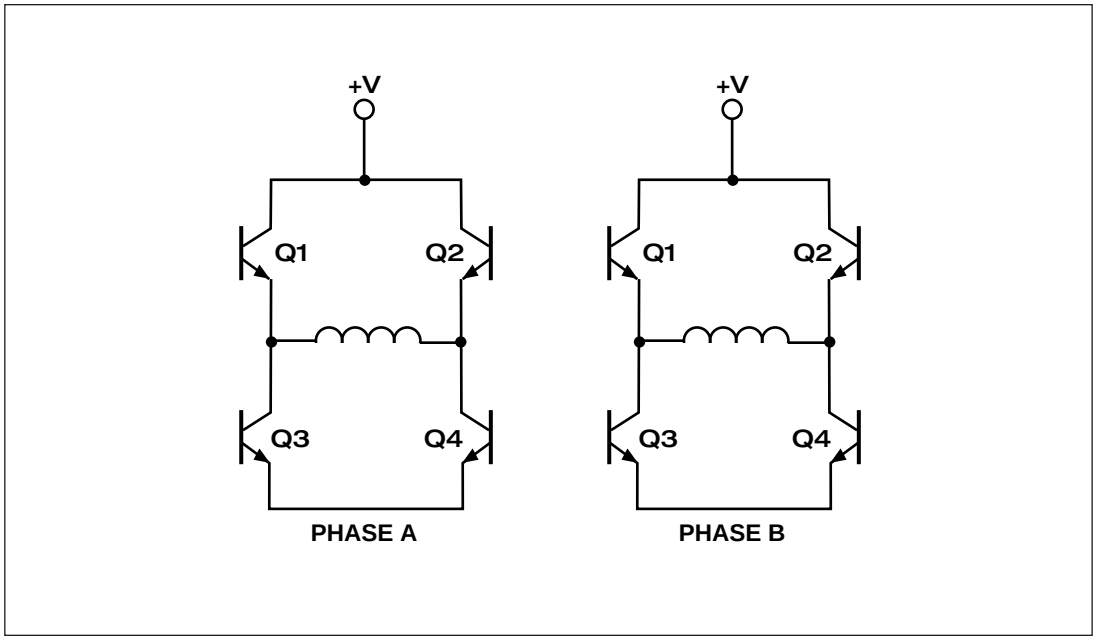
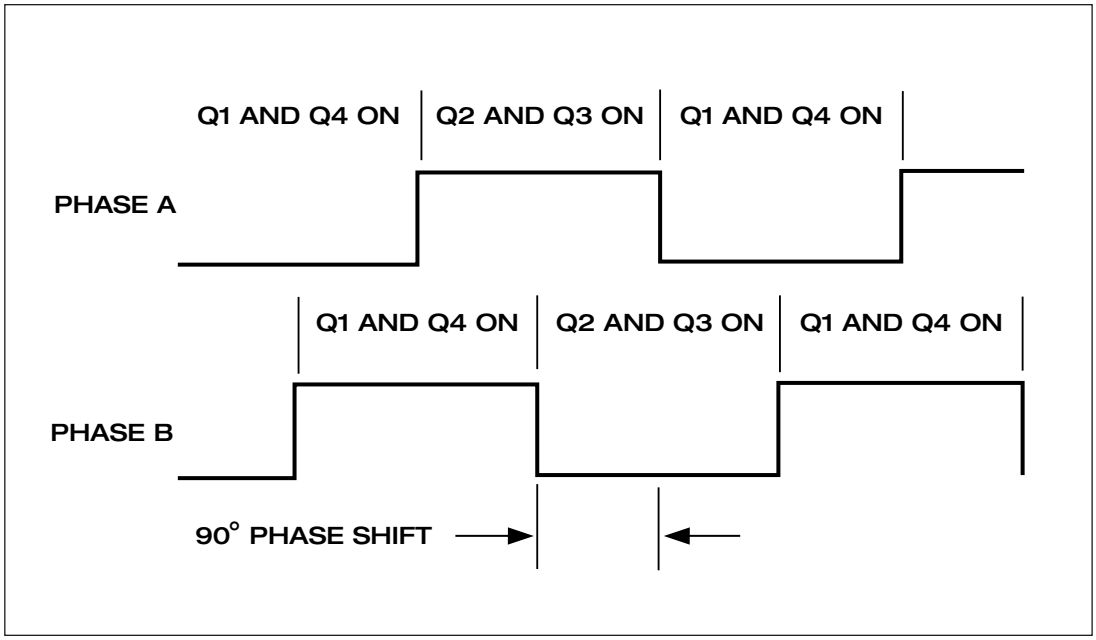


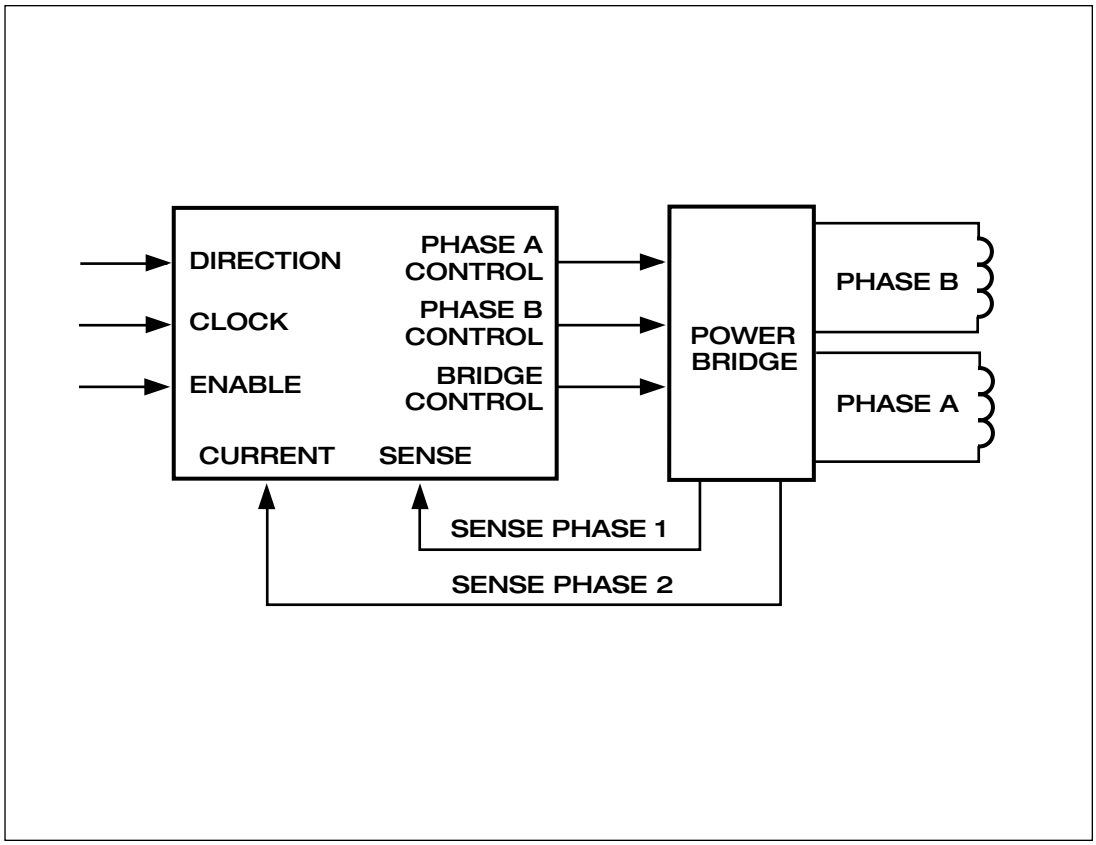
Figure 3. Phase Excitation in a Bipolar Stepping Motor.



Stepper Motor Operation (Cont.)

Phase timing for a stepper motor could be designed by either a combination of logic and linear electronics or by some stepper motor control IC's such as the L297 stepper motor controller. Figure 4 shows a block diagram of a stepper motor control and the L297 is used as the stepper motor control IC. The L297 provides control to an amplifier in the current mode. The chop frequency for the L297 is set to 20KHz. Chop frequency is used to regulate the amount of current in the motor windings. The current reference to the motor windings is set by a pair of resistors. The L297 is configured to FULL STEP mode. The ENABLE/DISABLE and axis DIRECTION control are controlled from PORT B of the PSD503B1. An electrical schematic using the L297 is given in Figure 13.

Figure 4. Simplified Block Diagram for a Stepper Motor Control

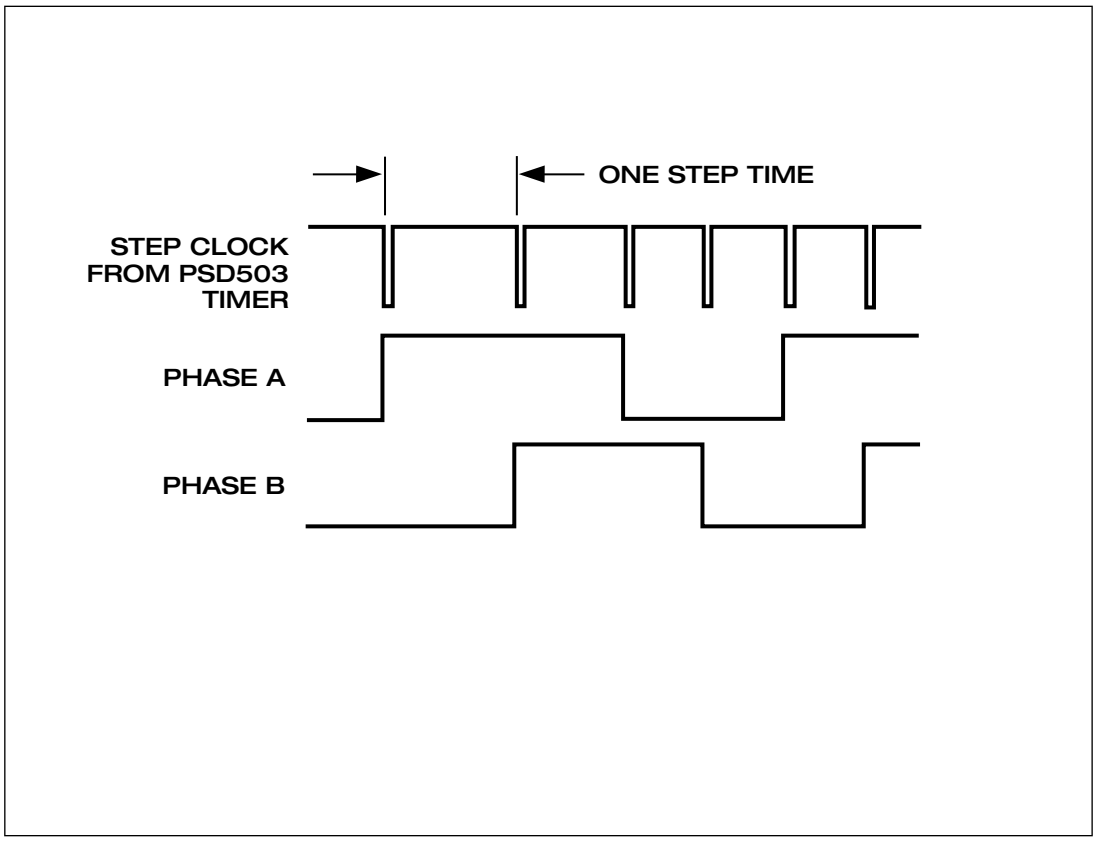


Stepper Motor Clock Generation by Using a PSD5XX

Figure 5 shows a timing diagram for the control of the phases in a stepper motor control where the steps and the step rate are controlled by clocks. The variation of the clock rate or the variation of the time between the two clock pulses determines the step rate. Change in the step rate determines the acceleration, deceleration, and the slew rate in a given motion profile.

Figure 6 shows a typical trapezoidal motion profile. In the acceleration mode the step rate starts slowly and as the motion progresses the step rate increases according to a step rate table until it reaches the slew rate. At the slew rate the step rate is fixed and the period of the step clocks is constant. At the end of the slew rate the deceleration starts. In this part of the profile the step rate decreases according to a step rate table until the last step. The repeatability and accuracy of the step clocks in a stepper motor plays a major role in the stepper motor performance.

Figure 5. Timing Diagram for a Stepper Motor Control



Stepper Motor Clock Generation by Using a PSD5XX (Cont.)

Figure 6. Typical Trapezoidal Speed Profile

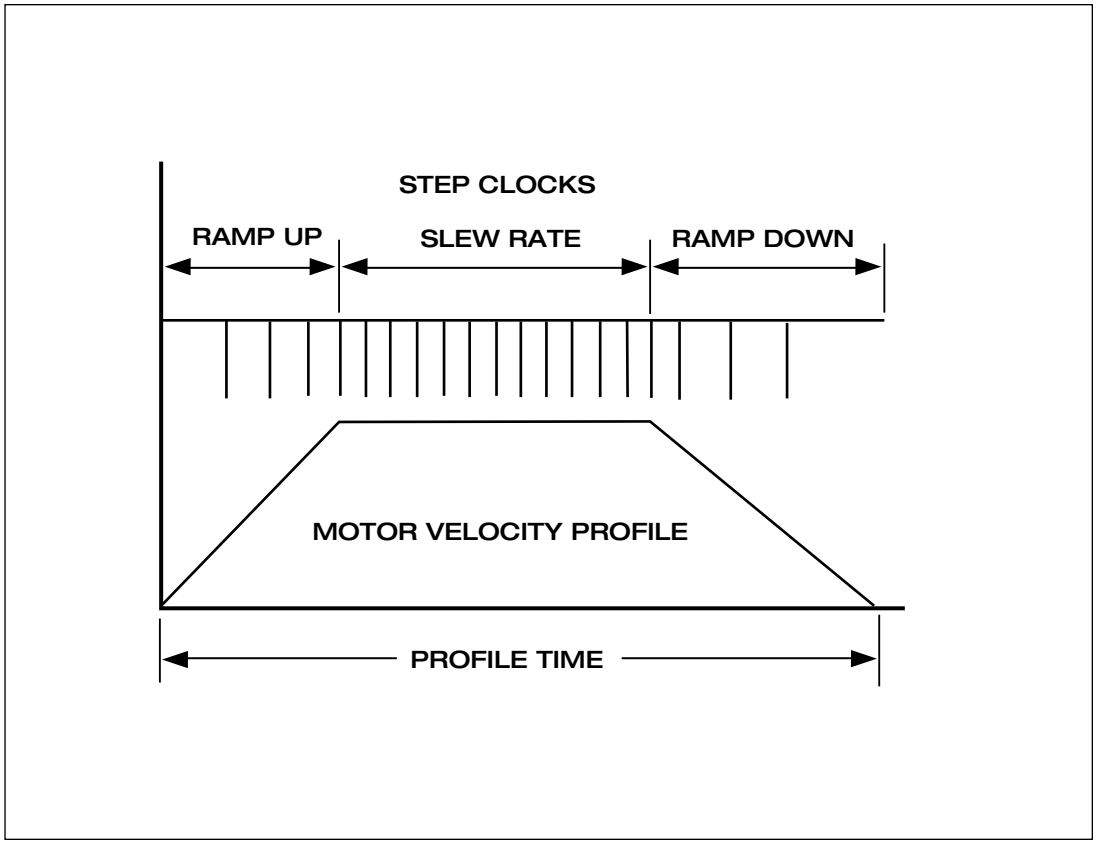
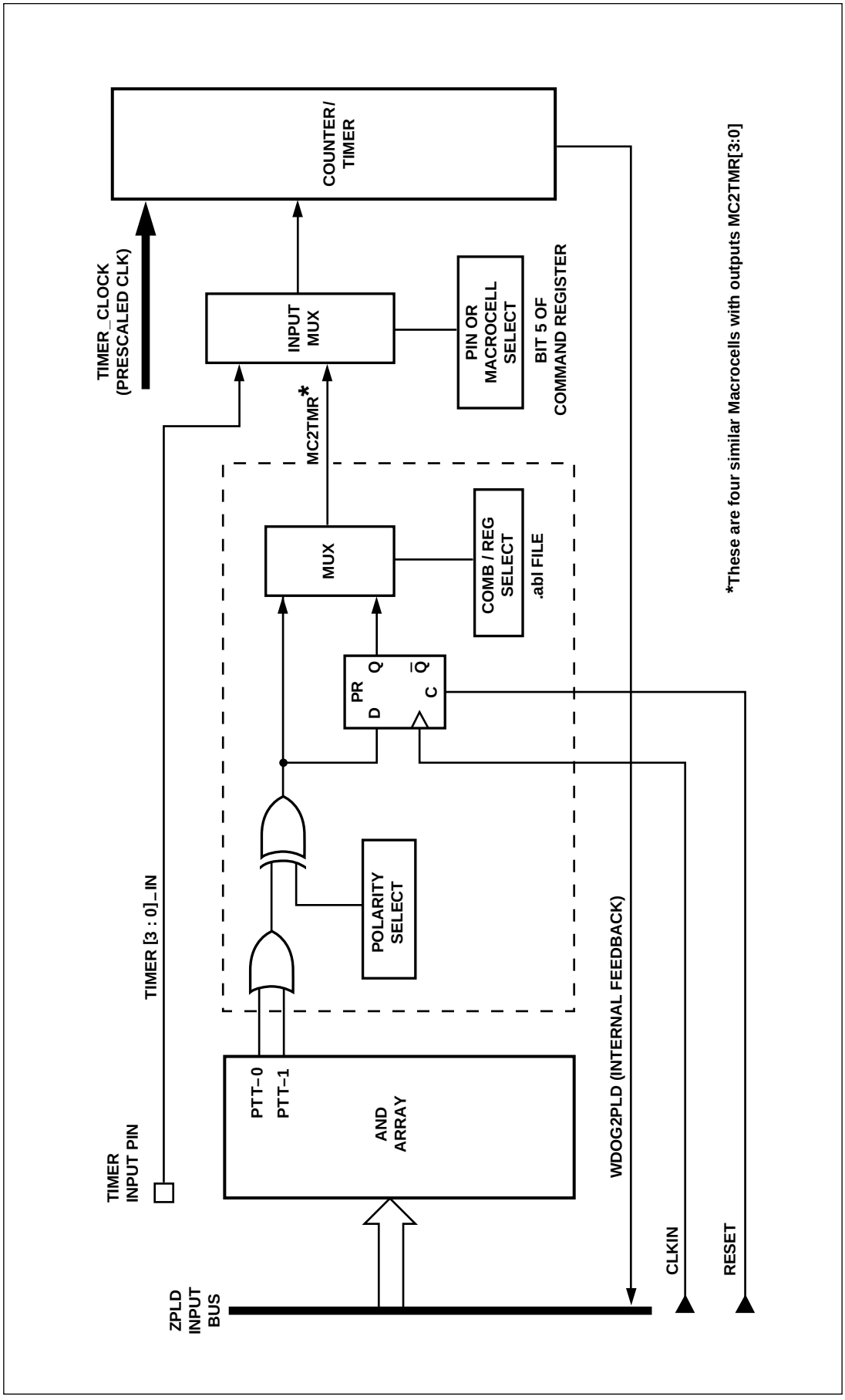


Figure 7 shows the programmable PLD (PPLD) macrocell for each counter/timer block diagram in the PSD5XX. In this design the four 16 bit timers on the PSD5XX are used to control a four axis stepper motor under microprocessor control. The four 16-bit timers in the PSD5XX are configured in the pulse mode. The Timers are loaded with a given step count for the duration of a pulse. When the pulse duration has expired, the logic on the PSD5XX is programmed such that the respective timer is preloaded with the count from the Image Registers. By preloading the timer, the step pulse duration will be exact with respect to the applied clock frequency. The timer clocks are configured to run at 1-MHz. In this case the preloading time on this system is based on a “one step ahead” stepper motor control. On the ramp up and ramp down mode each step clock will be preloaded in the image register because of the step rate changes. When the time for each step has expired the respective timer automatically preloads the image register in the count register and continues the new count. In this design the terminal count outputs (TC0 – TC3) of the timers are routed to the four inputs (INT0– INT3) of the interrupt controller on the PSD5XX device. The timer outputs are inverted and connected to the timer macrocell outputs MC2TMRx (x = 0 – 3 for three timers) in the PPLD logic. Figure 8 shows a simplified block diagram for the four axis stepper motor control.

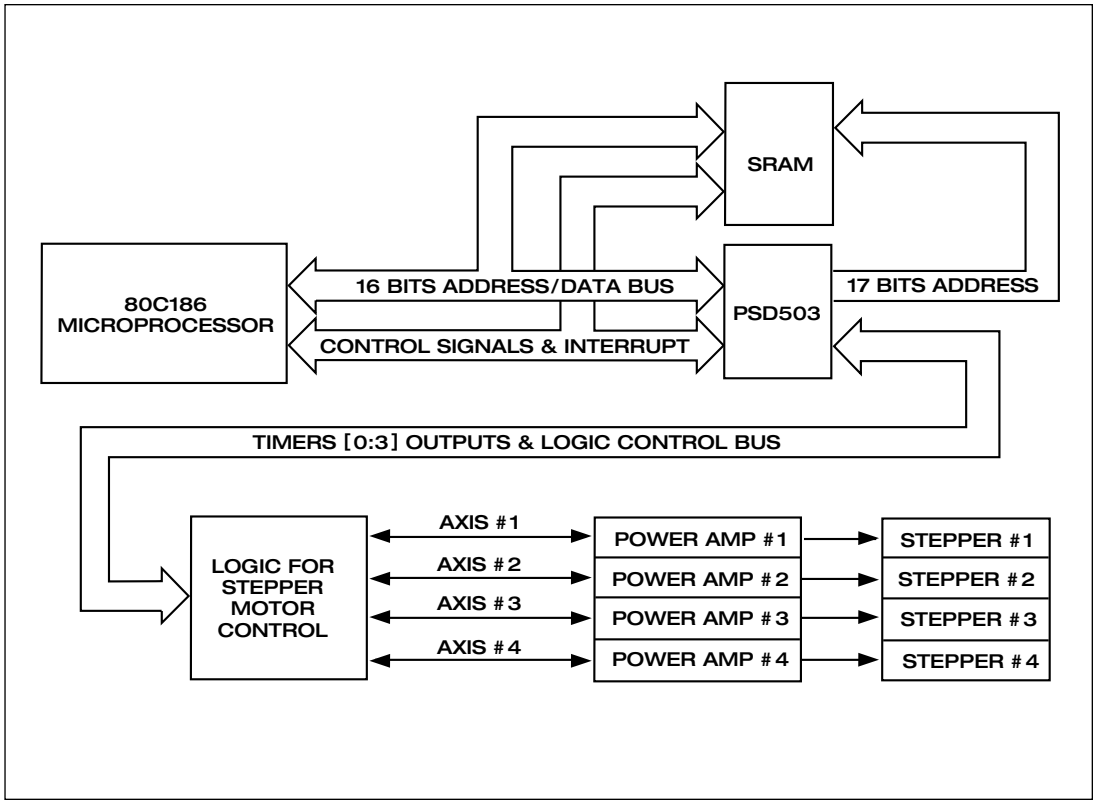
Stepper Motor Clock Generation by Using a PSD5XX (Cont.)

Figure 7. PPLD Macrocell for Each Counter/Timer



Stepper Motor Clock Generation by Using a PSD5XX (Cont.)

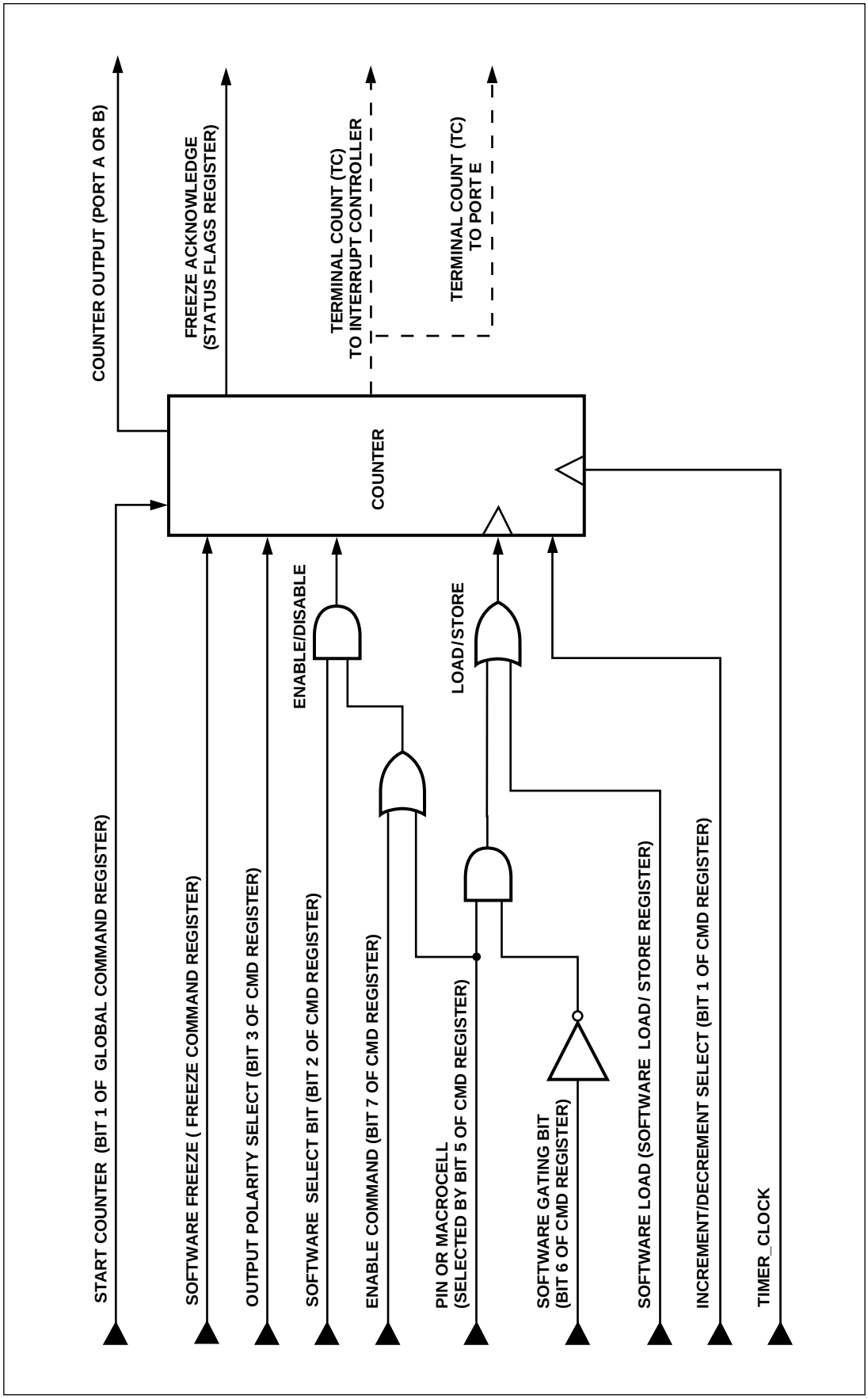
Figure 8. Simplified Block Diagram for the System



The output of the PSD5XX interrupt controller is connected to one ot the interrupt inputs on the 80C186 microprocessor. The PSD5XX interrupt controller interrupts the microprocessor in response to the timer underflow. In response to this interrupt, the microprocessor reads the INTERRUPT PRIORTY STATUS REGISTER and updates the respective timer image register. The output of a timer makes a high to low transition when a timer count expires.The high to low transition of the timer is inverted and is used to preload the respective timer from the last image register. In the slewing mode the IMAGE REGISTER for a timer does not need to be preloaded on each step interrupt. As the timer count expires the old count will be pre-loaded automatically. Figure 9 shows the logic configuration for a given axis and Figure 10 shows the *.abl file listing for the preloading capability of the timers.

Stepper Motor Clock Generation by Using a PSD5XX (Cont.)

Figure 9. Logic Configuration for PSD5XX in Pulse Mode



Stepper Motor Clock Generation by Using a PSD5XX (Cont.)

Figure 10. A Sample PPLD Configuration in an *.abl File for the PSD5XX

```
"PPLD Equation for the Timer to Preload

mc2tmr0  =  (!timerout0);
mc2tmr1  =  (!timerout1);
mc2tmr2  =  (!timerout2);
mc2tmr3  =  (!timerout3);
```

80C186 Interface to the PSD503

Figure 11 shows a block diagram of a PSD5XX family product. In this design the PSD503 is used. The PSD503 is configured to 64K x 16 EPROM in MUX mode. The address and data on the 80C186 are multiplexed so the PSD503 latches the address internally. The address lines A16 and A17 are internally latched using PA6 and PA5 from the PSD503 ports. Ports PC0 – PC7, PD0 – PD7, PE3 and PE4 on the PSD503 are used to output the address A0 – A17 externally to be used by the 128K x 16 SRAM external to the PSD503 device. PA0 through PA3 are used as timer outputs to provide clocks for the stepper motor control. Figure 12 shows the schematic for the processor connection to the PSD503 and Figure 13 shows a schematic for a typical stepper motor control unit interface to the PSD503. The stepper motor interface control uses PB0 – PB5 to control the four L297 stepper motor control chips. PB0 and PB1 are used to enable and disable the four axis of the motion. PB2 through PB5 are used to control the direction of the motor motion. PB0 through PB7 are configured in the software. Figure 14 shows the *.ABL file used in this design.

Figure 11. PSD5XX Block Diagram

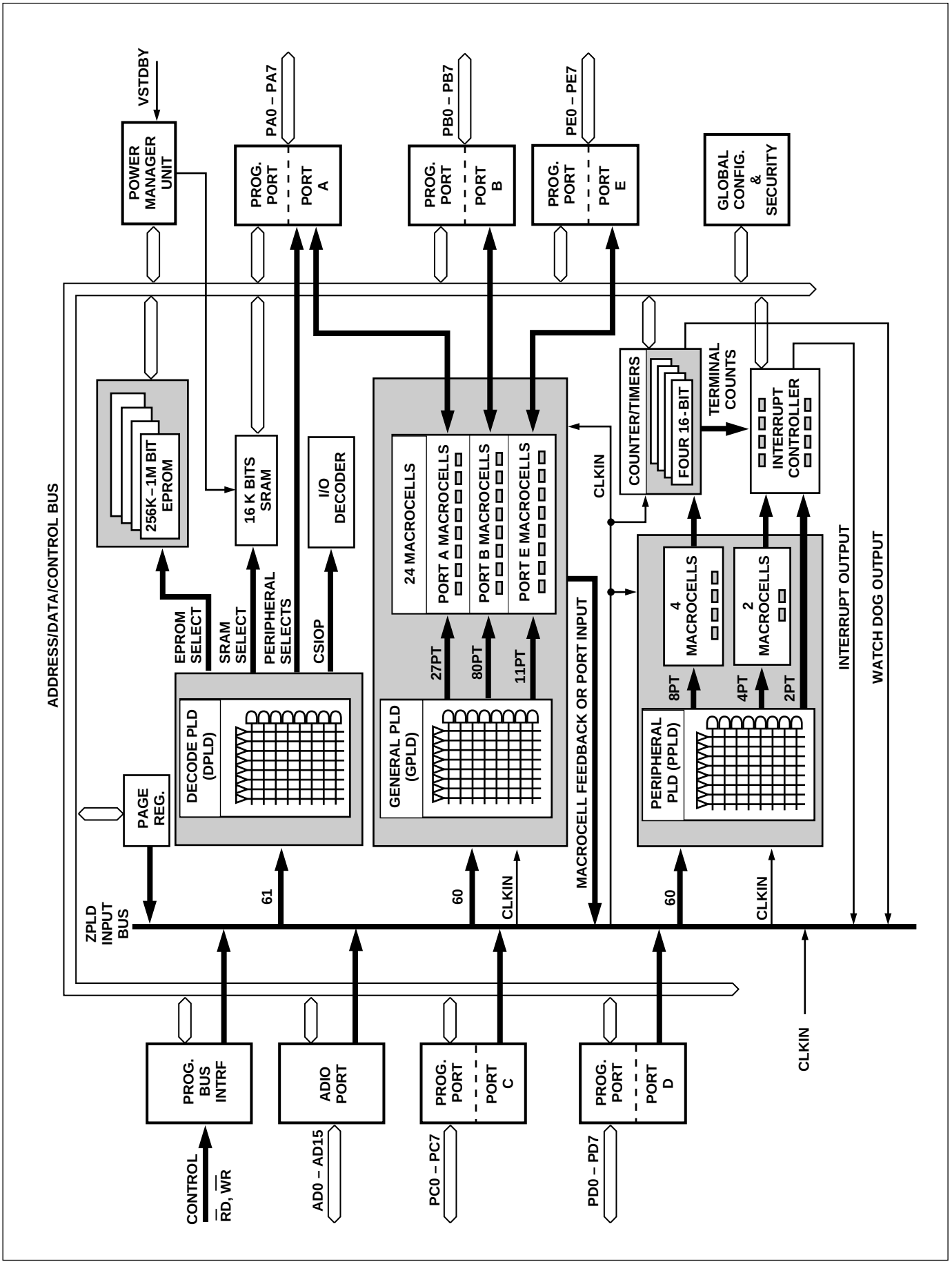


Figure 12. Schematic for the PSD503 Interface to a 80C186 Processor

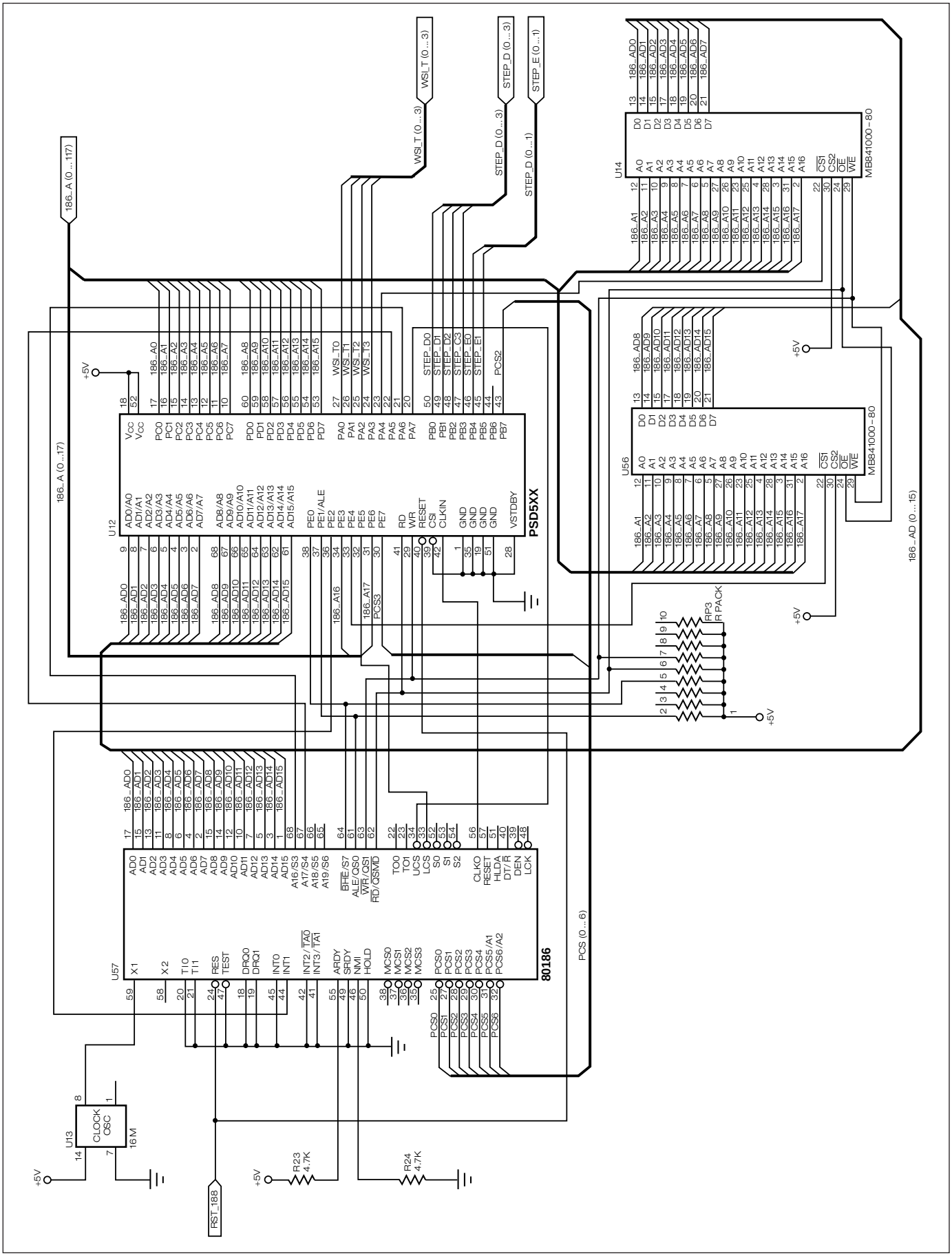
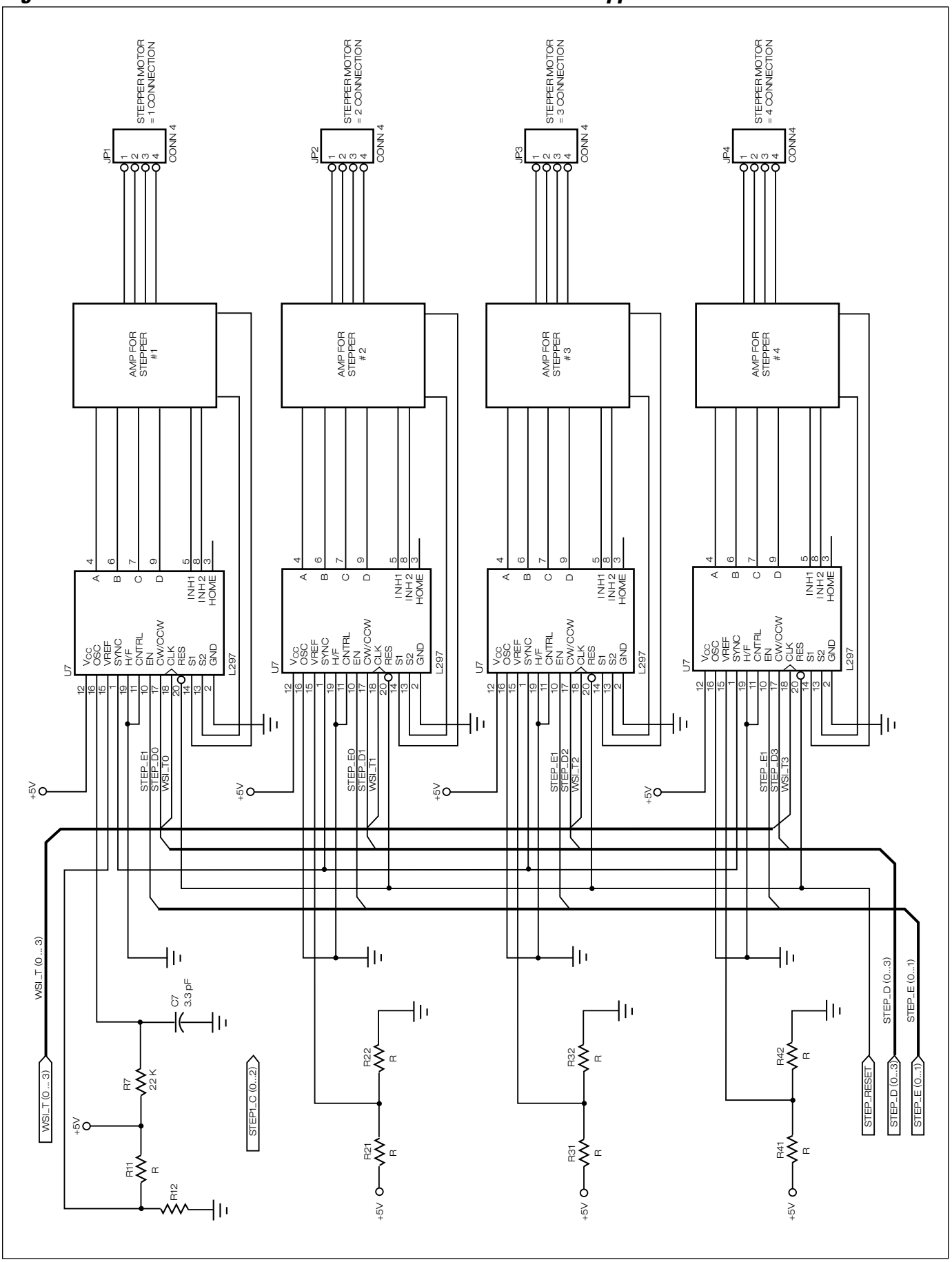


Figure 13. Schematic Interface Between PSD503 and the L297 Stepper Controller



80C186 Interface to the PSD503 (Cont.)

Figure 14. Program Listing for the ABEL File Used in this Design

```
module mfhs_16
title 'DESIGN FOR PSD503 ABEL source file to interface with 80C186';


"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
wr     pin;
rd     pin;
bhe    pin;

a16  pin 21;          "high order address input
a17  pin 22;          "high order address input
add_16  pin 34;       "address 16 latched output
add_17  pin 31;       "address 17 latched output

" PINS DEDFINED BY NPK "

umcs  pin;            " Upper Memory Chip Select
lmcs  pin;            " Lower Memory Chip Select

emcs  pin;            " even memory chip select for external SRAM
omcs  pin;            " odd memory chip select for external SRAM

pcs3  pin;            " PSD Upper 256 bytes address chip select space
pcs2  pin;            " PSD Lower 256 bytes address chip select space

pb0, pb1, pb2, pb3, pb4, pb5 pin;   " Stepper motor Control Port

" Timer Contol Pins "

timerout0 pin;        " Stepper 1 Clock 1
timerout1 pin;        " Stepper 2 Clock 2
timerout2 pin;        " Stepper 3 Clock 3
timerout3 pin;        " Stepper 4 Clock 4


" Port Control "

pd0, pd1, pd2, pd3, pd4, pd5, pd6, pd7 pin;       " Upper Address Output "

pc0, pc1, pc2, pc3, pc4, pc5, pc6, pc7 pin;       " Lower Address Output "

clkin, reset pin;       "using the reserved names.


"Output signals

csiop, rs0, es0, es1, es2, es3 node ;    "DPLD output chip selects
mc2tmr0 node;       " PPLD Output To Timer 0 "
mc2tmr1 node;       " PPLD Output To Timer 1 "
mc2tmr2 node;       " PPLD Output To Timer 2 "
mc2tmr3 node;       " PPLD Output To Timer 3 "
```

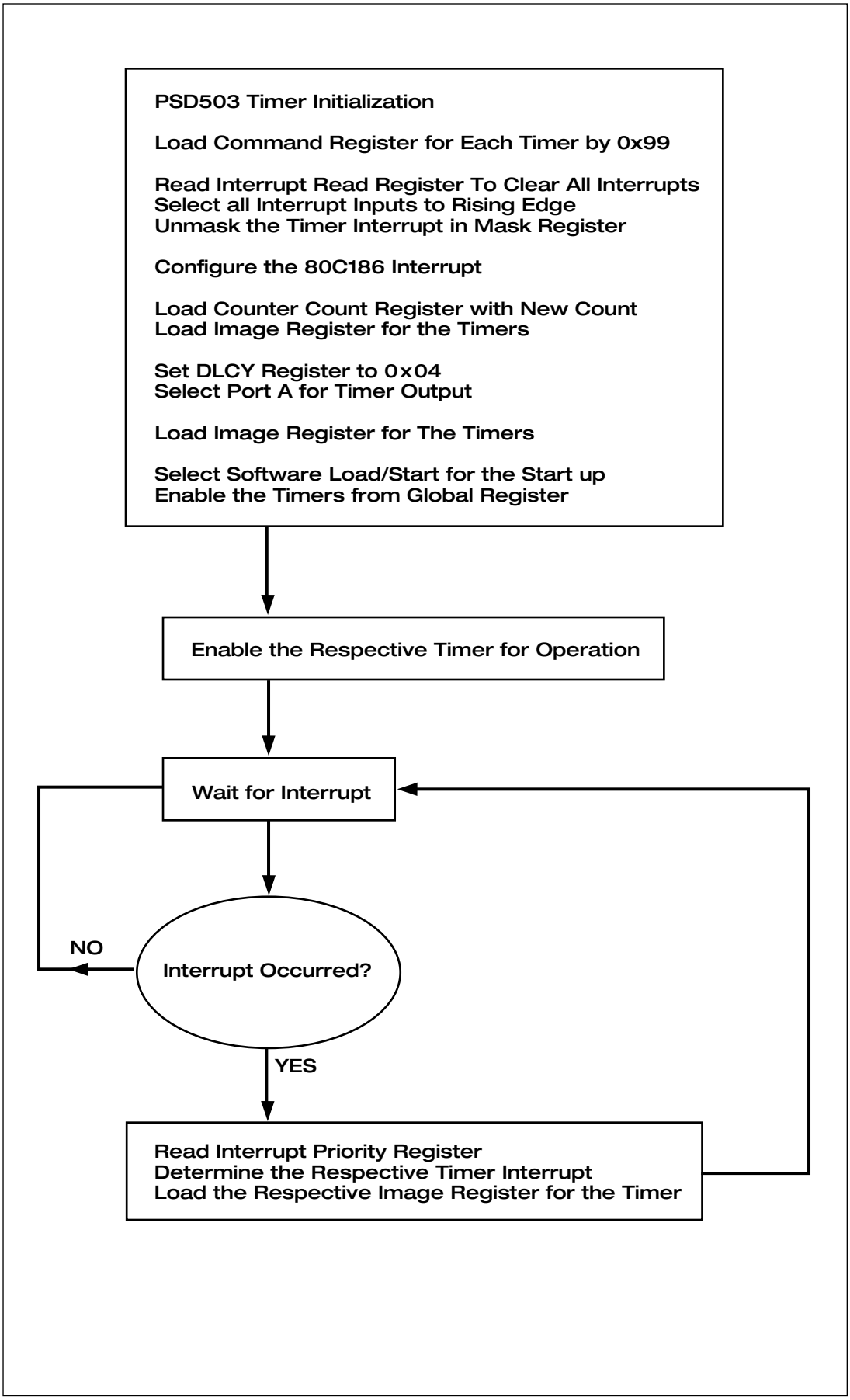
80C186 Interface to the PSD503 (Cont.)

Figure 14. Program Listing for the ABEL File Used in this Design (Cont.)

```
"General outputs

"DEFINITIONS


"page =  [pgr3,pgr2,pgr1,pgr0];"
CK    =  .c. ;     " Clock pulse definition
X     =  .x. ;     " Don't care
Address  =  [a16,a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

Add  =  [pc7,pc6,pc5,pc4,pc3,pc2,pc1,pc0];



equations

"DPLD EQUATIONS

csiop  =  ((Address >= ^h00100) & (Address <= ^h001ff));
rs0    =  0; " Disable The 2k On Board SRAM "
es0    =  (Address >= ^h00000) & (Address <= ^h07fff) & (!umcs);   " 32k block 0
es1    =  (Address >= ^h08000) & (Address <= ^h0ffff) & (!umcs);   " 32k block 1
es2    =  (Address >= ^h10000) & (Address <= ^h17fff) & (!umcs);   " 32k block 2
es3    =  (Address >= ^h18000) & (Address <= ^h1ffff) & (!umcs);   " 32k block 3


add_16  =  a16;        " Address 16 latched output "
add_17  =  a17;        " Address 17 latched output "

emcs  =  (!lmcs & bhe & !a0)  +  (!lmcs & !bhe & !a0);    " even address SRAM chip select
omcs  =  (!lmcs & !bhe & a0)  +  (!lmcs & !bhe & !a0);    " odd address SRAM chip select

"PPLD Equations

mc2tmr0 =  (!timerout0);    " Pre Load Timer 0 "
mc2tmr1 =  (!timerout1);    " Pre Load Timer 1 "
mc2tmr2 =  (!timerout2);    " Pre Load Timer 2 "
mc2tmr3 =  (!timerout3);    " Pre Load Timer 3 "


" ***************************************************************************************************
"                                    TEST VECTORS
" ***************************************************************************************************

END  mfhs_16
```

80C186 Interface to the PSD503 (Cont.)

Figure 15. Block Diagram for the Register Configuration and Interrupt Operation



80C186 Interface to the PSD503 (Cont.)

Figure 16. A Sample *.C Program for this Application (Cont.)

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

typedef unsigned short  USHORT;
typedef short           SHORT;
typedef unsigned long*  PULONG;

#include “x_step.dat”                          /* Stepper Profile Table */

#define PCS0  0x000
#define PCS1  0x080
#define PCS2  0x100
#define PCS3  0x100
#define PCS4  0x200
#define PCS5  0x280

/* WSi Registers
 */

 #define   WSiINTRREAD      PCS2+0xD4      /* Interrupt Read Clear */
 #define   WSiINTRMASK      PCS2+0xD3      /* Interrpt Mask Register */
 #define   WSiINTRMODE      PCS2+0xD2      /* Interrupt Edge/Level */
 #define   WSiINTRREQ       PCS2+0xD1      /* Interrupt Request Latch */
 #define   WSiINTRPRI       PCS2+0xD0      /* Interrupt Priority */

 #define   WSiSLR           PCS2+0xa5      /* Software Load/Stor */

 #define   WSiCNTR0         PCS2+0x98      /* Timer 0 control */
 #define   WSiCNTR1         PCS2+0x9A      /* Timer 1 control */
 #define   WSiCNTR2         PCS2+0x9C      /* Timer 2 control */
 #define   WSiCNTR3         PCS2+0x9E      /* Timer 3 control */

 #define   WSiCMD0          PCS2+0xa0      /* TImer 0 Control Register */
 #define   WSiCMD1          PCS2+0xa1      /* TImer 1 Control Register */
 #define   WSiCMD2          PCS2+0xA2
 #define   WSiCMD3          PCS2+0xA3
 #define   WSiDLCY          PCS2+0xa6      /* Scale Factor Control Of Timers */
 #define   WSiIMG0          PCS2+0x90      /* Timer 0 Image Register*/
 #define   WSiIMG1          PCS2+0x92      /* Timer 1 Image Register*/
 #define   WSiIMG2          PCS2+0x94      /* Timer 0 Image Register*/
 #define   WSiIMG3          PCS2+0x96      /* Timer 1 Image Register*/

 #define   WSiGLBREG        PCS2+0xa8      /* Timers Global Register */
 #define   WSiSFR           PCS2+0x08      /* Special function register for port A */
 #define   WSiFREEZ         PCS2+0xA4

 #define   WSiPORTB_CNTR    PCS2+0x03      /* Port B Configuration */
 #define   WSiPORTB_DIR     PCS2+0x07

 #define   WSiPORTC_CNTR    PCS2+0x12      /* Port C Configuration */
 #define   WSiPORTC_DIR     PCS2+0x16

 #define   WSiPORTD_CFG     PCS2+0x13      /* Port D Configuration */
 #define   WSiPORTD_DIR     PCS2+0x17

 #define   WSiPORTE_SFR     PCS2+0x28      /* Port E Configuration */
 #define   WSiPORTE_DIR     PCS2+0x26
```

80C186 Interface to the PSD503 (Cont.)

Figure 16. A Sample *.C Program for this Application (Cont.)

```
#define     WSiPORTE_OUT     PCS2+0x24
#define     WSiPORTE_IN      PCS2+0x20
#define     WSiPORTE_CFG     PCS2+0x22


#define     PULSE_MODE_DISABLED     0x99      /* was 99 */
#define     ENABLE                  0x04      /* was 00*/
/* 188 Registers
 */
#define     I0CON       0xFF38
#define     I1CON       0xFF3A
#define     IMASK       0xFF28
#define     EOI         0xFF22

/*
        Global Variables Here
*/

int    Stepper_1_Total_Step_Count;
int    Step_1_Max_Slew_Count;
int    Step_1_Motion_Index;
int    Step_1_Motion_Stat;
int    Step_1_Slew_Count;
int    Stepper_1_Ramp_Up_Count;
int    Stepper_1_Ramp_Down_Count;
int    Stepper_1_Step_Count;
int    Stepper_1_Step_Time;
int    Step_1_Time;
int    Step_1_Count_Old;
int    Stepper_1_Profile[20];

}

USHORT    Image;
USHORT    MotorNum;
/*
    INTERRUPT 1 ROUTINE
*/

USHORT Port  =  WSiINTRREAD;
USHORT  iw;
void _interrupt WSiHandler(void)
 {
  static USHORT Read;
    Image  =  0x200;
    switch( inportb(WSiINTRPRI) & 0x07)
        {
```

80C186 Interface to the PSD503 (Cont.)

Figure 16. A Sample *.C Program for this Application (Cont.)

```
        case 0:
                outport(WSiIMG0, Step_1_Time);        /* Load Timer 0 for step pulse */
                Stepper_1_Step_Count++;
                break;
        case 1:
                outport(WSiIMG1, Image);
                break;
        case 2:
                outport(WSiIMG2, Image);
                break;
        case 3:
                outport(WSiIMG3, Image);
                break;
        }
  outport(EOI,0x8000);
  _enable();
  }

void Init_Timers(void)
 {
   PULONG pIVT=NULL;
   _disable();
   Image = 100;

 /* Pulse Mode Timer 0-3
  */
  outportb(WSiCMD0, 0x99);              /* Program Command Register For The Counters */
  outportb(WSiCMD1, 0x99);              /* All Counters to Pulse Mode */
  outportb(WSiCMD2, 0x99);              /* All Counters Disabled */
  outportb(WSiCMD3, 0x99);

 /* Interrupt WSi Setup
  */
  inportb( WSiINTRREAD);                /* Clear All The Interrupts */
  outportb(WSiINTRMODE,0x00);
  outportb(WSiINTRMASK,0x0f);           /* Unmask Timers Interrupt */

  outportb(WSiPORTE_SFR,0x0d);          /* Confgure Port E For Special Function */

 /* Interrupt 188 Setup
  */
  pIVT[12] = (PULONG)SensorInt0;        /* Sensor Interrupt */
  outportb(I0CON,0x012);                /* Disable sensor inerrupt */
  pIVT[13] = (PULONG)WSiHandler;        /* Interrupt # 1 Initilization */
  outportb(I1CON,0x010);                /* Was level sensetive 0x07 */
  outportb(IMASK,0xDD);

  Image = 0;
```

80C186 Interface to the PSD503 (Cont.)

Figure 16. A Sample *.C Program for this Application (Cont.)

```
 outport(WSiCNTR0,    0x00);
 outport(WSiCNTR1,    0x00);
 outport(WSiCNTR2,    0x00);
 outport(WSiCNTR3,    0x00);

 outportb(WSiDLCY,    0x04);
 outportb(WSiSFR,     0x0f);

 outport( WSiIMG0,    Image+0x340);
 outport( WSiIMG1,    Image+0x300);
 outport( WSiIMG2,    Image+0x260);
 outport( WSiIMG3,    Image+0x220);

 outportb(WSiSLR,0x0F);
 outportb(WSiGLBREG,0x02);      /* Configure The Wsi Global Register */


 _enable();                     /* Enable Interrupt */

 }
/*
  This Routine Sets up timer 0 for the start up profile
*/
void Step_1_Init(void)
{
 static SHORT s_en1,s1_c;
 static SHORT c_r1,c_r2,m_c;

    Stepper_1_Step_Count = 0;
    Step_1_Count_Old = 0;

      Stepper_1_Step_Count = 0;
      Stepper_1_Total_Step_Count =1000;
      Step_1_Max_Slew_Count = 998;
      Step_1_Motion_Index = 0;
      Step_1_Slew_Count = 0;
      Stepper_1_Ramp_Up_Count = 5;
      Stepper_1_Ramp_Down_Count = 7;
      Step_1_Motion_Stat = 0;               /* Set Up For Ramp Up */
      Step_1_Time = 0x3000;
  outportb(Step_Motor1_Control,s_en1);      /* Reset Motor State */

 outportb(WSiCMD0, 0x9d);                   /* Enable Timer 0 for stepper 1 */

}
/*
This routin is used to update the profile table for motor 1
*/

void Stepper_1_Move(void)
{
 if( Stepper_1_Step_Count > Step_1_Count_Old )
 {
```

80C186 Interface to the PSD503 (Cont.)

Figure 16. A Sample *.C Program for this Application (Cont.)

```
    if( Step_1_Motion_Stat == 0 )
    {
        Step_1_Motion_Index++;              /* ***** RAMP UP STEPPER 1 ***** */
        Step_1_Time = x_axis[Step_1_Motion_Index];
        if( Step_1_Motion_Index == 132 )
        {
            Step_1_Motion_Stat = 1;         /* Set Status For Slew */
        }
    }

    if( Step_1_Motion_Stat == 1 )
    {
        Step_1_Slew_Count++;                /* ***** SLEW FOR STEPPER 1 ***** */
        if( Step_1_Slew_Count  ==  Step_1_Max_Slew_Count )
        {
            Step_1_Motion_Stat  =  2;       /* Set Status For Ramp Down */
            Step_1_Motion_Index = 132;
        }
    }

    if( Step_1_Motion_Stat  ==  2 )
    {
        Step_1_Motion_Index—;               /* ***** RAMP DOWN FOR STEPPER 1 ***** */
        if( Step_1_Motion_Index != 0 )
        {
            Step_1_Time  =  x_axis[Step_1_Motion_Index];
        }
        if( Step_1_Motion_Index ==  0 )
        {
            Stepper_1_Step_Count  =  0;
            Step_1_Count_Old = 0;
            outportb(WSiCMD0, 0x99);        /* Disable Motor */
            outportb(WSiCMD0, 0x99);        /* Disable Motor. This is Just For Ice */
       }
    }


  Step_1_Count_Old = Stepper_1_Step_Count;
  }
 }

}
```

80C186 Interface to the PSD503 (Cont.)

Figure 16. A Sample *.C Program for this Application (Cont.)

```
main()
 {
  static USHORT Read, y, d1=0xAA,d2=0xAA;
  static USHORT key;

 Init_Timers();

 Step_1_Init();

      key = 1;
while(1)
{
  switch( key )
        {
        case 1:
              Stepper_1_Move();
              break;
        case 2:
              stp_2();
              break;
        case 3:
              dcm_1();
              break;
        case 4:
              dcm_2();
              break;
        case 5:
              cres_12();
              break;
        case 6:
              C188_152();
              break;

        }
 }
 return 0;
 }
```

Software Configuration of the PSD503

Figure 15 shows a block diagram of the steps needed to configure the registers of the PSD503 for this application. Figure 16 shows a sample software program written in C that is used in this application to configure the PSD503. This software programs the special function register of Port A to be used as the timer outputs. Figure 17 shows the PSDSOFT configuration of the timers. The PSD503 must be configured through PSDSOFT for the BUS type, WR, RD, INTR and PORT operation.

The timer clock frequency is configured through the DLCY register to 1MHz. As the step rate increases the step rate accuracy deteriorates due to the quantization effect. The quantization effect is not a problem in this application. The output pulse width of each timer is one microsecond which is sufficient for this application.

Figure 17. PSDsoft Configuration of the Timers

Counter / Timer 0:	Waveform/Pulse Mode.
Counter / Timer 1:	Pulse Output.
Counter / Timer 2:	Waveform/PulseMode.
Counter / Timer 3:	Pulse Output
Do you need Automatic Power Down Clock Input ?	NO
Do you want to set the security bit ?	NO
Do you need the Intr output signal ?	YES

Conclusion

In this application the PSD503 provided a very useful integrated means of design. The following were benefited from this design:

- 64 K x 16 EPROM
- Eighteen bits of latched output for demultiplexing ADDRESS from DATA.
- An 8-bit Interrupt Controller Equivalent to an 8259.
- Four 16-bit preloadable timers with a prescaler for the timer clocks.
- Logic for decoding.
- Programmable external PORTS.

The board space reduction and the amount of noise reduction that resulted from this design is immeasurable.

Design for PSD503 ABEL Source File to Interface with 80C186

****

W S I - PSDsoft Version 1.05B
Output of PSD Fitter

****

TITLE : DESIGN FOR PSD503 ABEL source file to interface with 80C186
PROJECT : mfhs_16 DATE : 04/07/1995
DEVICE : PSD503B1 TIME : 09:31:05
FIT OPTION : Keep Current

****

Pin Assignment

	1] GND	GND [35	
Address/Data Bus ADIO_7	2] adio7	pe2 [36	introut
Address/Data Bus ADIO_6	3] adio6	pe1 [37	ale
Address/Data Bus ADIO_5	4] adio5	pe0 [38	bhe
Address/Data Bus ADIO_4	5] adio4	csi [39	csi
Address/Data Bus ADIO_3	6] adio3	reset [40	reset
Address/Data Bus ADIO_2	7] adio2	rd [41	rd
Address/Data Bus ADIO_1 (a1)	8] adio1	clkin [42	clkin
Address/Data Bus ADIO_0 (a0)	9] adio0	pb7 [43	pcs2
pc7	10] pc7	pb6 [44	(Not Used)
pc6	11] pc6	pb5 [45	pb5
pc5	12] pc5	pb4 [46	pb4
pc4	13] pc4	pb3 [47	pb3
pc3	14] pc3	pb2 [48	pb2
pc2	15] pc2	pb1 [49	pb1
pc1	16] pc1	pb0 [50	pb0
pc0	17] pc0	GND [51	
	18] VCC	VCC [52	
	19] GND	pd7 [53	pd7
umcs	20] pa7	pd6 [54	pd6
a16	21] pa6	pd5 [55	pd5
a17	22] pa5	pd4 [56	pd4
omcs	23] pa4	pd3 [57	pd3
timerout3	24] pa3	pd2 [58	pd2
timerout2	25] pa2	pd1 [59	pd1
timerout1	26] pa1	pd0 [60	pd0
timerout0	27] pa0	adio15 [61	Address/Data Bus ADIO_15 (a15)
	28] VSTBY	adio14 [62	Address/Data Bus ADIO_14 (a14)
wr	29] wr	adio13 [63	Address/Data Bus ADIO_13 (a13)
pcs3	30] pe7	adio12 [64	Address/Data Bus ADIO_12 (a12)
add_17	31] pe6	adio11 [65	Address/Data Bus ADIO_11 (a11)
lmcs	32] pe5	adio10 [66	Address/Data Bus ADIO_10 (a10)
emcs	33] pe4	adio9 [67	Address/Data Bus ADIO_9 (a9)
add_16	34] pe3	adio8 [68	Address/Data Bus ADIO_8 (a8)

Global Configuration

Data Bus : 16 Multiplexed
ALE/AS Signal : Active High
WatchDog Mode : Off
Security Protection : Off

Address & Data Bus Assignment

Stimulus Bus Name			Signal Description
`adiol	= adio[7:0]	=	Address/Data Bus ADIO_7 – ADIO_0
`adioh	= adio[15:8]	=	Address/Data Bus ADIO_15 – ADIO_8
adio	= adio[15:0]	=	Address/Data Bus ADIO_15 – ADIO_0

Design for PSD503 ABEL Source File to Interface with 80C186 (Cont.)

Resource Usage Summary

Device Resources	Used/Total	Percentage
Port A: (pin 20 – pin 27)		
I/O pins	8 / 8	100 %
MCU I/O or Address Out	0 / 8	0 %
Peripheral I/O	0 / 8	0 %
ZPLD Inputs	3 / 8	37 %
ZPLD Combinatorial Outputs	1 / 8	12 %
ZPLD Registered Outputs	0 / 8	0 %
Other Information		
Buried Macrocells	0 / 7	0 %
Product Terms	1 / 27	3 %
Timer Outputs	4 / 4	100 %
Port B: (pin 43 - pin 50)		
I/O pins	7 / 8	87 %
MCU I/O or Address Out	7 / 8	87 %
ZPLD Inputs	0 / 8	0 %
ZPLD Combinatorial Outputs	0 / 8	0 %
ZPLD Registered Outputs	0 / 8	0 %
Other Information		
Buried Macrocells	0 / 8	0 %
Product Terms	0 / 80	0 %
Timer Outputs	0 / 4	0 %
Port C: (pin 10 - pin 17)		
I/O Pins	8 / 8	100 %
MCU I/O or Address Out	8 / 8	100 %
ZPLD Input Pins	0 / 8	0 %
Data Port (Non-Mux Bus)	0 / 8	0 %
Port D: (pin 53 - pin 60)		
I/O Pins	8 / 8	100 %
MCU I/O or Address Out	8 / 8	100 %
ZPLD Input Pins	0 / 8	0 %
Data Port (16-Bit Non-Mux Bus)	0 / 8	0 %
Port E: (pin 30 - pin 34, pin 36 - pin 38)		
I/O pins	8 / 8	100 %
MCU I/O or Address Out	1 / 8	12 %
ZPLD Inputs	1 / 8	12 %
ZPLD Combinatorial Outputs	3 / 8	37 %
ZPLD Registered Outputs	0 / 8	0 %
Control Signal Inputs	2 / 2	100 %
Timer Control Inputs	0 / 4	0 %
Interrupt Control Output	1 / 1	100 %
APD Clock Input	0 / 1	0 %
Terminal Counts (TC)	0 / 4	0 %
Other Information		
Buried Macrocells	0 / 5	0 %
Product Terms	3 / 11	27 %
Counter/Timer: Embedded Nodes		
Product Terms	4 / 8	50%
Interrupt: Embedded Nodes		
Product Terms	0 / 4	0%

Design for PSD503 ABEL Source File to Interface with 80C186 (Cont.)

OMC Resource Assignment

Resources Used	User Name		
Port A :			
macro cell 4	omcs	(mc_pa4)	=> Combinatorial
Port B:			
Port E:			
macro cell 3	add_16	(mc_pe3)	=> Combinatorial
macro cell 4	emcs	(mc_pe4)	=> Combinatorial
macro cell 6	add_17	(mc_pe6)	=> Combinatorial

EQUATIONS

DPLD EQUATIONS:

```
es0   =  !a15 & !a16 & !umcs;
es1   =  a15 & !a16  & !umcs;
es2   =  !a15 & a16  & !umcs;
es3   =  a15 & a16   & !umcs;
rs0   =  0;
csiop =  !a15 & !a14 & !a13 & !a12 & !a11 & !a10 & !a9 & a8 & !a16;
```

TIMER EQUATIONS:

```
mc2tmr0 = !timerout0;
mc2tmr1 = !timerout1;
mc2tmr2 = !timerout2;
mc2tmr3 = !timerout3;
```

INTERRUPT EQUATIONS:

PORT A EQUATIONS:

```
omcs = !bhe & !lmcs;
[omcs].OE = 1;
```

PORT B EQUATIONS:

PORT E EQUATIONS:

```
add_16 = a16;
emcs   = !a0 & !lmcs;
add_17 = a17;
[add_16, emcs, add_17].OE = 1;
```