



AP-248

**APPLICATION
NOTE**

Using The 8096

IRA HORDEN
MCO APPLICATIONS ENGINEER

September 1987



Order Number: 270061-002

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1-800-879-4683

Using The 8096	CONTENTS	PAGE
	1.0 INTRODUCTION	1
	2.0 8096 OVERVIEW	1
	2.1. General Description	1
	2.1.1. CPU Section	2
	2.1.2. I/O Features	4
	2.2. The Processor Section	4
	2.2.1. Operations and Addressing Modes	4
	2.2.2. Assembly Language	7
	2.2.3. Interrupts	8
	2.3. On-Chip I/O Section	10
	2.3.1. Timer/Counters	10
	2.3.2. HSI	11
	2.3.3. HSO	12
	2.3.4. Serial Port	13
	2.3.5. A to D Converter	16
	2.3.6. PWM Register	17
	3.0 BASIC SOFTWARE EXAMPLES	19
	3.1. Using the 8096's Processing Section	19
	3.1.1. Table Interpolation	19
	3.1.2. PL/M-96	22
	3.2. Using the I/O Section	24
	3.2.1. Using the HSI Unit	24
	3.2.2. Using the HSO Unit	25
	3.2.3. Using the Serial Port in Mode 1	29
	3.2.4. Using the A to D	31
	4.0 ADVANCED SOFTWARE EXAMPLES	31
	4.1. Simultaneous I/O Routines under Interrupt Control	31
	4.2. Software Serial Port Using the HSIO Unit	34
	4.3. Interfacing an Optical Encoder to the HSI Unit	39
	5.0 HARDWARE EXAMPLE	51
	5.1. EPROM Only Minimum System	51
	5.2. Port Reconstruction	53
	6.0 CONCLUSION	54
	7.0 BIBLIOGRAPHY	54

CONTENTS PAGE

APPENDICES

Appendix A. Basic Software Examples A-1

A.1. Table Lookup 1 A-1

A.2. Table Lookup 2 A-3

A.3. PLM-96 Code with Expansion A-5

A.4. Pulse Measurement A-11

A.5. Enhanced Pulse Measurement A-13

CONTENTS PAGE

A.6. PWM Using the HSO A-15

A.7. Serial Port A-19

A.8. A to D Converter A-21

Appendix B. HSO and A to D Under Interrupt Control B-1

Appendix C. Software Serial Port C-1

Appendix D. Motor Control Program ... D-1

Figures

2-1.	8096 Block Diagram	1
2-2.	Memory Map	2
2-3.	SFR Layout	3
2-4.	Major I/O Functions	4
2-5.	Instruction Summary	5
2-6.	Instruction Format	7
2-7.	Interrupt Sources	8
2-8.	Interrupt Vectors and Priorities	8
2-9.	Interrupt Structure Block Diagram	9
2-10.	The PSW Register	10
2-11.	HSI Unit Block Diagram	11
2-12.	HSI Mode Register	11
2-13.	HSO Command Register	12
2-14.	HSO Block Diagram	12
2-15.	Serial Port Control/Status Register	13
2-16.	Baud Rate Formulas	14
2-17.	Baud Rate Values for 10, 11, 12 MHz	15
2-18.	Multiprocessor Communication	16
2-19.	A to D Result/Command Register	17
2-20.	PWM Output Waveforms	18
2-21.	PWM to Analog Conversion Circuitry	18
3-1.	Using the HSIO to Monitor Rotating Machinery	28
3-2.	Serial Port Level Conversion	30
4-1.	10-Bit Asynchronous Frame	35
4-2.	Optical Encoder and Waveforms	39
4-3.	Filtered Encoder Waveforms	40
4-4.	Schematic of Optical Encoder to 8096 Interface	41
4-5.	Motor Driver Circuitry	41
4-6.	Mode State Diagram	44
4-7.	Motor Control Modes	49
5-1.	Minimum System Configuration	52

Listings

3-1.	Include File DEMO96.INC	19
3-2.	ASM-96 Code for Table Lookup Routine 1	20
3-3.	ASM-96 Code for Table Lookup Routine 1	21
3-4.	PLM-96 Code for Table Lookup Routine 1	23
3-5.	32-Bit Result Multiply Procedure for PLM-96	23
3-6.	Measuring Pulses Using the HSI Unit	24
3-7.	Enhanced HSI Pulse Measurement Routine	25
3-8.	Generating a PWM with the HSO	26
3-9.	Changes to Declarations for HSO Routine	27
3-10.	Driver Module for HSO PWM Program	27
3-11.	Using the Serial Port in Mode 1	29
3-12.	Scanning the A to D Channels	31
4-1.	Using Multiple I/O Devices	32
4-2.	Software Serial Port Declarations	35
4-3.	Software Serial Port Interface Routines	36
4-4.	Software Serial Port Initialization Routine	36
4-5.	Software Serial Port Transmit Process	37
4-6.	Receive Process	37
4-7.	Motor Control HSO.0 Timer Routine	42
4-8.	Motor Control HSI Data Available Routine	44
4-9.	Motor Control Mode 1 Routines	45
4-10.	Motor Control Mode 0 Routines	46
4-11.	Motor Control Software Timer 1 Routine	47
4-12.	Motor Control Next Position Lookup	49
4-13.	Motor Control Timer Interrupt Routine	50
4-14.	Motor Control Software Timer Interrupt Handler	50
4-15.	Motor Control Software Timer 2 Routine	51

1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS®-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

2.0 8096 OVERVIEW

2.1. General Description

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096; the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

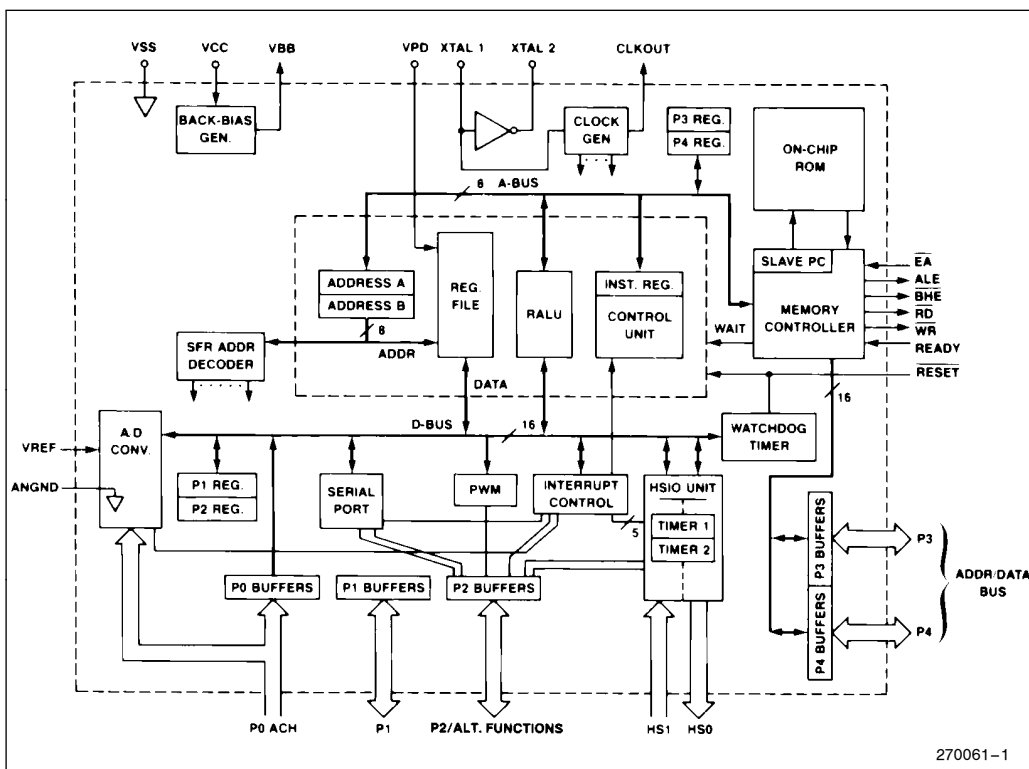


Figure 2-1. 8096 Block Diagram



2.1.1. CPU SECTION

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.

In the lower 24 bytes of the register file are the register-mapped I/O control locations, also called Special Function Registers or SFRs. These registers are used to control the on-chip I/O features. The remaining 232 bytes are general purpose RAM, the upper 16 of which can be kept alive using a low current power-down mode.

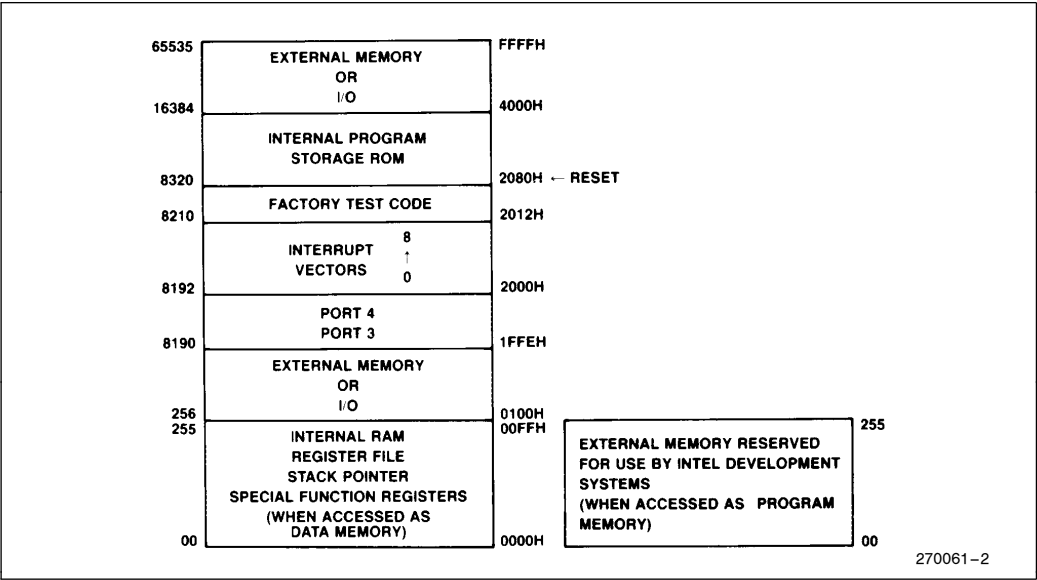


Figure 2-2. Memory Map



Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written

to. More information about the use of these registers is included in the description of the features which they control.

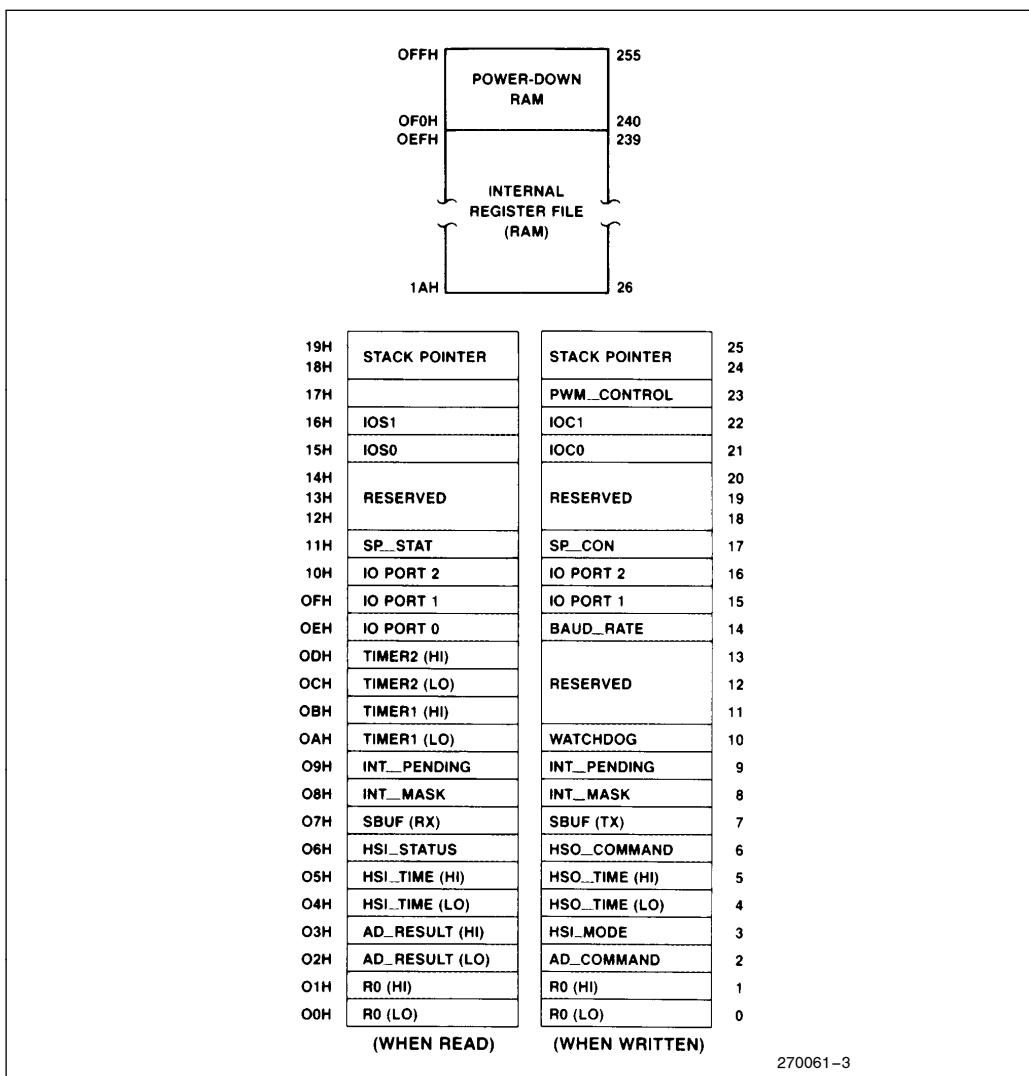


Figure 2-3: SFR Layout



2.1.2. I/O FEATURES

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

2.2. The Processor Section

2.2.1. OPERATIONS AND ADDRESSING MODES

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one, two or three operands. An example of a one operand instruction would be:

```
NOT Value1 ; Value1 : = 1's complement (Value1)
```

A two operand instruction would have the form:

```
ADD Value2,Value1 ; Value2 : = Value2 + Value1
```

A three operand instruction might look like:

```
MUL Value3,Value2,Value1 ;  
Value3 : = Value2* Value1
```

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

Major I/O Functions	
High Speed Input Unit	Provides Automatic Recording of Events
High Speed Output Unit	Provides Automatic Triggering of Events and Real-Time Interrupts
Pulse Width Modulation	Output to Drive Motors or Analog Circuits
A to D Converter	Provides Analog Input
Watchdog Timer	Resets 8096 if a Malfuction Occurs
Serial Port	Provides Synchronous or Asynchronous Link
Standard I/O Lines	Provide Interface to the External World when other Special Features are not needed

Figure 2-4. Major I/O Functions



Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	$D \leftarrow D + A$	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	$D \leftarrow B + A$	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	$D \leftarrow D + A + C$	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	$D \leftarrow D - A$	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	$D \leftarrow B - A$	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	$D \leftarrow D - A + C - 1$	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	$D - A$	✓	✓	✓	✓	↑	—	
MUL/MULU	2	$D, D + 2 \leftarrow D * A$	—	—	—	—	—	?	2
MUL/MULU	3	$D, D + 2 \leftarrow B * A$	—	—	—	—	—	?	2
MULB/MULUB	2	$D, D + 1 \leftarrow D * A$	—	—	—	—	—	?	3
MULB/MULUB	3	$D, D + 1 \leftarrow B * A$	—	—	—	—	—	?	3
DIVU	2	$D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$	—	—	—	✓	↑	—	2
DIVUB	2	$D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$	—	—	—	✓	↑	—	3
DIV	2	$D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$	—	—	—	?	↑	—	2
DIVB	2	$D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$	—	—	—	?	↑	—	3
AND/ANDB	2	$D \leftarrow D \text{ and } A$	✓	✓	0	0	—	—	
AND/ANDB	3	$D \leftarrow B \text{ and } A$	✓	✓	0	0	—	—	
OR/ORB	2	$D \leftarrow D \text{ or } A$	✓	✓	0	0	—	—	
XOR/XORB	2	$D \leftarrow D \text{ (excl. or) } A$	✓	✓	0	0	—	—	
LD/LDB	2	$D \leftarrow A$	—	—	—	—	—	—	
ST/STB	2	$A \leftarrow D$	—	—	—	—	—	—	
LDBSE	2	$D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$	—	—	—	—	—	—	3, 4
LDBZE	2	$D \leftarrow A; D + 1 \leftarrow 0$	—	—	—	—	—	—	3, 4
PUSH	1	$SP \leftarrow SP - 2; (SP) \leftarrow A$	—	—	—	—	—	—	
POP	1	$A \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
PUSHF	0	$SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$ $\text{PSW} \leftarrow 0000\text{H}$ $I \leftarrow 0$	0	0	0	0	0	0	
POPF	0	$\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2; I \leftarrow \text{✓}$	✓	✓	✓	✓	✓	✓	
SJMP	1	$PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LJMP	1	$PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
BR (indirect)	1	$PC \leftarrow (A)$	—	—	—	—	—	—	
SCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
RET	0	$PC \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
J (conditional)	1	$PC \leftarrow PC + 8\text{-bit offset (if taken)}$	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

Figure 2-5. Instruction Summary

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D - 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D - 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 - D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb ———— lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb ———— lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb ———— lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRV	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP - 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

Figure 2-5. Instruction Summary (Continued)

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.

5. Offset is a 2's complement number.

6. Specified bit is one of the 2048 bits in the register file.

7. The "L" (Long) suffix indicates double-word operation.

8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.

9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed addressing

mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

2.2.2. ASSEMBLY LANGUAGE

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

Mnem	Dest or Src1	; One operand direct
Mnem	Dest, Src1	; Two operand direct
Mnem	Dest, Src1, Src2	; Three operand direct
Mnem	#Src1	; One operand immediate
Mnem	Dest, #Src1	; Two operand immediate
Mnem	Dest, Src1, #Src2	; Three operand immediate
Mnem	[addr]	; One operand indirect
Mnem	[addr] +	; One operand indirect auto-increment
Mnem	Dest, [addr]	; Two operand indirect
Mnem	Dest, [addr] +	; Two operand indirect auto-increment
Mnem	Dest, Src1, [addr]	; Three operand indirect
Mnem	Dest, Src1, [addr] +	; Three operand indirect auto-increment
Mnem	Dest, offs [addr]	; Two operand indexed (short or long)
Mnem	Dest, Src1, offs [addr]	; Three operand indexed (short or long)

Where: "Mnem" is the instruction mnemonic
"Dest" is the destination register
"Src1", "Src2" are the source registers
"addr" is a register containing a value to be used in computing the address of an operand
"offs" is an offset used in computing the address of an operand

270061-B3

Figure 2-6. Instruction Format

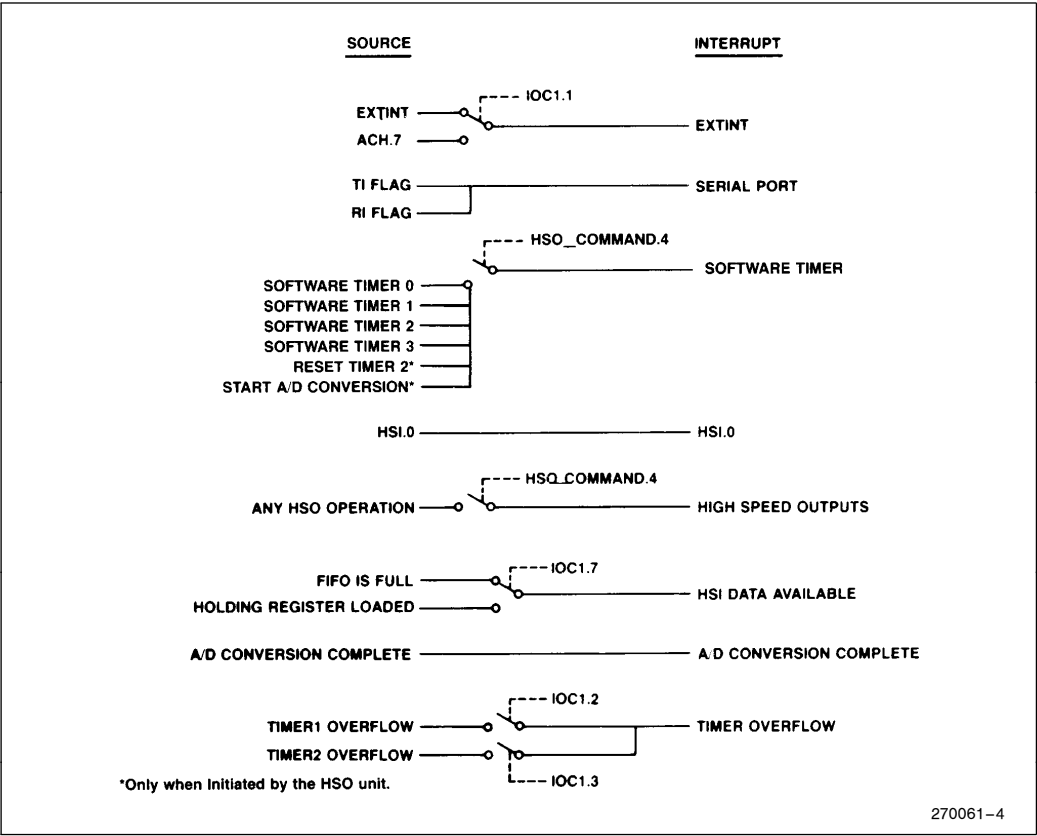


Figure 2-7. Interrupt Sources

2.2.3. INTERRUPTS

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT_PENDING), the Interrupt Mask Register (INT_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9 = 1, a vector will be taken to the address listed in the interrupt vector table for that

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-8. Interrupt Vectors and Priorities

source. When the vector is taken the INT_PENDING bit is cleared. If more than one bit is set in the INT_PENDING register with the corresponding bit set in the INT_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

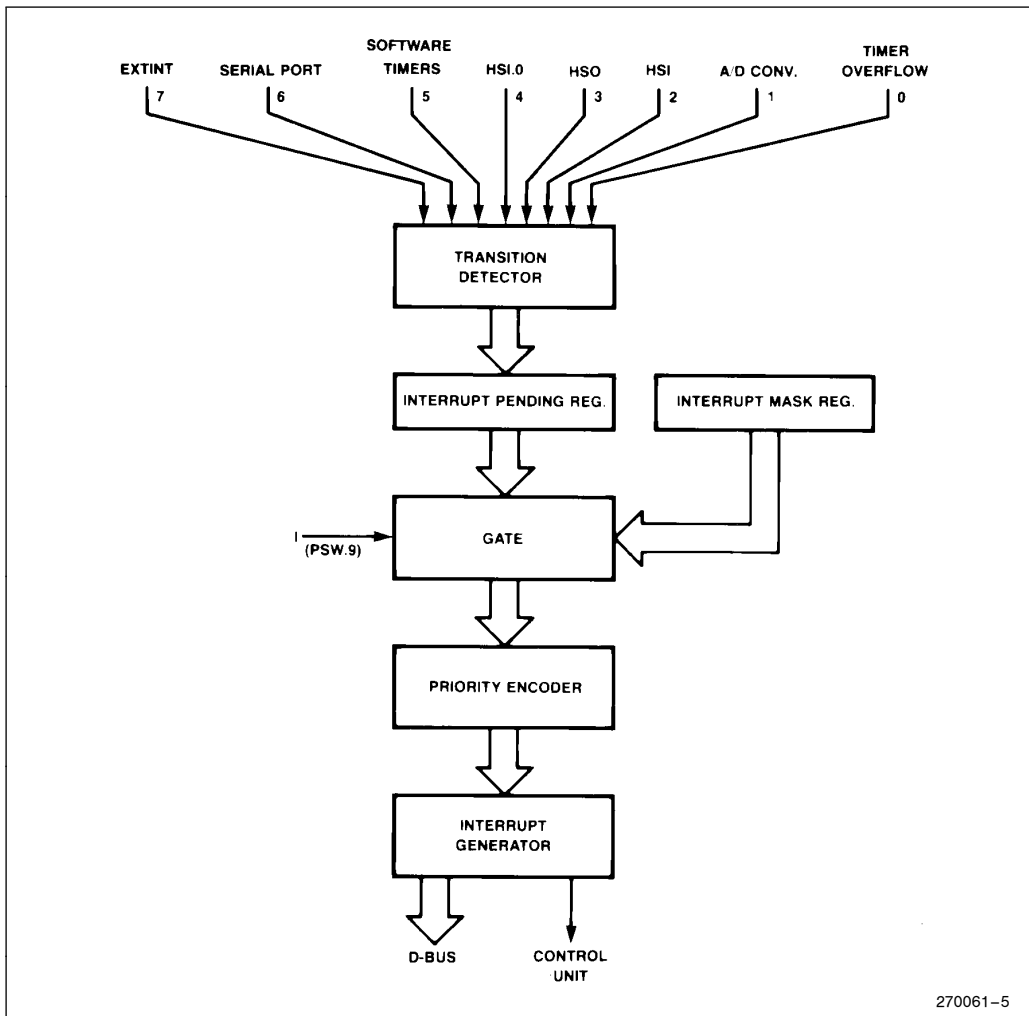


Figure 2-9. Interrupt Structure Block Diagram

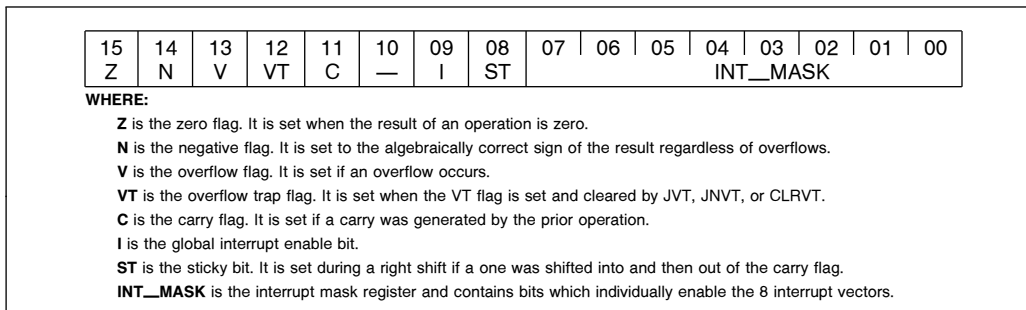


Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT_MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

```

INT    VECTOR:

    PUSHF
    LDB  INT_MASK, #xxxxxxxhB
    EI
    -
    - ;Insert service routine here
    -
    POPF
    RET
  
```

The PUSHF instruction saves the PSW including the old INT_MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT_MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a POPF in-

struction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

2.3. On-Chip I/O Section

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O, (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

2.3.1. TIMER/COUNTERS

The 8096 has two time bases, Timer 1 and Timer 2. Timer 1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz crystal.)

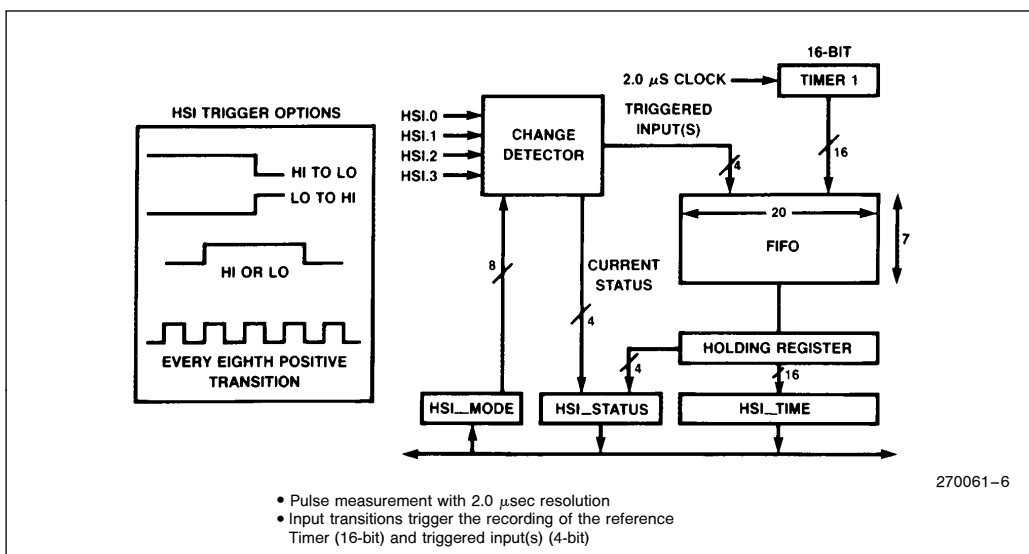


Figure 2-11. HSI Unit Block Diagram

Its value can be read at any time and used as a reference for both the HSI section and the HSO section. Timer 1 can cause an interrupt when it overflows, and cannot be modified or stopped without resetting the entire chip. Timer 2 is really an event counter since it uses an external clock source. Like Timer 1, it is 16-bits wide, can be read at any time, can be used with the HSO section, and can generate an interrupt when it overflows. Control of Timer 2 is limited to incrementing it and resetting it. Specific values can not be written to it.

Although the 8096 has only two timers, the timer flexibility is equal to a unit with many timers thanks to the HSIO unit. The HSI enables one to measure times of external events on up to four lines using Timer 1 as a timer base. The HSO unit can schedule and execute internal events and up to six external events based on the values in either Timer 1 or Timer 2. The 8096 also includes separate, dedicated timers for the baud rate generator and watchdog timer.

2.3.2. HSI

The HSI unit can be thought of as a message taker which records the line which had an event and the time at which the event occurred. Four types of events can trigger the HSI unit, as shown in the HSI block diagram in Figure 2-11. The HSI unit can measure pulse widths and record times of events with a 2

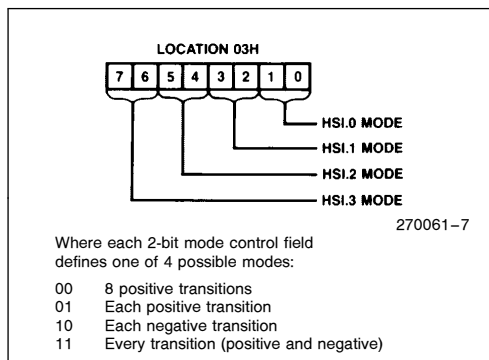


Figure 2-12. HSI Mode Register

microsecond resolution. It can look for one of four events on each of four lines simultaneously, based on the information in the HSI Mode register, shown in Figure 2-12. The information is then stored in a seven level FIFO for later retrieval. Whenever the FIFO contains information, the earliest entry is placed in the holding register. When the holding register is read, the next valid piece of information is loaded into it. Interrupts can be generated by the HSI unit at the time the

holding register is loaded or when the FIFO has six or more entries.

2.3.3.3. HSO

Just as the HSI can be thought of as a message taker, the HSO can be thought of as a message sender. At times determined by the software, the HSO sends mes-

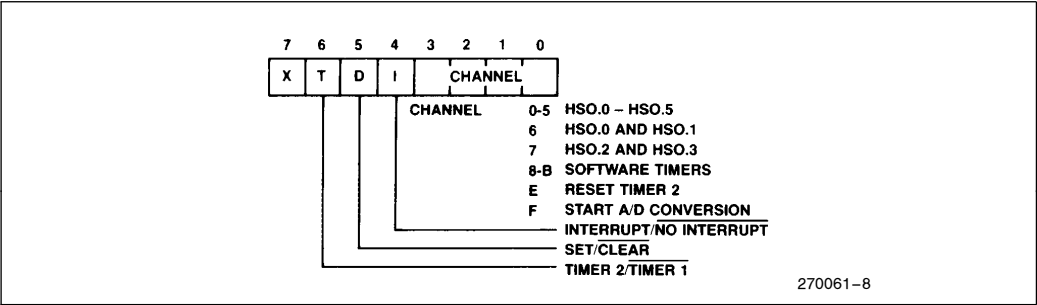


Figure 2-13. HSO Command Register

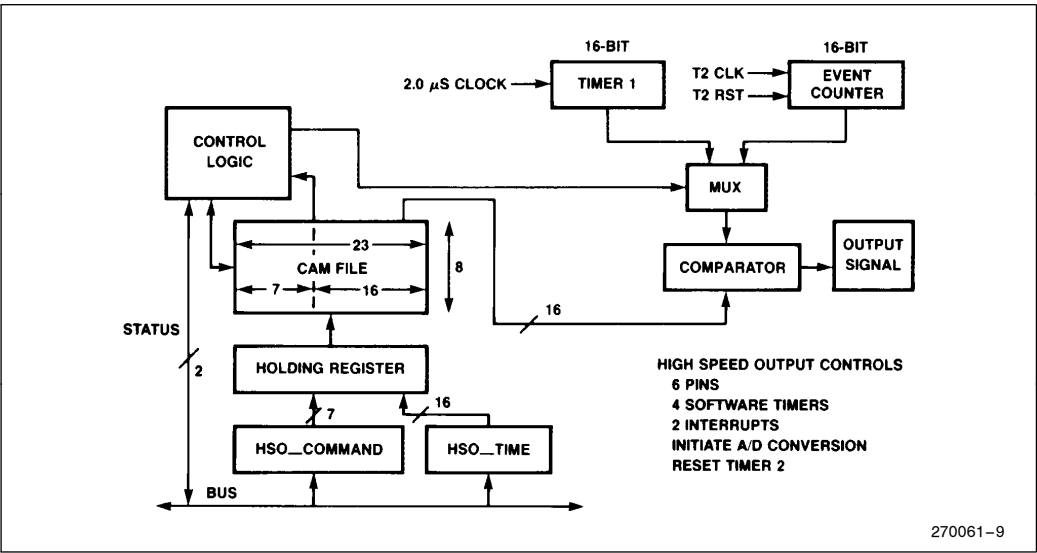


Figure 2-14. HSO Block Diagram

sages to various devices to have them turn on, turn off, start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multitask process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared with the two timers. After 8 state times, (two microseconds with a 12 MHz clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system.

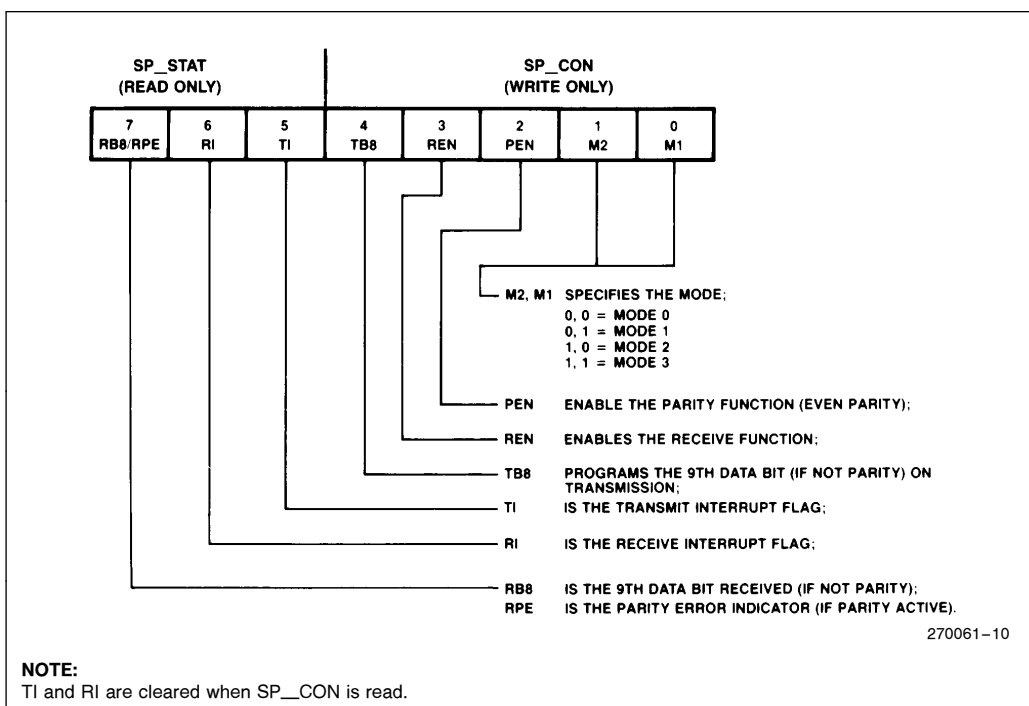


Figure 2-15. Serial Port Control/Status Register

The serial port is similar to that on the MCS-51 product line. It has one synchronous and three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONTROL/Serial Port STATUS) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2 and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:	
Mode 0:	$\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4 \cdot (B + 1)}; B \neq 0$
Others:	$\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64 \cdot (B + 1)}$
Using T2CLK:	
Mode 0:	$\text{Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$
Others:	$\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16 \cdot B}; B \neq 0$
Note that B cannot equal 0, except when using XTAL1 in other than mode 0.	

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eighths of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port BUffer. If parity is enabled, (i.e., PEN=1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (i.e., PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (i.e., PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

XTAL1 Frequency = 12.0 MHz		
Baud Rate	Baud Register Value	Percent Error
19.2K	8009H	+ 2.40
9600	8013H	+ 2.40
4800	8026H	− 0.16
2400	804DH	− 0.16
1200	809BH	− 0.16
300	8270H	0.00
XTAL1 Frequency = 11.0 MHz		
19.2K	8008H	+ 0.54
9600	8011H	+ 0.54
4800	8023H	+ 0.54
2400	8047H	+ 0.54
1200	808EH	− 0.16
300	823CH	+ 0.01
XTAL1 Frequency = 10.0 MHz		
19.2K	8007H	− 1.70
9600	800FH	− 1.70
4800	8020H	+ 1.38
2400	8040H	− 0.16
1200	8081H	− 0.16
300	8208H	+ 0.03

Figure 2-17. Baud Rate Values for 10, 11, 12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to

receive the message switches to Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communication can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.

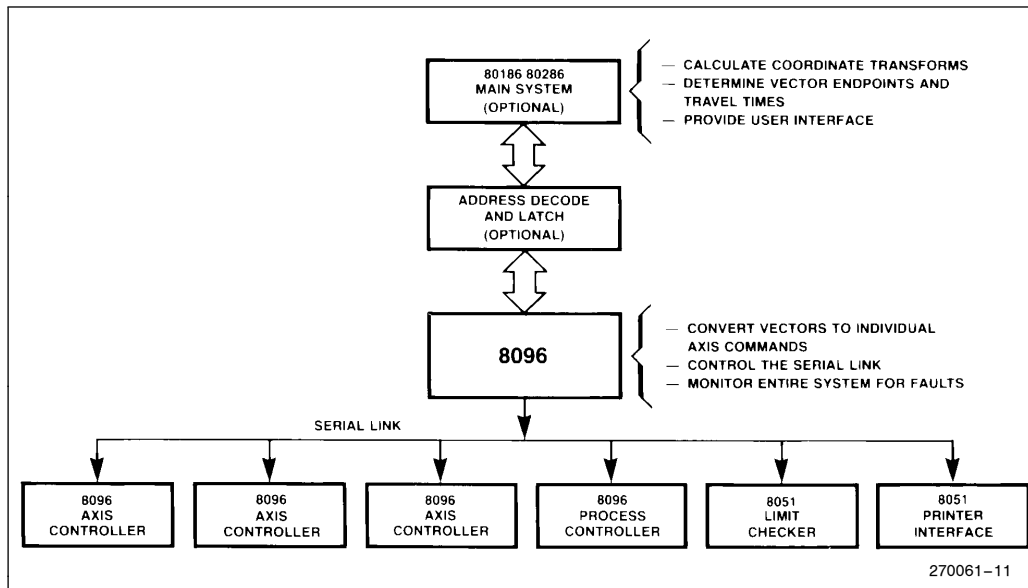


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN = 1. In either case, RI is set again when the received byte is available in SBUF.

2.3.5. A to D CONVERTER

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog

input. When doing process control algorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be directly

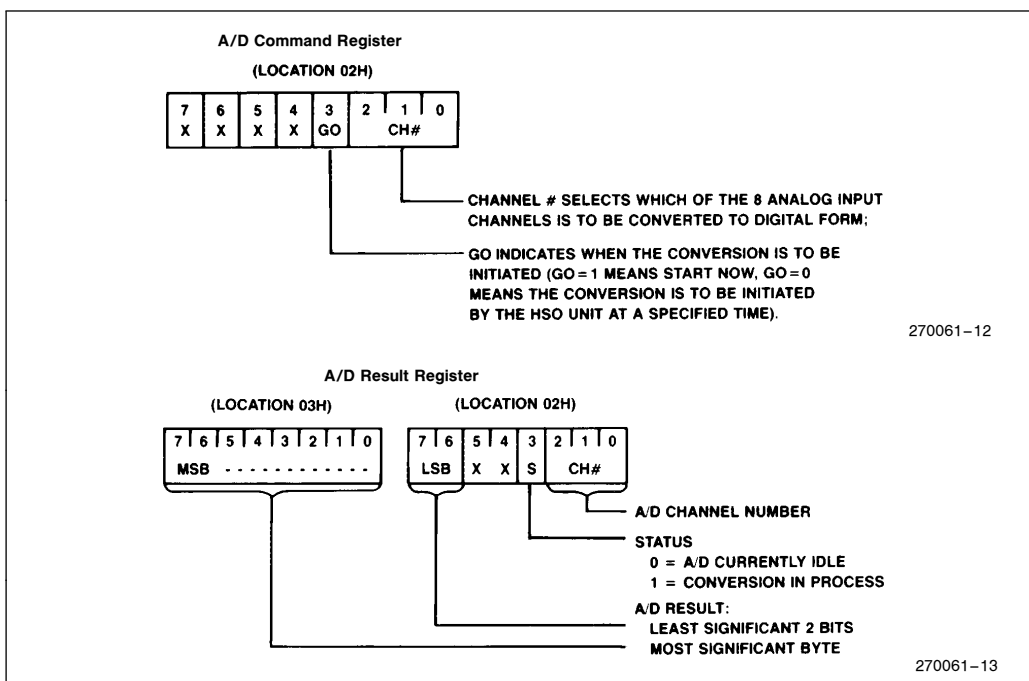


Figure 2-19. A to D Result/Command Register

controlled by software flags which are located in the AD_RESULT/AD_COMMAND Register, shown in Figure 2-19.

2.3.6. PWM REGISTER

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a micro-processor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware, and

takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20.

Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will



be zero when the input is a logical zero. The filter can be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

The RC network determines how quiet the output is, but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

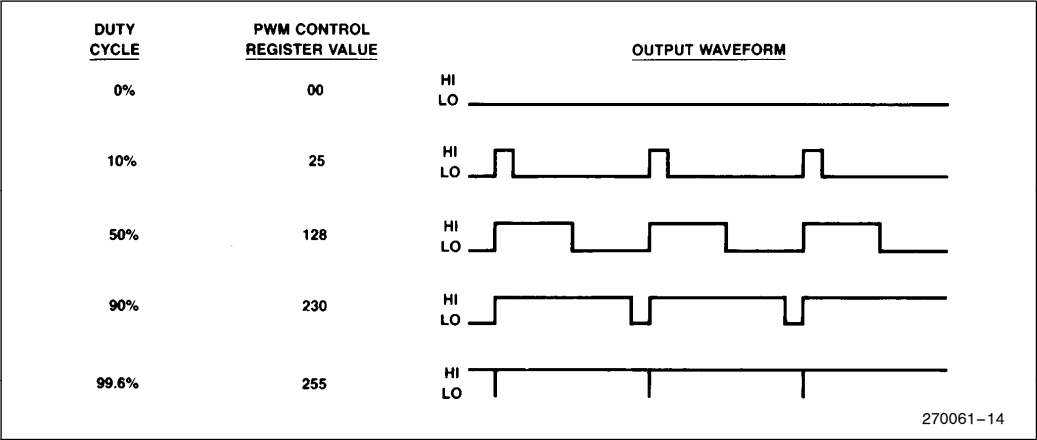


Figure 2-20. PWM Output Waveforms

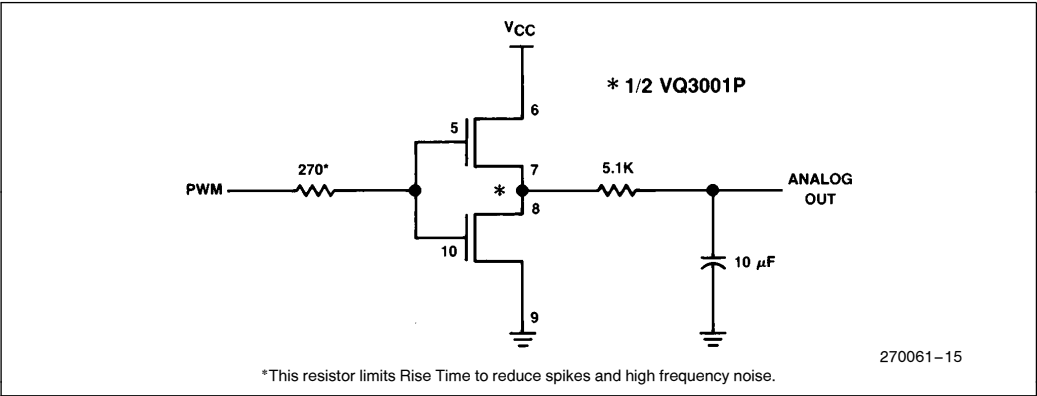


Figure 2-21. PWM to Analog Conversion Circuitry



3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of the lab-

els in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

3.1. Using the 8096's Processing Section

3.1.1. TABLE INTERPOLATION

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

```

; *****
;
; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
;
; *****
;
ZERO EQU 00h:WORD ; R/W
AD_COMMAND EQU 02h:BYTE ; W
AD_RESULT_LO EQU 02h:BYTE ; R
AD_RESULT_HI EQU 03h:BYTE ; R
HSI_MODE EQU 03h:BYTE ; W
HSO_TIME EQU 04h:WORD ; W
HSI_TIME EQU 04h:WORD ; R
HSO_COMMAND EQU 06h:BYTE ; W
HSI_STATUS EQU 06h:BYTE ; R
SBUF EQU 07h:BYTE ; R/W
INT_MASK EQU 08h:BYTE ; R/W
INT_PENDING EQU 09h:BYTE ; R/W
SPCON EQU 11h:BYTE
SPSTAT EQU 11h:BYTE
WATCHDOG EQU 0Ah:BYTE ; W WATCHDOG TIMER
TIMER1 EQU 0Ah:WORD ; R
TIMER2 EQU 0Ch:WORD ; R
PORT0 EQU 0Eh:BYTE ; R
BAUD_REG EQU 0Eh:BYTE ; W
PORT1 EQU 0Fh:BYTE ; R/W
PORT2 EQU 10h:BYTE ; R/W
IOC0 EQU 15h:BYTE ; W
IOS0 EQU 15h:BYTE ; R
IOC1 EQU 16h:BYTE ; W
IOS1 EQU 16h:BYTE ; R
PWM_CONTROL EQU 17h:BYTE ; W
SP EQU 18h:WORD ; R/W STACK POINTER

RSEG at 1CH

AX: DSW 1
DX: DSW 1
BX: DSW 1
CX: DSW 1

AL EQU AX : BYTE
AH EQU (AX+1) : BYTE

```

270061-16

Listing 3-1. Include File DEMO.96.INC



In both methods the procedure is similar. Values of a function are stored in memory for specific input values. To compute the output function for an input that is not listed, a linear approximation is made based on the nearest inputs and nearest outputs. As an example, consider the table below.

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{Upper Output} - \text{Lower Output}}{\text{Upper Input} - \text{Lower Input}} * (\text{Actual Input} - \text{Lower Input})$$

$$\text{Actual Output} = \text{Lower Output} + \text{Delta Out}$$

For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900 - 400}{300 - 200} * (259 - 200) = \frac{500}{100} * 59 = 5 * 59 = 295$$
$$\text{Actual Output} = 400 + 295 = 695$$

To make the algorithm easier, (and therefore faster), it is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 (2*4) was selected. To cover the input range 17 words are needed, 255/16 + 1 word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

Input Value	Relative Table Address	Table Value
100	0001H	100
200	0002H	400
300	0003H	900
400	0004H	1600

```
$TITLE('INTER1.APT: Interpolation routine 1')
;;;;;;;; 8096 Assembly code for table lookup and interpolation

$INCLUDE(:F1:DEMO96.INC)      ; Include demo definitions

RSEG  at 22H

    IN_VAL:      dsb      1          ; Actual Input Value
    TABLE_LOW:  dsb      1
    TABLE_HIGH: dsb      1
    IN_DIF:      dsb      1          ; Upper Input - Lower Input
    IN_DIFB:     equ     IN_DIF      ;byte
    TAB_DIF:     dsb      1          ; Upper Output - Lower Output
    OUT:         dsb      1
    RESULT:      dsb      1
    OUT_DIF:     dsb      1          ; Delta Out

CSEG at 2080H

    LD      SP, #100H
```

270061-17

Listing 3-2. ASM-96 Code for Table Lookup Routine 1



```

look:  LDB     AL, IN_VAL      ; Load temp with Actual Value
       SHRB   AL, #3         ; Divide the byte by 8
       ANDB   AL, #1111110B ; Insure AL is a word address
                               ; This effectively divides AL by 2
                               ; so AL = IN_VAL/16

       LDBZB  AX, AL         ; Load byte AL to word AX
       LD     TABLE_LOW, TABLE [AX] ; TABLE_LOW is loaded with the value
                               ; in the table at table location AX

       LD     TABLE_HIGH, (TABLE+2)[AX] ; TABLE_HIGH is loaded with the
                               ; value in the table at table
                               ; location AX+2
                               ; (The next value in the table)

       SUB    TAB_DIF, TABLE_HIGH, TABLE_LOW ; TAB_DIF=TABLE_HIGH-TABLE_LOW

       ANDB   IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
                               ; of IN_VAL
       LDBZB  IN_DIF, IN_DIFB ; Load byte IN_DIFB to word IN_DIF

       MUL    OUT_DIF, IN_DIF, TAB_DIF ; Output_difference =
                               ; Input_difference*Table_difference
       SHRAL  OUT_DIF, #4         ; Divide by 16 (2**4)

       ADD    OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                               ; generated with truncated IN_VAL
                               ; as input
       SHRA   OUT, #4            ; Round to 12-bit answer

       ADDC   OUT, zero          ; Round up if Carry = 1

no_inc: ST     OUT, RESULT        ; Store OUT to RESULT

       BR     look              ; Branch to "look:"

cseg   AT 2100H

table: DCW     0000H, 2000H, 3400H, 4C00H ; A random function
       DCW     5D00H, 6A00H, 7200H, 7800H
       DCW     7B00H, 7D00H, 7600H, 6D00H
       DCW     5D00H, 4B00H, 3400H, 2200H
       DCW     1000H

END

```

270061-18

Listing 3-2. ASM-96 Code for Table Lookup Routine 1 (Continued)

If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size of the

lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.

```

$TITLE('INTER2.APT: Interpolation routine 2')

;;;;;;;; 8096 Assembly code for table lookup and interpolation
;;;;;;;; Using tabled values in place of division

$INCLUDE(:F1:DEMO96.INC) ; Include demo definitions

RSEG   at 24H

IN_VAL:      dsb      1          ; Actual Input Value
TABLE_LOW:   dsb      1          ; Table value for function
TABLE_INC:   dsb      1          ; Incremental change in function
IN_DIF:      dsb      1          ; Upper Input - Lower Input
IN_DIFB:     equ      IN_DIF :byte
OUT:         dsb      1
RESULT:      dsb      1
OUT_DIF:     dsb      1          ; Delta Out

```

270061-19

Listing 3-3. ASM-96 Code For Table Lookup Routine 2

```

CSEG at 2080H

    LD      SP, #100H      ; Initialize SP to top of reg. file

look:  LDB   AL, IN_VAL     ; Load temp with Actual Value
      SHRB  AL, #3         ; Divide the byte by 8
      ANDB  AL, #1111110B ; Insure AL is a word address
                        ; This effectively divides AL by 2
                        ; so AL = IN_VAL/16
      LDBZ  AX, AL         ; Load byte AL to word AX

      LD     TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                        ; in the value table at location AX

      LD     TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                        ; in the increment table at
                        ; location AX

      ANDB  IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
                        ; of IN_VAL
      LDBZ  IN_DIP, IN_DIFB ; Load byte IN_DIFB to word IN_DIP

      MUL   OUT_DIP, IN_DIP, TABLE_INC
                        ; Output_difference =
                        ; Input_difference*Incremental_change

      ADD   OUT, OUT_DIP, TABLE_LOW ; Add output difference to output
                        ; generated with truncated IN_VAL
                        ; as input
      SHR   OUT, #4         ; Round to 12-bit answer
      ADDC  OUT, zero       ; Round up if Carry = 1

no_inc: ST   OUT, RESULT    ; Store OUT to RESULT
      BR    look           ; Branch to "look:"

cseg  AT 2100H

val_table:
DCW    0000H, 2000H, 3400H, 4C00H ; A random function
DCW    5D00H, 6A00H, 7200H, 7800H
DCW    7B00H, 7D00H, 7600H, 6D00H
DCW    5D00H, 4B00H, 3400H, 2200H
DCW    1000H

inc_table:
DCW    0200H, 0140H, 0180H, 0110H ; Table of incremental
DCW    00D0H, 0080H, 0060H, 0030H ; differences
DCW    00020H, 0FF90H, 0FF70H, 0FF00H
DCW    0FEE0H, 0FE90H, 0FEE0H, 0FEE0H

END

```

270061-20

Listing 3-3. ASM-96 Code for Table Lookup Routine 2 (Continued)

By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/1, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place of many

lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocater and linker. The command line to be used is:

```

RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)

```

```

/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

PLMEX:    DO;

DECLARE IN_VAL      WORD      PUBLIC;
DECLARE TABLE_LOW  INTEGER   PUBLIC;
DECLARE TABLE_HIGH INTEGER   PUBLIC;
DECLARE TABLE_DIF  INTEGER   PUBLIC;
DECLARE OUT          INTEGER   PUBLIC;
DECLARE RESULT      INTEGER   PUBLIC;
DECLARE OUT_DIF      LONGINT   PUBLIC;
DECLARE TEMP        WORD      PUBLIC;

DECLARE TABLE(17)  INTEGER DATA (
    0000H, 2000H, 3400H, 4C00H,      /* A random function */
    5D00H, 6A00H, 7200H, 7800H,
    7B00H, 7D00H, 7600H, 6D00H,
    5D00H, 4B00H, 3400H, 2200H,
    1000H);

DMPY:    PROCEDURE (A,B) LONGINT EXTERNAL;
        DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
    TEMP=SHR(IN_VAL,4);      /* TEMP is the most significant 4 bits of IN_VAL */
    TABLE_LOW=TABLE(TEMP);  /* If "TEMP" was replaced by "SHR(IN_VAL,4)" */
    TABLE_HIGH=TABLE(TEMP+1); /* The code would work but the 8096 would */
                                /* do two shifts */
    TABLE_DIF=TABLE_HIGH-TABLE_LOW;
    OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;
    OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                    in this case 4 places are shifted */
    IF CARRY=0 THEN RESULT=OUT; /* Using the hardware flags must be done */
    ELSE RESULT=OUT+1;          /* with care to ensure the flag is tested */
                                /* in the desired instruction sequence */
GOTO LOOP;

/* END OF PLM-96 CODE */

END;

```

270061-21

Listing 3-4. PLM-96 Code For Table Lookup Routine 1



270061-22

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

3.2. Using the I/O Section

3.2.1. USING THE HSI UNIT

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.), engine control and

```
$TITLE('PULSE.APT: Measuring pulses using the HSI unit')
$INCLUDE(DEMO96.INC)

rseg      at 28H
          HIGH_TIME:      dsw      1
          LOW_TIME:       dsw      1
          PERIOD:         dsw      1
          HI_EDGE:        dsw      1
          LO_EDGE:        dsw      1

cseg      at 2080H

          LD      SP, #100H
          LDB     IOC0, #00000001B      ; Enable HSI 0
          LDB     HSI_MODE, #00001111B   ; HSI 0 look for either edge

wait:     ADD     PERIOD, HIGH_TIME, LOW_TIME
          JBS     IOS1, 6, contin        ; If FIFO is full
          JBC     IOS1, 7, wait          ; Wait while no pulse is entered

contin:   LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                              ; HSI_TIME clears HSI_STATUS

          LD      BX, HSI_TIME           ; Load the HSI_TIME

          JBS     AL, 1, hsi_hi          ; Jump if HSI.0 is high

hsi_lo:   ST      BX, LO_EDGE
          SUB     HIGH_TIME, LO_EDGE, HI_EDGE
          BR      wait

hsi_hi:   ST      BX, HI_EDGE
          SUB     LOW_TIME, HI_EDGE, LO_EDGE
          BR      wait

          END
```

270061-23

Listing 3-6. Measuring Pulses Using The HSI Unit

transmission control. The HSI unit is used extensively in the example in section 4.3.

3.2.2. USING THE HSO UNIT

Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a

HSO line not change so quickly that it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the “MCS-96 Utilities Users Guide”, listed in the bibliography. In order for the program to link, the register dec-

```

$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')

$INCLUDE (DEMO96.INC)

RSEG AT 28H

        TIME:          DSW 1
        LAST_RISE:      DSW 1
        LAST_FALL:      DSW 1
        HSI_S0:         DSB 1
        IOS1_BAK:       DSB 1
        PERIOD:         DSW 1
        LOW_TIME:       DSW 1
        HIGH_TIME:      DSW 1
        COUNT:          DSW 1

cseg    at      2080H

init:   LD      SP,#100H

        LDB      IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                ; Enable PWM,TXD,TIMER1_OVRFLOW_INT

        LDB      HSI_MODE,#10011001B ; set hsi.1 -; hsi.0 +
        LDB      IOC0,#00000111B ; Enable hsi 0,1
                                ; T2 CLOCK=T2CLK, T2RST=T2RST
                                ; Clear timer2

wait:   ANDB     IOS1_BAK,#01111111B ; Clear IOS1_BAK.7
        ORB      IOS1_BAK,IOS1 ; Store into temp to avoid clearing
                                ; other flags which may be needed
        JBC      IOS1_BAK,7,wait ; If hsi is not triggered then
                                ; jump to wait

        ANDB     HSI_S0,HSI_STATUS,#01010101B
        LD        TIME, HSI_TIME

        JBS      HSI_S0,0,a_rise
        JBS      HSI_S0,2,a_fall
        BR        no_cnt

a_rise: SUB      LOW_TIME, TIME, LAST_FALL
        SUB      PERIOD, TIME, LAST_RISE
        LD        LAST_RISE, TIME
        BR        increment

a_fall: SUB      HIGH_TIME, TIME, LAST_RISE
        SUB      PERIOD, TIME, LAST_FALL
        LD        LAST_FALL, TIME

increment:
        INC      COUNT

no_cnt: BR        wait

        END

```

270061-24

Listing 3-7. Enhanced HSI Pulse Measurement Routine

```

$TITLE ('HSOPWM.APT: 8096 EXAMPLE PROGRAM FOR PWM OUTPUTS')

; This program will provide 3 PWM outputs on HSO pins 0-2
; The input parameters passed to the program are:
;
;           HSO_ON_N      HSO on time for pin N
;           HSO_OFF_N     HSO off time for pin N
;
;      Where:  Times are in timer1 cycles
;             N takes values from 0 to 3
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
$INCLUDE(DEMO96.INC)

RSEG AT 28H

      HSO_ON_0:      DSW      1
      HSO_OFF_0:     DSW      1
      HSO_ON_1:      DSW      1
      HSO_OFF_1:     DSW      1
      OLD_STAT:      dsb      1
      NEW_STAT:      dsb      1

cseg      AT 2080H

      LD      SP, #100H
      LD      HSO_ON_0, #100H      ; Set initial values
      LD      HSO_OFF_0, #400H     ; Note that times must be long enough
      LD      HSO_ON_1, #280H     ; to allow the routine to run after each
      LD      HSO_OFF_1, #280H    ; line change.
      ANDB    OLD_STAT, IOS0, #0FH
      XORB    OLD_STAT, #0FH

wait:     JBS      IOS0, 6, wait      ; Loop until HSO holding register
      NOP                                ; is empty

      ; For operation with interrupts 'store_stat:' would be the
      ; entry point of the routine.
      ; Note that a DI or PUSHF might have to be added.

store_stat:
      ANDB    NEW_STAT, IOS0, #0FH    ; Store new status of HSO
      CMPB    OLD_STAT, NEW_STAT
      JE      wait                    ; If status hasn't changed
      XORB    OLD_STAT, NEW_STAT

check_0:
      JBC     OLD_STAT, 0, check_1    ; Jump if OLD_STAT(0)=NEW_STAT(0)
      JBS     NEW_STAT, 0, set_off_0

set_on_0:
      LDB     HSO_COMMAND, #00110000B ; Set HSO for timer1, set pin 0
      ADD     HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
      BR      check_1                ; + Time for pin to be low

set_off_0:
      LDB     HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
      ADD     HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = Timer1 value
      ; + Time for pin to be high

check_1:
      JBC     OLD_STAT, 1, check_done ; Jump if OLD_STAT(1)=NEW_STAT(1)
      JBS     NEW_STAT, 1, set_off_1

set_on_1:
      LDB     HSO_COMMAND, #00110001B ; Set HSO for timer1, set pin 1
      ADD     HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
      BR      check_done

set_off_1:
      LDB     HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
      ADD     HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = Timer1 value
      ; + Time for pin to be high

check_done:
      LDB     OLD_STAT, NEW_STAT      ; Store current status and
      ; wait for interrupt flag

      BR      wait
      ; use RET if "wait" is called from another routine

      END

```

270061-25

Listing 3-8. Generating a PWM with the HSO

laration section (i.e., the section between “RSEG” and “CSEG”) in Listing 3-8 must be changed to that in Listing 3-9.

The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a fre-

quency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.

```
; NOTE: Use this file to replace the declaration section of
; the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
; the line prior to the label "wait". Also change the last
; branch in the program to a "RET".

RSEG

D_STAT:      DSB      1
extrn  HSO_ON_0 :word , HSO_OFF_0 :word
extrn  HSO_ON_1 :word , HSO_OFF_1 :word
extrn  HSO_TIME :word , HSO_COMMAND :byte
extrn  TIMER1  :word , IOS0      :byte
extrn  SP       :word

public OLD_STAT
OLD_STAT:      dsb      1
NEW_STAT:      dsb      1

cseg

PUBLIC wait
```

270061-26

Listing 3-9. Changes to Declarations for HSO Routine

```
$TITLE('HSODRV.APT: Driver module for HSO PWM program')

HSODRV      MODULE MAIN, STACKSIZE(8)

PUBLIC  HSO_ON_0 , HSO_OFF_0
PUBLIC  HSO_ON_1 , HSO_OFF_1
PUBLIC  HSO_TIME , HSO_COMMAND
PUBLIC  SP , TIMER1 , IOS0

$INCLUDE(DEMO96.INC)

rseg at 28H

EXTRN  OLD_STAT      :byte

HSO_ON_0:      dsb      1
HSO_OFF_0:     dsb      1
HSO_ON_1:      dsb      1
HSO_OFF_1:     dsb      1
count:        dsb      1

cseg at 2080H

EXTRN  wait          :entry

strt:  DI
LD     SP, #100H
ANDB   OLD_STAT, IOS0, #0FH
XORB   OLD_STAT, #0FH

initial:
LD     CX, #0100H

loop:  LD     AX, #1000H
SUB    BX, AX, CX
LD     AX, CX

ST     AX, HSO_ON_0
ST     BX, HSO_OFF_0
```

270061-27

Listing 3-10. Driver Module for HSO PWM Program

```

SHR    AX, #1
SHR    BX, #1
ST     AX, HSO_ON_1
ST     BX, HSO_OFF_1

CALL   wait

INC     CX
CMP     CX, #00F00H
BNE     loop

BR      initial

END

```

270061-28

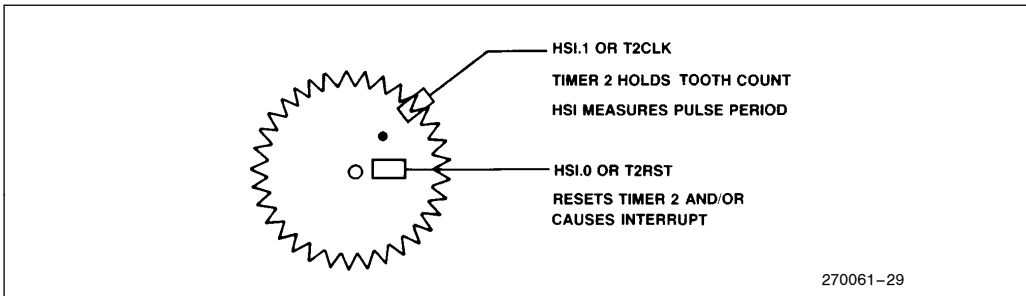
Listing 3-10. Driver Module for HSO PWM Program (Continued)

Since the 8096 needs to keep track of events which often repeat at set intervals it is convenient to be able to have Timer 2 act as a programmable modulo counter. There are several ways of doing this. The first is to program the HSO to reset Timer 2 when Timer 2 equals a set value. A software timer set to interrupt at Timer 2 equals zero could be used to reload the CAM. This software method takes up two locations in the CAM and does not synchronize Timer 2 to the external world.

To synchronize Timer 2 externally the T2 RST (Timer 2 ReSeT) pin can be used. In this way Timer 2 will get reset on each rising edge of T2 RST. If it is desired to have an interrupt generated and time recorded when Timer 2 gets reset, the signal for its reset can be taken from HSI.0 instead of T2RST. The HSI.0 pin has its own interrupt vector which functions independently of the HSI unit.

Another option available is to use the HSI.1 pin to clock Timer 2. By using this approach it is possible to use the HSI to measure the period of events on the input to Timer 2. If both of the HSI pins are used instead of the T2RST and T2CLK pins the HSIO unit can keep track of speed and position of the rotating device with very little software overhead. This type of setup is ideal for a system like the one shown in Figure 3-1, and similar to the one used in section 4.3.

In this system a sequence of events is required based on the position of the gear which represents any piece of rotating machinery. Timer 2 holds the count of the number of tooth edges passed since the index mark. By using HSI.1 as the input to Timer 2, instead of T2 CLK, it is possible to determine tooth count and time information through the HSI. From this information instantaneous velocity and acceleration can be calculated. Having the tooth edge count in Timer 2 means



270061-29

Figure 3-1. Using the HSIO to Monitor Rotating Machinery

that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

3.2.3. USING THE SERIAL PORT IN MODE 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then

transmits the same character. The code is set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags as initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Inter-processor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to ± 12 or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.

```
$TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')
$INCLUDE(DEMO96.INC)

rseg    at 28H
        CHR:    ddb    1
        SPTEMP: ddb    1
        TEMP0:  ddb    1
        TEMP1:  ddb    1
        RCV_FLAG: ddb    1

cseg    at 200CH
        DCW     ser_port_int

cseg    at 2080H
        LD      SP, #100H
        LDB     IOC1, #00100000B          ; Set P2.0 to TXD
        ; Baud rate = input frequency / (64*baud_val)
        ; baud_val = (input frequency/64) / baud rate

baud_val    equ    39          ; 39 = (12,000,000/64)/4800 baud
BAUD_HIGH   equ    ((baud_val-1)/256) OR 80H      ; Set MSB to 1
BAUD_LOW    equ    (baud_val-1) MOD 256

        LDB     BAUD_REG, #BAUD_LOW
        LDB     BAUD_REG, #BAUD_HIGH
        LDB     SPCON, #01001001B          ; Enable receiver, Mode 1
        ; The serial port is now initialized

        STB     SBUF, CHR                ; Clear serial Port
        LDB     TEMP0, #00100000B        ; Set TI-temp
        LDB     INT_MASK, #01000000B     ; Enable Serial Port Interrupt
        EI
loop:      BR     loop                    ; Wait for serial port interrupt

ser_port_int:
        PUSHF
rd_again:
        LDB     SPTEMP, SPSTAT            ; This section of code can be replaced
        ORB     TEMP0, SPSTAT            ; with "ORB TEMP0, SP_STAT" when the
        ANDB     SPTEMP, #01100000B      ; serial port TI and RI bugs are fixed
        JNE     rd_again                  ; Repeat until TI and RI are properly cleared
```

270061-30

Listing 3-11. Using the Serial Port in Mode 1

```

get_byte:
    JBC      TEMP0, 6, put_byte      ; If RI-temp is not set
    STB      SBUF, CHR              ; Store byte
    ANDB     TEMP0, #10111111B      ; CLR RI-temp
    LDB      RCV_FLAG, #0FFH        ; Set bit-received flag

put_byte:
    JBC      RCV_FLAG, 0, continue   ; If receive flag is cleared
    JBC      TEMP0, 5, continue      ; If TI was not set
    LDB      SBUF, CHR              ; Send byte
    ANDB     TEMP0, #11011111B      ; CLR TI-temp

    ANDB     CHR, #01111111B         ; This section of code appends
    CMPB     CHR, #0DH              ; an LF after a CR is sent
    JNE      clr_rcv
    LDB      CHR, #0AH
    BR       continue

clr_rcv:
    CLRB     RCV_FLAG               ; Clear bit-received flag

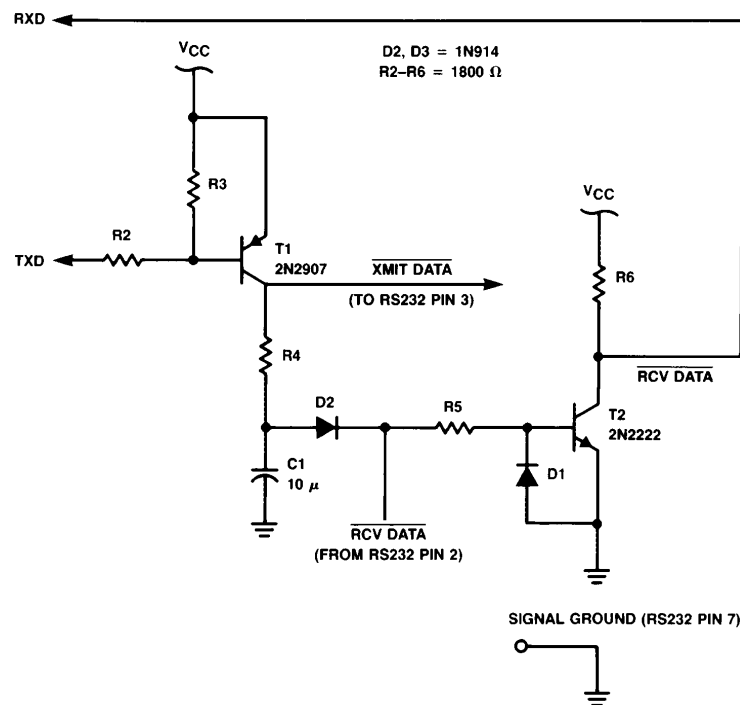
continue:
    POPF
    RET

    END

```

270061-31

Listing 3-11. Using the Serial Port in Mode 1 (Continued)



270061-32

Figure 3-2. Serial Port Level Conversion

3.2.4. USING THE A TO D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1. When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100Ω resistor in series with a 0.22 uF capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section does not

really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

4.1. Simultaneous I/O Routines under Interrupt Control

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and in which

```
$TITLE('ATOD.APT: SCANNING THE A TO D CHANNELS')
$INCLUDE(DEMO96.INC)

RSEG      at      28H

          BL      EQU      BX:BYTE
          DL      EQU      DX:BYTE

RESULT_TABLE:
RESULT_1:      dsw      1
RESULT_2:      dsw      1
RESULT_3:      dsw      1
RESULT_4:      dsw      1

cseg      at      2080H

start:    LD      SP, #100H      ; Set Stack Pointer
          CLR     BX

next:     ADDB    AD_COMMAND,BL, #1000B      ; Start conversion on channel
          ; indicated by BL register

          NOP     ; Wait for conversion to start
          NOP
check:    JBS     AD_RESULT_LO, 3, check      ; Wait while A to D is busy

          LDB     AL, AD_RESULT_LO      ; Load low order result
          LDB     AH, AD_RESULT_HI      ; Load high order result

          ADDB    DL, BL, BL      ; DL=BL*2
          LDBZ    DX, DL
          ST      AX, RESULT_TABLE[DX]      ; Store result indexed by BL*2

          INCB    BL      ; Increment BL modulo 4
          ANDB    BL, #03H

          BR      next

          END
```

270061-33

Listing 3-12. Scanning the A to D Channels

order, Port 1 output pins are used to indicate the current status of each task. The actual code listing is included in Appendix B.

The initialization section, shown in Listing 4-1a, clears a few variables and then loads the first set of on and off times to the HSO unit. Note that 8 state times must

be waited between consecutive loads of the HSO. If this is not done it is possible to overwrite the contents of the CAM holding register. An A/D interrupt is forced by setting the bit in the Interrupt Pending register. This causes the first A/D interrupt to occur just after the Interrupt Mask register is set and interrupts are enabled.

Listing 4-1. Using Multiple I/O Devices

```

$TITLE ('8096 EXAMPLE PROGRAM FOR PWM OUTPUTS FROM A TO D INPUTS')
$PAGEWIDTH(130)

; This program will provide 3 PWM outputs on HSO pins 0-2
; and one on the PWM.
;
; The PWM values are determined by the input to the A/D converter.
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

$INCLUDE (DEMO96.INC)

RSEG AT 28H

      DL      EQU      DX:BYTE

ON_TIME:
      PWM_TIME_1:      DSW      1
      HSO_ON_0:        DSW      1
      HSO_ON_1:        DSW      1
      HSO_ON_2:        DSW      1

RESULT_TABLE:
      RESULT_0:        DSW      1
      RESULT_1:        DSW      1
      RESULT_2:        DSW      1
      RESULT_3:        DSW      1

      NXT_ON_T:        DSW      1
      NXT_OFF_0:       DSW      1
      NXT_OFF_1:       DSW      1
      NXT_OFF_2:       DSW      1
      COUNT:          DSW      1
      AD_NUM:          DSW      1          ; Channel being converted
      TMP:             DSW      1
      HSO_PER:         DSW      1
      LAST_LOAD:       DSW      1

cseg   AT 2000H

      DCW      start      ; Timer_ovf_int
      DCW      Atod_done_int
      DCW      start      ; HSI_data_int
      DCW      HSO_exec_int

cseg   AT 2080H

start: LD      SP, #100H      ; Set Stack Pointer
      CLR     AX
wait:  DEC     AX              ; wait approx. 0.2 seconds for
      JNE     wait           ; SBE to finish communications

      CLRB    AD_NUM

      LD      PWM_TIME_1, #080H
      LD      HSO_PER, #100H
      LD      HSO_ON_0, #040H
      LD      HSO_ON_1, #080H
      LD      HSO_ON_2, #0C0H

      ADD     NXT_ON_T, Timer1, #100H

```

270061-34

Listing 4-1a. Initializing the A to D to PWM Program

270061-35

Listing 4-1a. Initializing the A to D to PWM program (Continued)

270061-36

Listing 4-1b. Interrupt Driven HSO Routine

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;; A TO D COMPLETE INTERRUPT ;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ATOD_done_int:
    PUSHF
    ORB     Port1, #00000100B      ; Set P1.2

    ANDB    AL, AD_RESULT_LO, #11000000B ; Load low order result
    LDB     AH, AD_RESULT_HI      ; Load high order result
    ADB     DL, AD_NUM, AD_NUM    ; DL= AD_NUM *2
    LDBZ    DX, DL
    ST      AX, RESULT_TABLE[DX] ; Store result indexed by DX

    CMPB    AL, #01000000B
    JNH     no_rnd                ; Round up if needed
    CMPB    AH, #0FFH            ; Don't increment if AH=0FFH
    JE      no_rnd
    INCB    AH

no_rnd:    LDB     AL, AH          ; Align byte and change to word
    CLRB    AH
    ST      AX, ON_TIME[DX]

    INCB    AD_NUM
    ANDB    AD_NUM, #03H        ; Keep AD_NUM between 0 and 3

next:     ADB     AD_COMMAND, AD_NUM, #1000B ; Start conversion on channel
                                                ; indicated by AD_NUM register
    ANDB    Port1, #11111011B    ; Clear P1.2
    POPF
    RET

END

```

270061-37

Listing 4-1c. Interrupt Driven A to D Routine

The HSO routine shown in Listing 4-1b is slightly different than the one in section 3. All of the HSO lines turn on at the same time, only the turn-off-time is varied between lines. This action is what is most commonly required for multiple PWM outputs and simplifies the software. A comparison is made between Timer1 and the next HSO turn on time at the beginning of the routine. If the next turn on time has passed, then the on-times are loaded into the CAM, otherwise the off times are loaded.

The maximum number of events in the CAM at any given time is 7. This occurs when the first line to turn off does so, causing the off-times for all of the lines to be loaded. For two of the lines there will be an off-time, an on-time, and the just loaded off-time. The other line (the one that just turned off) will have only the on-time and the just loaded off-time.

A/D conversions are performed by the code in Listing 4-1c about every 60 microseconds, 42 for the conversion, the rest for overhead. The A/D routine sets up the HSO and PWM on and off times. Since the A/D

has a ten bit output, the most significant 8 bits are rounded up or down based on the least significant two bits.

4.2. Software Serial Port Using the HSIO Unit

There are many systems which require more than one serial port, an example is a system which must communicate with other computers and have an additional port for a local console. If the on-board UART is being used as an inter-processor link, the HSIO unit can be used to interface the 8096 to an additional asynchronous line.

Figure 4-1 shows the format of a standard 10-bit asynchronous frame. The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the START bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are the eight data bits which are transmitted least significant bit first. The STOP bit is set to the opposite state of the START bit to guar-

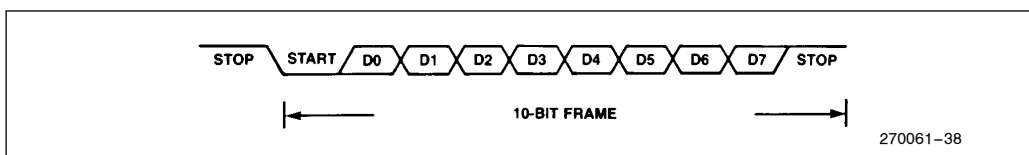


Figure 4-1. 10-bit Asynchronous Frame

antee that the leading edge of the START bit will cause a transition on the line; it also provides for a dead time on the line so that the receiver can maintain its synchronization.

The remainder of this section will show how a full-duplex asynchronous port can be built from the HSO unit. There are four sections to this code:

1. Interface routines. These routines provide a procedural interface between the interrupt driven core of the software serial port and the remainder of the application software.
2. Initialization routine. This routine is called during the initialization of the overall system and sets up the various variables used by the software port.

3. Transmit ISR. This routine runs as an ISR (interrupt service routine) in response to an HSO interrupt interrupt. Its function is to serialize the data passed to it by the interface routines.

4. Receive ISRs. There are two ISRs involved in the receive process. One of them runs in response to an HSI interrupt and is used to synchronize the receive process at the leading edge of the start bit. The second receive ISR runs in response to an HSO generated software timer interrupt, this routine is scheduled to run at the center of each bit and is used to deserialize the incoming data.

The routines share the set of variables that are shown in Listing 4-2. These variables should be accessed only by the routines which make up the software serial port.

```

;
;   VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
;   =====
;
;   rseg
;
rcve_state:    dsb 1
rxTdy         equ 1           ; indicates receive done
rxoverrun     equ 2           ; indicates receive overflow
rip           equ 4           ; receive in progress flag
rcve_buf:     dsb 1           ; used to double buffer receive data
rcve_reg:     dsb 1           ; used to deserialize receive
sample_time:  dsw 1           ; records last receive sample time

serial_out:   dsw 1           ; Holds the output character+framing (start and
                             ; stop bits) for transmit process.
baud_count:   dsw 1           ; Holds the period of one bit in units
                             ; of T1 ticks.
txd_time:     dsw 1           ; Transition time of last Txd bit that was
                             ; sent to the CAM
char:         dsb 1           ; for test only
;
;   COMMANDS ISSUED TO THE HSO UNIT
;   =====
;
mark_command   equ 0110101b    ; timer1,set,interrupt on 5
space_command  equ 0010101b    ; timer1,clr,interrupt on 5
sample_command equ 0011000b    ; software timer 0

$eject

```

270061-39

Listing 4-2. Software Serial Port Declarations

The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling *char_out*, which waits for any in-process transmission to complete and stores the character into the variable *serial_out*. As the data is

stored the START and STOP bits are added to the data bits. The routine *char_in* is called when the application software requires a character from the port. The data is returned in the *ax* register in conformance to PLM 96 calling conventions. The routine *cts* can be called to determine if a character is available at the port before calling *char_in*. (If no character is available *char_in* will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

```

; char_out:
; Output character to the software serial port
;
    pop     cx          ; the return address
    pop     bx          ; the character for output
    ldb     (bx+1),%01h ; add the start and stop bits
    add     bx,bx        ; to the char and leave as 16 bit
wait_for_xmit:
    cmp     serial_out,0 ; wait for serial_out=0 (it will be cleared by
    bne     wait_for_xmit ; the hso interrupt process)
    st      bx,serial_out ; put the formatted character in serial_out
    br      [cx]         ; return to caller

;
cts:
; Returns "true" (ax<>0) if char_in has a character.
;
    clr     ax
    bbc     rcve_state,0,cts_exit
    inc     ax
cts_exit:
    ret

; char_in:
; Get a character from the software serial port
;
    bbc     rcve_state,0,char_in ; wait for character ready
    pushf
    andb    rcve_state,%not(rxrdy) ; set up a critical region
    ldbz    al,rcve_buf
    popf
    ret

```

270061-40

Listing 4-3. Software Serial Port Interface Routines

```

;
; setup_serial_port:
; Called on system reset to initiate the software serial port.
;
    pop     cx          ; the return address
    pop     bx          ; the baud rate (in decimal)
    ld      dx,%0007h   ; dx:ax:=500,000 (assumes 12 Mhz crystal)
    ld      ax,%0A120h
    divu    ax,bx        ; calculate the baud count (500,000/baudrate)
    st      ax,baud_count
    st      0,serial_out ; clear serial out
    ldb     ioc1,%01I00000b ; Enable HSO.5 and Txd
    bbs     ios0,6,$     ; Wait for room in the HSO CAM
                    ; and issue a MARK command.
    add     txd_time,timer1,20
    ldb     hso_command,%mark_command
    ld      hso_time,txd_time
    clrb    rcve_buf     ; clear out the receive variables
    clrb    rcve_reg
    clrb    rcve_state
    call    init_receive ; setup to detect a start bit
    br      [cx]         ; return

```

270061-41

Listing 4-4. Software Serial Port Initialization Routine

stack; it calculates the bit time from the baud rate and stores it in the variable *baud_count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init_receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial_out* register is zero a MARK (idle condition) is output. If the *serial_out* register contains data then the least sig-

nificant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi_isr* and *software_timer_isr*. The listings of these routines are shown in Listings 4-6a, 4-6b, and 4-6c respectively. The

```

;
hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso service strategy for an
; actual system.

        cseg      at 2006h
        dcw       hso_isr          ; Set up vector

        cseg
        pushf
        add       txd_time,baud_count
        cmp       serial_out,0     ; if character is done send a mark
        be        send_mark
        shr       serial_out,#1    ; else send bit 0 of serial_out and shift
        bc        send_mark        ; serial_out left one place.

send_space:
        ldb       hso_command,#space_command
        ld        hso_time,txd_time
        br        hso_isr_exit

send_mark:
        ldb       hso_command,#mark_command
        ld        hso_time,txd_time

hso_isr_exit:
        popf
        ret

$reject

```

270061-42

Listing 4-5. Software Serial Port Transmit Process

Listing 4-6. Receive Process

```

;
init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.
;
        ldb       ioc0,#00000000b    ; disconnect change detector
        ldb       hsi_mode,#00100000b ; negative edges on HSI.2

flush_fifo:
        orb       ioc1_save,ioc1
        bbc       ioc1_save,7,flush_fifo_done
        ldb       al,hsi_status
        ld        ax,hsi_time        ; trash the fifo entry
        andb      ioc1_save,#not(80h) ; clear bit 7.
        br        flush_fifo

flush_fifo_done:
        ldb       ioc0,#00010000b    ; connect HSI.2 to detector
        ret

```

270061-43

Listing 4-6a. Software Serial Port Receive Initialization

```

;
; hsi_isr:
; Fields interrupts from the HSI unit, used to detect the leading edge
; of the START bit
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
;
; cseg at 2004h
; dcw hsi_isr ; setup the interrupt vector
;
; cseg
; pushf
; push ax
; ldb al,hsi_status
; ld sample_time,hsi_time
; bbc al,4,exit_hsi
; bbs ios0,7,$ ; wait for room in HSO holding reg
; ld ax,baud_count ; send out sample command in 1/2
; shr ax,#1 ; bit time
; add sample_time,ax
; ldb hso_command,#sample_command
; st sample_time,hso_time
; ldb ios0,#00000000b ; disconnect hsi.2 from change detector
;
; exit_hsi:
; pop ax
; popf
; ret

```

270061-44

Listing 4-6b. Software Serial Port Start Bit Detect

```

;
; software_timer_isr:
; Fields the software timer interrupt, used to deserialize the incoming data.
; Note: this routine would be incorporated into the software timer strategy
; in an actual system.
;
; cseg at 200ah
; dcw software_timer_isr ; setup vector
;
; cseg
; pushf
; orb ios1_save,ios1
; andb ios1_save,#not(01h) ; clear bit 0
; andb 0,rcve_state,#0fch ; All bits except rxrdy and overrun=0
; bne process_data
;
; process_start_bit:
; bbc hsi_status,5,start_ok
; call init_receive
; br software_timer_exit
;
; start_ok:
; orb rcve_state,#rip ; set receive in progress flag
; br schedule_sample
;
; process_data:
; bbs rcve_state,7,check_stopbit
; shr rcve_reg,#1
; bbc hsi_status,5,datazero
; orb rcve_reg,#80h ; set the new data bit
;
; datazero:
; addb rcve_state,#10h ; increment bit count
; br schedule_sample
;
; check_stopbit:
; bbc hsi_status,5,$ ; DEBUG ONLY
; ldb rcve_buf,rcve_reg
; orb rcve_state,#rxrdy
; andb rcve_state,#03h ; Clear all but ready and overrun bits
; call init_receive
; br software_timer_exit
;
; schedule_sample:
; bbs ios0,7,$ ; wait for holding reg empty
; ldb hso_command,#sample_command
; add sample_time,baud_count
; st sample_time,hso_time
;
; software_timer_exit:
; popf
; ret

```

270061-45

Listing 4-6c. Software Serial Port Data Reception

start is detected by the *hsi_isr* which schedules a software timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable *rcve_state* to determine whether it should check for a valid START bit, deserialize data, or check for a valid STOP bit. When a complete character has been received it is moved to the receive buffer and *init_receive* is called to set up the receive process for the next character. This routine is also called when an error (e.g., invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

4.3. Interfacing an Optical Encoder to the HSI Unit

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

Frequently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pitmann has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.

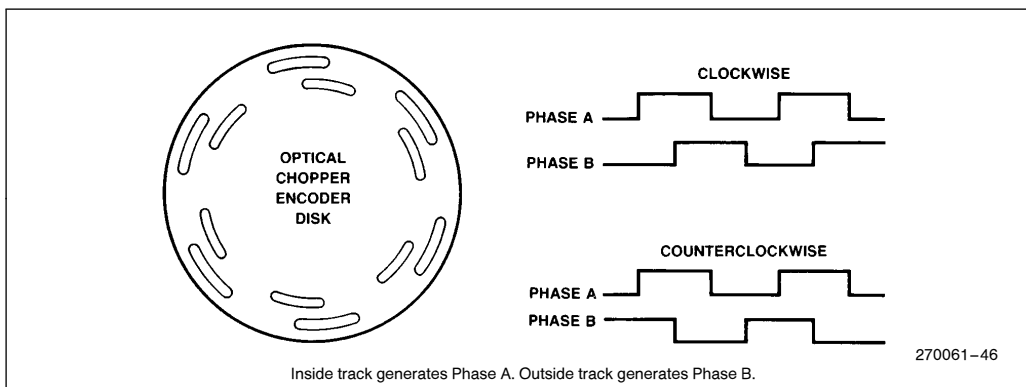


Figure 4-2. Optical Encoder and Waveforms



Digital filtering external to the 8096 is used on the encoder signals. The idealized signals coming from the encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

If it is desired to determine when each edge occurs before filtering, the encoder outputs can be attached directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the ± 15 volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control with

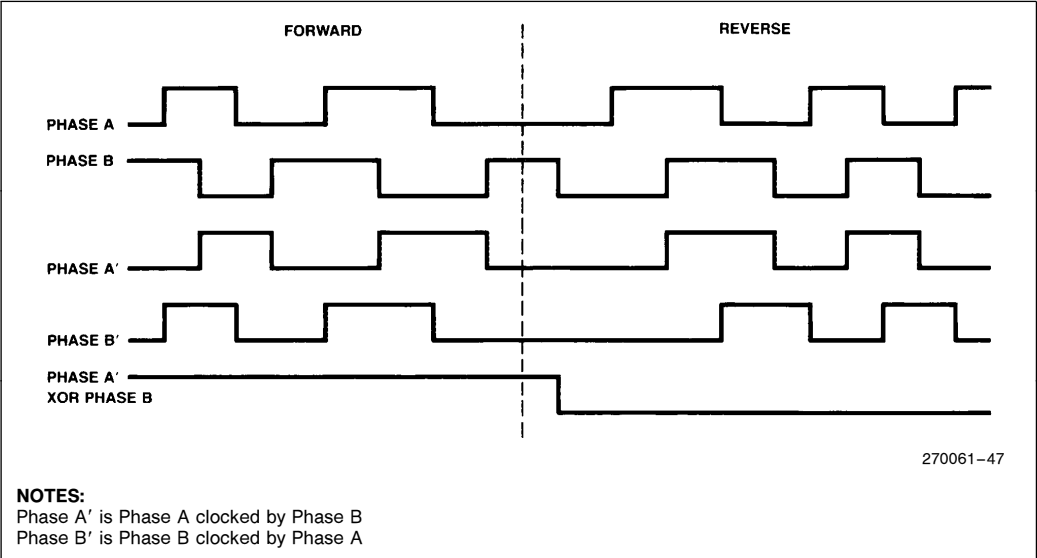


Figure 4-3. Filtered Encoder Waveforms



some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found that the

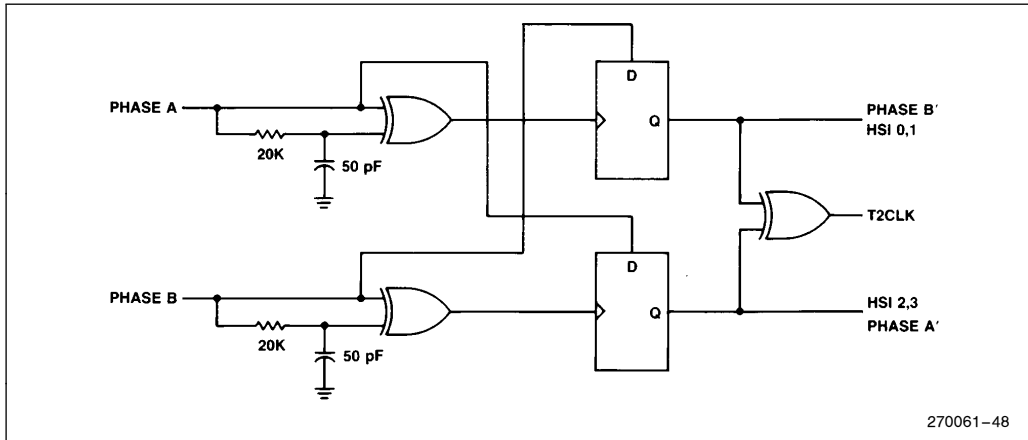


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

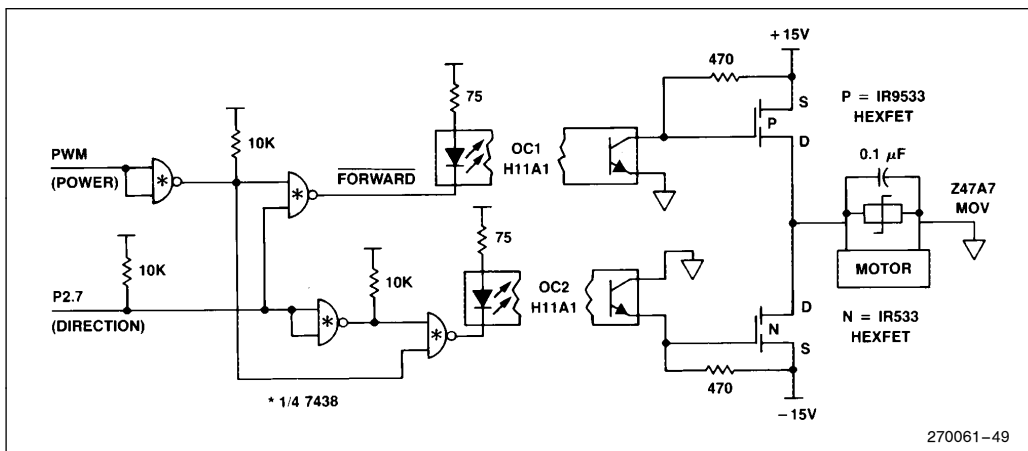


Figure 4-5. Motor Driver Circuitry

motor would decelerate smoothly until the time between encoder edges was around 100 microseconds. At this point the motor would either continue to decelerate slowly, or would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, I/O port 1 is used to indicate the program status. This information consists of which routine the program is in and under which mode it is operating. The main program sections are: Main loop, HSI interrupt, Timer 2 check, and Motor drive. There are also minor sections such as initialization, timer overflow handling, and software timer handling. Tying everything together is some overhead and glue. Where the glue is not obvious it will be discussed, otherwise it can be derived from the listings.

The program is a main loop which does nothing except serve as a place for the program to go when none of the interrupt routines are being run. All of the processing is done on an interrupt basis.

There are three basic software modes which are invoked depending on the speed of the motor. The modes referred to as 0, 1 and 2, in order from slowest to fastest operation. When the program is running the operating

mode is indicated by the lower 2 bits of Port 1, with the following coding:

P1.0	P1.1	Mode	Description
0	0	0	HSI looks at every edge
1	0	1	HSI looks at Phase A edges only
0	1	2	Timer 2 used instead of HSI
1	1	2	(alternate form of above)

The example is easiest to see if mode 2 is described first, followed by mode 1 then mode 0. In mode 2 Timer 2 is used to count edges on the incoming signal. A software timer routine, which is actually run using HSO.0, uses the Timer 2 value to update a LONG (32-bit) software counter labeled *POSITION*. The HSO routine runs every 260 microseconds. The HSO.0 interrupt is used instead of an actual software timer because of the ability to easily unmask it while other software timer routines are running.

In the code in Listing 4-7, the mode is first determined. For the first pass ignore the code starting with the label *in_mode_1*. Starting with *in_mode_2* the counter is incremented or decremented based on bit zero of *DIRECT*. If *DIRECT.0* = 0 the motor is going backward, if it is a 1 the motor is going forward. Next the count difference is checked to see if it is slow enough to go into mode 1. If not the routine returns to the code it was running when the interrupt occurred.

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!                                SOFTWARE TIMER ROUTINE 0                                !!!!!!!!!!!!!!!
!!!!!!                                NOW USING HSO.0 TO TRIGGER                                !!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        CSEG AT 2280H

hso_exec_int:                                ; Check mode - Update position in mode 2

        PUSHF
        ldb     HSO_COMMAND,#30H
        add     HSO_TIME,TIMER1,HSO0_dly

        orb     port1,#00100000B             ; set P1.5
        ld      Timer_2,TIMER2
        jbs     Port1,1,in_mode2

in_mode1:
        sub     tmp1,Timer_2,old_t2          ; Check count difference in tmp1
        cmp     tmp1,#2
        jh      end_swt0
set_mode0:
        jbc     Port1,0,end_swt0              ; if already in mode 0
        andb    Port1,#11111100B             ; Clear P1.0, P1.1 (set mode 0)
        ldb     IOC0,#01010101B             ; enable all HSI
        ldb     last_stat,zero
        br      end_swt0

```

270061-50

Listing 4-7. Motor Control HSO.0 Timer Routine

```

in_mode2:
    sub    delta_p,timer_2,tmr2_old      ; get timer2 count difference
    ld     tmr2_old,timer_2
    jbc    direct,0,in_rev

in_fwd:   add    position,delta_p
    addc   position+2,zero
    br     chk_mode

in_rev:   sub    position,delta_p
    subc   position+2,zero

chk_mode:
    sub    tmp1,timer_2,old_t2          ; Check count difference in tmp1
    cmp    tmp1,#5                     ; set model if count is too low
    jgt    end_swt0                    ; count <= 5

set_model:
    andb   Port1,#11111101B           ; Clear Pl.1, set Pl.0 (set mode 1)
    orb    Port1,#00000001B
    ldb    IOC0,#00000101B            ; enable HSI 0 and 1
    ld     zero, HSI_TIME
    sub    last1_time,timer1,min_hsil
    ; set up so (time-last2_time)>min_hsil on next HSI

clr_hsi:
    ld     ZERO, HSI_TIME
    andb   ios1_bak,#01111111B         ; clear bit 7
    orb    ios1_bak,ios1
    jbs    ios1_bak,7,clr_hsi          ; If hsi is triggered then clear hsi

end_swt0:
    ld     old_t2,TIMER_2
    andb   port1,#11011111B            ; clear Pl.5
    POPF
    ret

```

270061-51

Listing 4-7. Motor Control HSO.0 Timer Routine (Continued)

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI_TIME* register is read to speed up clearing the HSI FIFO and the *LAST1_TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI FIFO is then cleared, the Timer 2 value used throughout this routine is saved, and the routine returns.

This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in_mode1* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds. The primary method, (under the HSI routine), can only

enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI FIFO data is valid, and then the FIFO is read. Next, the main body of code (for either mode 0 or mode 1) is

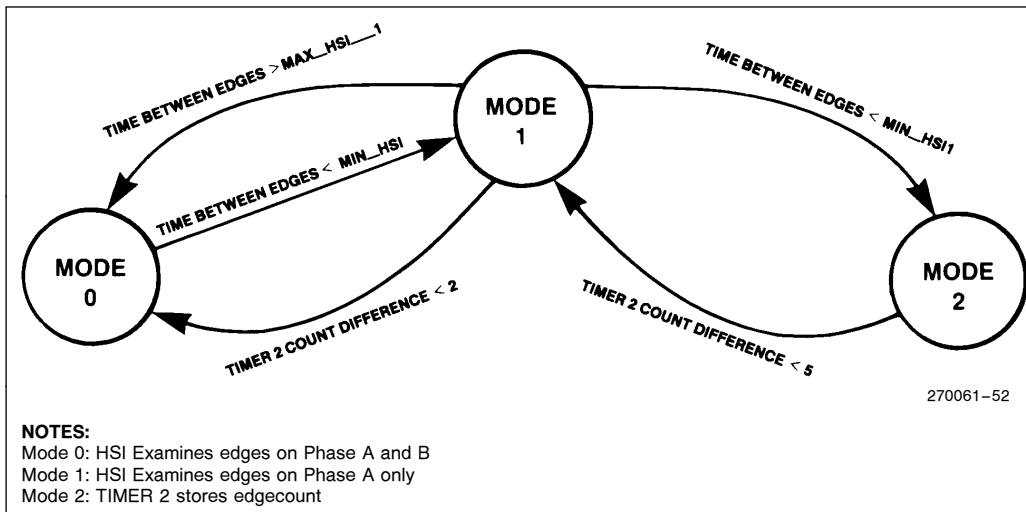


Figure 4-6. Mode State Diagram

```

; ; ; ; ; HSI DATA AVAILABLE INTERRUPT ROUTINE ; ; ; ; ;
; ; ; ; ;
; This routine keeps track of the current time and position of the motor.
; The upper word of information is provided by the timer overflow routine.

CSEG AT 2400H

now_mode_1: br      in_mode_1          ; used to save execution time for
no_intl:    br      no_int             ; worst case loop


hsi_data_int: pushf
              orb     portl,#01000000B   ; set P1.6
              andb    iosl_bak,#01111111B ; Clear iosl_bak.7
              orb     iosl_bak,iosl
              jbc     iosl_bak,7,no_intl   ; If hsi is not triggered then
                                           ; jump to no_int

get_values:  ld       timer_2,TIMER2
              andb    hsi_s0,HSI_STATUS,#01010101B
              ld       time,HSI_TIME

              jbs     portl,0,now_mode_1   ; jump if in mode 1

In_mode_0:

; ; ; ; ;
; ; ; ; ; INSERT BODY OF ROUTINE ; ; ; ; ;
; ; ; ; ;

load_last:  ld       tmr2_old,timer_2
no_cnt:    andb    iosl_bak,#01111111B   ; clr bit 7
           orb     iosl_bak,iosl
           jbc     iosl_bak,7,no_int
again:     br      get_values

no_int:    andb    portl,#10111111B      ; Clear P1.6
           popf
           ret
           ; end of hsi_data interrupt routine
           ; Routine for mode 1 follows and then returns to "load_last"

$EJECT
```

Listing 4-8. Motor Control HSI Data Available Routine

run. At the end time and count values are saved and the holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST_STAT* is cleared. *LAST_STAT* = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update

POSITION. The program then returns to the completion section of the routine.

There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1_TIME* and *LAST2_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.

```

In_mode_1:                ; mode 1 HSI routine

    andb    tmpl,hsi_s0,#01010000B
    jne     no_cnt

cmp_time:
    ld      last2_time,last1_time
    ld      last1_time,time

    cmpl:   sub    tmpl,time,last2_time
    cmp     tmpl,min_hsil
    jh      check_max_time

set_mode_2:
    orb     Port1,#00000010B      ; Set Pl.1 (in mode 2)
    ldb     IOC0,#00000000B      ; Disable all HSI
    mt_hsi: ld      zero,hsi_time ; empty the hsi fifo
    andb    los1_bak,#01111111B  ; clear bit 7
    orb     los1_bak,los1
    jbs     los1_bak,7,mt_hsi     ; If hsi is triggered then clear hsi
    br      done_chk

check_max_time:
    sub     tmpl,time,last2_time
    cmp     tmpl,max_hsil        ; max_hsil = addition to min_hsil for
    jnh     done_chk             ; total time

set_mode_0:
    andb    Port1,#11111100B     ; clear Pl.0,1 set mode 0)
    ldb     IOC0,#01010101B     ; Enable all HSI
    ldb     last_stat,zero

done_chk:
    sub     delta_p,timer_2,tmr2_old ; get timer2 count difference
    jbc     direct,0,add_rev

add_fwd:
    add     position,delta_p
    addc    position+2,zero
    br      load_last

add_rev:
    sub     position,delta_p
    subc    position+2,zero
    br      load_last

$reject

```

270061-54

Listing 4-9. Motor Control Mode 1 Routines

```

In_mode_0:
    jbs     hsi_s0,0,a_rise
    jbs     hsi_s0,2,a_fall
    jbs     hsi_s0,4,b_rise
    jbs     hsi_s0,6,b_fall
    br      no_cnt

a_rise: ld     last2_time,last1_time
        ld     last1_time,time
        sub    time,last2_time
        cmp    time,min_hsi
        jh     tst_statr
;set model-
        orb    Port1,#00000001B      ; Set P1.0 (in mode 1)
        ldb    IOC0,#00000101B      ; Enable HSI 0 and 1
tst_statr:
    jbs     last_stat,6,going_fwd
    jbs     last_stat,4,going_rev
    jbs     last_stat,2,change_dir
    cmpb    last_stat,zero
    je      first_time                ; first time in mode0
    br      inp_err

a_fall: ld     last2_time,last1_time
        ld     last1_time,time
        sub    time,last2_time
        cmp    time,min_hsi
        jh     tst_statf
;set model-
        orb    Port1,#00000001B      ; Set P1.0 (in mode 1)
        ldb    IOC0,#00000101B      ; Enable HSI 0 and 1
tst_statf:
    jbs     last_stat,4,going_fwd
    jbs     last_stat,6,going_rev
    jbs     last_stat,0,change_dir
    cmpb    last_stat,zero
    je      first_time                ; first time in mode0
    br      inp_err

b_rise: jbs     last_stat,0,going_fwd
        jbs     last_stat,2,going_rev
        jbs     last_stat,6,change_dir
        cmpb    last_stat,zero
        je      first_time                ; first time in mode0
        br      inp_err

b_fall: jbs     last_stat,2,going_fwd
        jbs     last_stat,0,going_rev
        jbs     last_stat,4,change_dir
        cmpb    last_stat,zero
        je      first_time                ; first time in mode0
        br      inp_err

first_time:
    stb     hsi_s0,last_stat
    br      done_chk                ; add delta position
inp_err:
    br      no_int

change_dir:
    notb
no_inc: jbc    direct,0,going_rev

going_fwd:
    orb     PORT2,#01000000B        ; set P2.6
    ldb     direct,#01                ; direction = forward
    add     position,#01
    addc    position+2,zero
    st_stat
    br
going_rev:
    andb    PORT2,#10111111B        ; clear P2.6
    ldb     direct,#00                ; direction = reverse
    sub     position,#01
    subc    position+2,zero

st_stat:
    stb     hsi_s0,last_stat

```

270061-55

Listing 4-10. Motor Control Mode 0 Routines

After mode correctness is confirmed and the *LASTx_TIME* values are updated the *LAST_STAT* (Last Status) variable is used to determine the current direction of travel. The *POSITION* value is then updated in the direction specified by the last two edges and the status is stored. Note that the first time in Mode 0 after being in Mode 1, the Mode 1 *done_chk* routine is used to update *POSITION*, instead of the routines *going_fwd* and *going_rev* from the Mode 0 section of code. The completion section of code is then executed.

Providing the PWM value to drive the motor is done by a routine running under Software Timer 1. The first section of code, shown in Listing 4-11a, has to do with calculating the position and timer errors. Listing 4-11b shows the next section of code where the power to be supplied to the motor is calculated. First the direction is checked and if the direction is reverse the absolute value of the error is taken. If the error is greater than 64K counts, the PWM routine is loaded with the maximum value. The next check is made to see if the motor

is close enough to the desired location that the power to it should be reversed, (i.e., enter the Braking mode). If the motor is very close to the position or has slowed to the point that is likely to turn around, the *Hold_Position mode is entered*.

The determination of which modes are selected under what conditions was done empirically. All of the parameters used to determine the mode are kept in RAM so they can be easily changed on the fly instead of by re-assembling the program. The parameters in the listing have been selected to make the motor run, but have not been optimized for speed or stability. A diagram of the modes is shown in Figure 4-7.

In the *Hold_Position* mode power is eased onto the motor to lock it into position. Since the motor could be stopped in this mode, some integral control is needed, as proportional control alone does not work well when the error is small and the load is large. The BOOST variable provides this integral control by increasing the output a fixed amount every time period in which the

Listing 4-11. Motor Control Software Timer 1 Routine

```

; ; ; ; ; SOFTWARE TIMER ROUTINE 1 ; ; ; ; ;
; ; ; ; ;
CSEG AT 2600H

swtl_expired:

    pushf
    orb     portl,$10000000B      ; set portl.7

    ldb     int_mask,$00001101B   ; enable HSI, Tovf, HSO

    ldb     HSO_COMMAND,$39H
    add     HSO_TIME,TIMER1,swtl_dly

    ld      time_err+2,des_time+2  ; Calculate time & position error
    ld      pos_err+2,des_pos+2
    sub     time_err,des_time,time    ; values are set
    subc    time_err+2,time+2
    sub     pos_err,des_pos,position
    subc    pos_err+2,position+2

    EI

    sub     time_delta,last_time_err,time_err
    ld      last_time_err,time_err

    sub     pos_delta,last_pos_err,pos_err
    ld      last_pos_err,pos_err

; ; ; ; ; Time_err = Desired time to finish - current time
; ; ; ; ; Pos_err = Desired position to finish - current position
; ; ; ; ; Pos_delta = Last position error - Current position error
; ; ; ; ; Time_delta = Last time error - Current time error
; ; ; ; ; note that errors should get smaller so deltas will be
; ; ; ; ; positive for forward motion (time is always forward)

```

270061-56

Listing 4-11a. Motor Control Software Position Counter

```

chk_dir:
    cmp     pos_err+2,zero
    jge     go_forward

go_backward:
    neg     pos_err          ; Pos_err = ABS VAL (pos_err)
    ldb     pwm_dir,000h
    cmp     pos_err+2,0ffffH
    jne     ld_max
    br      chk_brk

go_forward:
    ldb     pwm_dir,001H
    cmp     pos_err+2,zero
    je      chk_brk

ld_max:   ldb     pwm_pwr,max_pwr
    br      chk_sanity

chk_brk:
    ; Position_Error now = ABS(pos_err)
    cmp     pos_err,pos_pnt
    jnh     hold_position    ; position_error < position_control_point
    cmp     pos_err,brk_pnt
    jh      ld_max           ; position_error > brake_point

braking:
    cmp     pos_delta,zero
    jge     chk_delta
    neg     pos_delta

chk_delta:
    cmp     pos_delta,vel_pnt    ; velocity = pos_delta/sample_time
    jnh     hold_position        ; jmp if ABS(velocity) < vel_pnt

brake:    ldb     pwm_pwr,max_brk
    ldb     tmp,direct           ; If braking apply power in opposite
    notb    tmp                 ; direction of current motion
    ldb     pwm_dir,tmp
    br      ld_pwr

ld_pwr:   ; position hold mode
    cmp     pos_err,02
    jh      calc_out           ; if position error < 2 then turn off power
    clr     tmp+2
    clr     boost
    BR      output

calc_out:
    mulub   tmp,max_hold,#255
    mulu    tmp,pos_err        ; Tmp = pos_err * max_hold
    cmp     pos_delta,zero
    jne     no_bst
    add     boost,#04          ; Boost is integral control
    add     tmp+2,boost        ; TMP+2 = MSB(pos_err*max_hold)
    br      ck_max
no_bst:   clr     boost
ck_max:   cmp     tmp+2,max_hold
    jnh     output
maxed:    ld      tmp+2,max_hold
    output:  ldb     pwm_pwr,tmp+2

chk_sanity:
    br      ld_pwr

ld_pwr:   ldb     rpwr,pwm_pwr
    notb    rpwr
    jbs     pwm_dir,0,p2fwd

p2bkwd:   DI
    andb    port2,#01111111B    ; clear P2.7
    ldb     pwm_control,rpwr
    EI
    br      pwrset

p2fwd:    DI
    orb     port2,#10000000B    ; set P2.7
    ldb     pwm_control,rpwr
    EI

```

270061-57

Listing 4-11b: Motor Control Power Algorithm

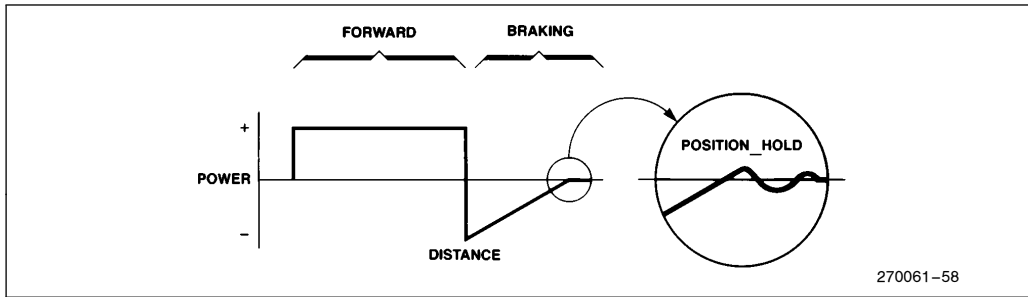


Figure 4-7. Motor Control Modes

error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is the proto-

```

pwrset:  cmp     time_err+2,zero  ; do pos_table when err is negative
        jgt     end_p
        br      end_p
; ; ;
        cmp     nxt_pos,#(32+pos_table)
        jlt     get_vals         ; jump if lower
        ld      nxt_pos,#pos_table
        clr     time+2
get_vals:
        ld      des_pos,[nxt_pos]+
        ld      des_pos+2,[nxt_pos]+
        ld      des_time+2,[nxt_pos]+
        ld      max_pwr,[nxt_pos]+
        ld      max_brk,max_pwr
        add     des_pos,offset
        addc    des_pos+2,zero
        sub     last_pos_err,des_pos,position

end_p:   andb    port1,#01111111B      ; clear P1.7
        popf
        ret

pos_table:
        dcl     00000000H      ; position 0
        dcw     0020H, 0080H   ; next time, power
        dcl     0000c000H      ; position 1
        dcw     0040H, 0040H   ; next time, power
        dcl     00000000H      ; position 2
        dcw     0060H, 00c0H   ; next time, power
        dcl     0FFFF8000H      ; position 3
        dcw     0080H, 0080H   ; next time, power

        dcl     00000800H      ; position 4
        dcw     0058H, 0080H   ; next time, power
        dcl     00003000H      ; position 5
        dcw     0070H, 00ffH   ; next time, power
        dcl     00000000H      ; position 6
        dcw     0090H, 00f0H   ; next time, power
        dcl     00000000H      ; position 7
        dcw     0091H, 00f0H   ; next time, power
    
```

270061-59

Listing 4-12. Motor Control Next Position Lookup

type will need to be reset, so the sanity check was not used. If one were desired, it could be as simple as checking a hardware generated direction indicator, or as complex as checking motor condition and other environmental factors.

After all checks have been made, the power value is loaded to the RPWR register using a software inversion to compensate for the hardware inversion. Direction is determined next and the power and direction are changed in adjacent instructions with interrupts disabled to prevent changing power without direction and vice versa.

To exercise the program logic the desired position is changed based on the time value using the code and lookup table shown in Listing 4-12.

The remaining sections of the program are relatively simple, but worth discussing briefly. The initialization routine initializes the I/O features and places several variables from ROM into RAM. Having these variables in RAM makes it easier to tweak the algorithm. Timer 1 is expanded into a 32-bit timer by the interrupt routine shown in Listing 4-13.

Software timer overhead is handled by the routine shown in Listing 4-14. In this routine the status of each timer bit is checked in a shadow register. If any of the timers have expired the appropriate routine is called.

[illegible]

270061-60

Listing 4-13. Motor Control Timer Interrupt Routine

```

; ; ; ; ; SOFTWARE TIMER INTERRUPT SERVICE ROUTINE ; ; ; ; ;
; ; ; ; ;
CSEG AT 2220H

soft_tm_r_int:
    pushf
    orb     iosl_bak,IOSI

chk_sw0:
    jbc     iosl_bak,0,chk_sw0_1
    andb    iosl_bak,$1111110B      ; Clear bit 0 - end swt0
chk_sw0_1: call    swt0_expired
    jbc     iosl_bak,1,chk_sw1
    andb    iosl_bak,$1111101B      ; Clear bit 1
chk_sw1:   call    swt1_expired
    jbc     iosl_bak,2,chk_sw2
    andb    iosl_bak,$1111101B      ; Clear bit 2
chk_sw2:   call    swt2_expired
    jbc     iosl_bak,4,swt_int_done
    andb    iosl_bak,$1111011B      ; Clear bit 3
chk_sw3:   call    swt3_expired
; , call

swt_int_done:
    popf
    ret     ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

Select

```

270061-B2

Listing 4-14. Motor Control Software Timer Interrupt Handler

270061-61

Listing 4-15: Motor Control Software Timer 2 Routine

always operates smoothly and provides at least 200 micro-seconds between the last several edges of Phase A before reversing. This idea was originally tried because the motor was not characterized thoroughly at first, and caused problems because of the motors tendency to stop suddenly when its speed was low.

If an encoder has a lower line count and therefore more time between output pulses the two mode solution can be used. The software for the two mode version can be easily extracted from the three mode version, so it will not be presented.

5.0 HARDWARE EXAMPLE

5.1. EPROM Only Minimum System

The diagram in Figure 5-1 illustrates how to connect an 8096 in a minimum configuration system. Either 2764s or 27128s can be used in the system. Note that the lower EPROM contains the even bytes while the upper

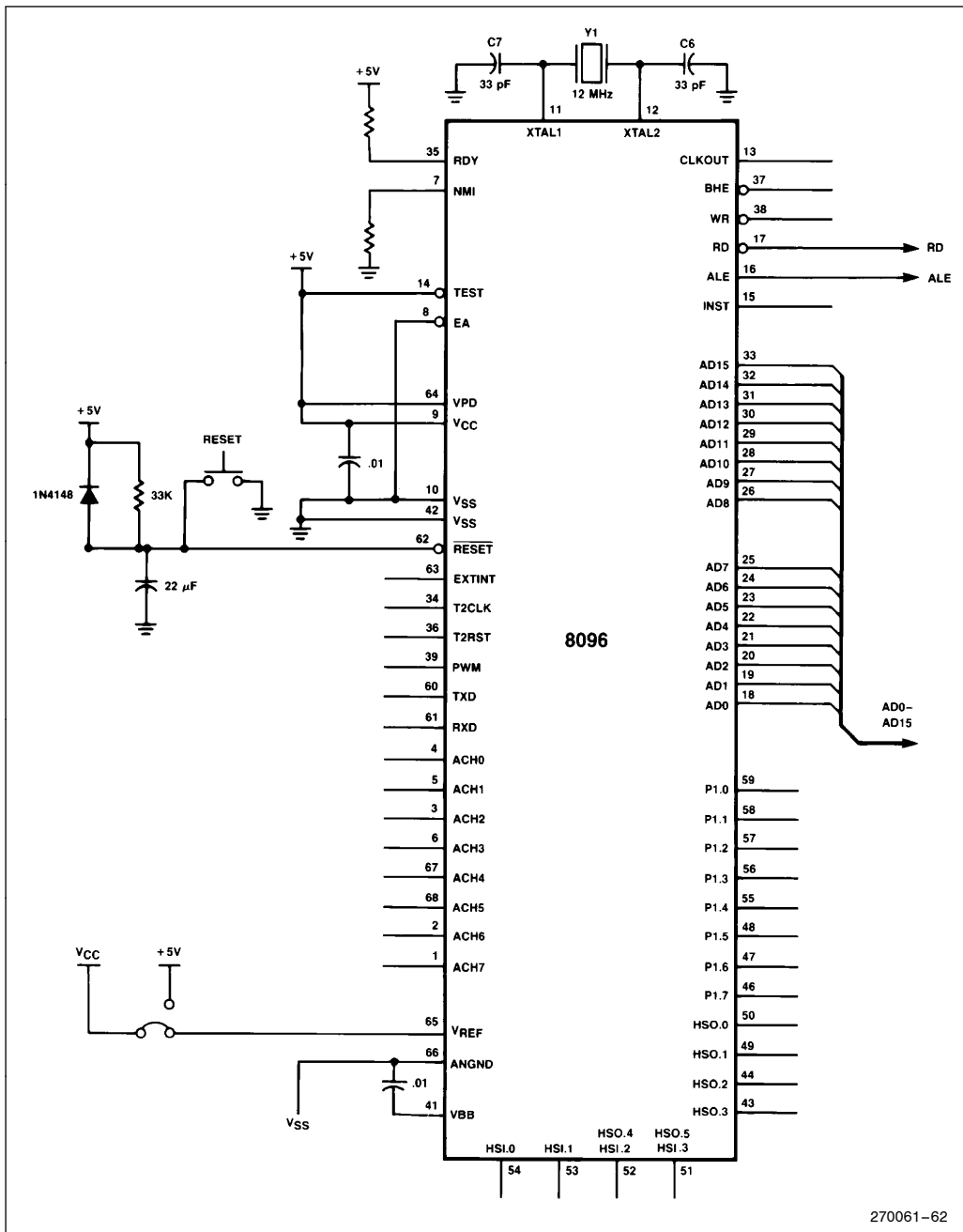


Figure 5-1 (1 of 2).

one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H while the upper

4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

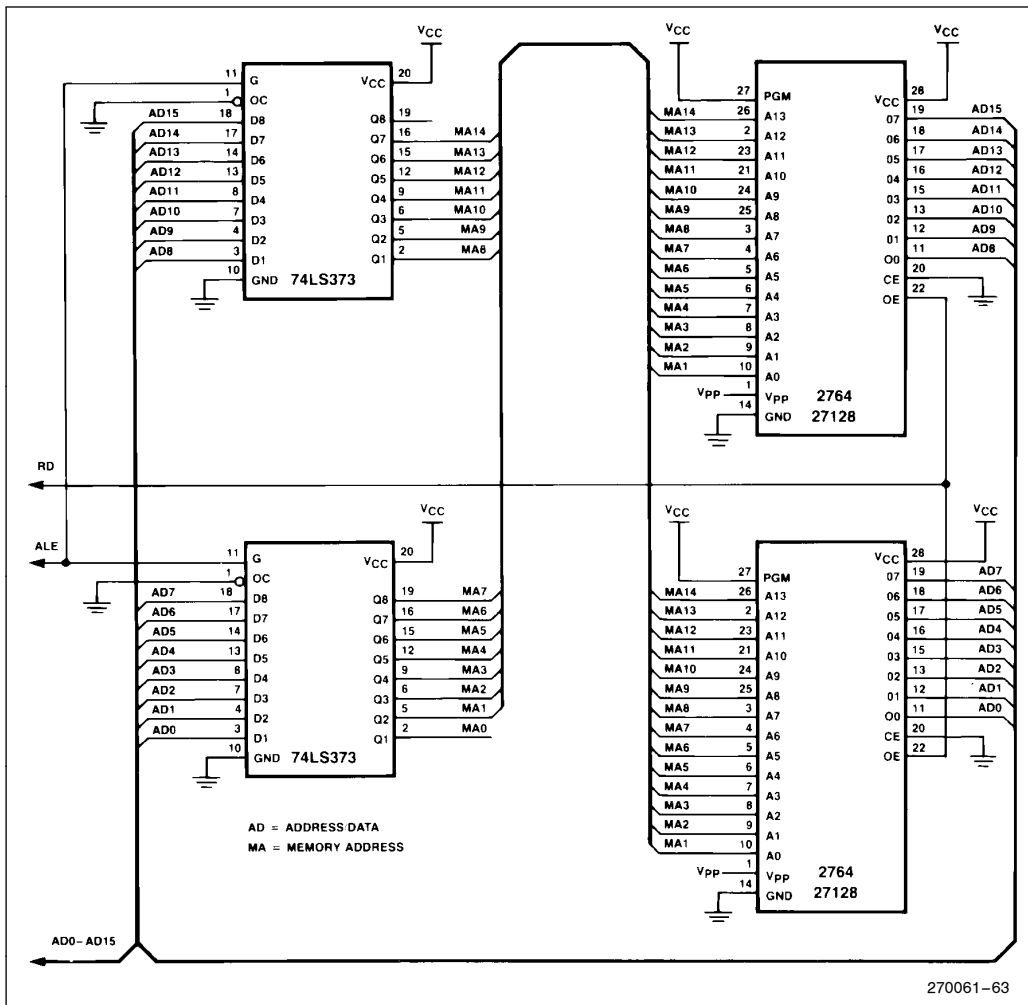


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

5.2. Port Reconstruction

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if

the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (i.e., address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

6.0 CONCLUSION

An overview of the MCS-96 family has been presented along with several simple examples and a few more complex ones. The source code for all of these programs are available in the Insite Users Library using order code AE-16. Additional information on the 8096 can be found in the Microcontroller Handbook and it is recommended that this book be in your possession before attempting any work with the MCS-96 family of products. Your local Intel sales office can assist you in getting more information on the 8096 and its hardware and software development tools.

7.0 BIBLIOGRAPHY

1. MSC-96 Macro Assembler User's Guide, Intel Corporation, 1983.
Order number 122048-001.
2. Microcontroller Handbook (1985), Intel Corporation, 1984.
Order number 210918-002.
3. MSC-96 Utilities User's Guide, Intel Corporation, 1983.
Order number 122049-001.
4. PL/M-96 User's Guide, Intel Corporation, 1983.
Order number 122134-001.

APPENDIX A BASIC SOFTWARE EXAMPLES

```

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0
SOURCE FILE: F3.INTER1.A96
OBJECT FILE: F3.INTER1.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('INTER1.A96: Interpolation routine 1')
2 $INCLUDE('F3.DEMO96.ASM') ; Include demo definitions
3 $INCLUDE('F3.DEMO96.INC') ; Include demo definitions
4 $nolist ; Turn listing off for include file
5 $include ; End of include file
6
7 RSEG at 22H
8
9 IN_VAL: dsb 1 ; Actual Input Value
10 TABLE_LOW: dsb 1
11 TABLE_HIGH: dsb 1
12 IN_DIF: dsb 1 ; Upper Input - Lower Input
13 IN_DIFB: equ IN_DIF ; byte ; Upper Output - Lower Output
14 TAB_DIF: dsb 1
15 OUT: dsb 1
16 RESULT: dsb 1
17 OUT_DIF: dsb 1 ; Delta Out
18
19 CSEG at 2080H
20
21 LD SP, #100H
22
23 look: LDB AL, IN_VAL ; Load temp with Actual Value
24 SHR8 AL, #3 ; Divide the byte by 8
25 ANDB AL, #1111110B ; Insure AL is a word address
26 ; This effectively divides AL by 2
27 ; so AL = IN_VAL/16
28
29 LDB8 AX, AL ; Load byte AL to word AX
30 LD TABLE_LOW, TABLE [AX] ; TABLE_LOW is loaded with the value
31 ; in the table at table location AX
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81

```

270061-64

A.1. Table Lookup 1

2095	A31D022126	LD	TABLE_HIGH, (TABLE+2)[AX]	; TABLE_HIGH is loaded with the	82
				; value in the table at table	83
				; location AX+2	84
				; (The next value in the table)	85
					86
209A	4824262A	SUB	TAB_DIF, TABLE_HIGH, TABLE_LOW	; TAB_DIF=TABLE_HIGH-TABLE_LOW	87
					88
209E	510F222B	ANDB	IN_DIFB, IN_VAL, #0FH	; IN_DIFB=least significant 4 bits	89
				; of IN_VAL	90
20A2	AC282B	LDBZ	IN_DIF, IN_DIFB	; Load byte IN_DIFB to word IN_DIF	91
					92
20A5	FE4C2A2B30	MUL	OUT_DIF, IN_DIF, TAB_DIF	; Output_difference =	93
				; Input_difference*Table_difference	94
				; Divide by 16 (2**4)	95
20AA	0E0430	SHR	OUT_DIF, #4		96
					97
20AD	4424302C	ADD	OUT, OUT_DIF, TABLE_LOW	; Add output difference to output	98
				; generated with truncated IN_VAL	99
				; as input	100
20B1	0A042C	SHR	OUT, #4		101
20B4	A4002C	ADDC	OUT, zero	; Round to 12-bit answer	102
				; Round up if Carry = 1	103
20B7	C02E2C	ST	OUT, RESULT	; Store OUT to RESULT	104
					105
20BA	27C8	BR	look	; Branch to "look."	106
					107
					108
					109
2100		AT	2100H		110
					111
2100	000000200034004C	DCW	0000H, 2000H, 3400H, 4C00H	; A random function	112
210B	005D006A0072007B	DCW	5D00H, 6A00H, 7200H, 7B00H		113
2110	007B007D0076006D	DCW	7B00H, 7D00H, 7600H, 6D00H		114
211B	005D004B00340022	DCW	5D00H, 4B00H, 3400H, 2200H		115
2120	0010	DCW	1000H		116
					117
2122		END			118
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.					

270061-65

A.1. Table Lookup 1 (Continued)

270061-66

A.2. Table Lookup 2

209A 510F242A	87	ANDB	IN_DIFB, IN_VAL, #0FH	; IN_DIFB=least significant 4 bits
209E AC2A2A	88			; of IN_VAL
20A1 FE4C2B2A30	89	LDBZE	IN_DIF, IN_DIFB	; Load byte IN_DIFB to word IN_DIF
	90			
	91	MUL	OUT_DIF, IN_DIF, TABLE_INC	; Output_difference =
	92			; Input_difference*Incremental_change
	93			
20A6 4426302C	94			
	95	ADD	OUT, OUT_DIF, TABLE_LOW	; Add output difference to output
	96			; generated with truncated IN_VAL
	97			; as input
20AA 0B042C	98	SHR	OUT, #4	; Round to 12-bit answer
20AD A4002C	99	ADDC	OUT, zero	; Round up if Carry = 1
20B0 C02E2C	100			
20B3 27CF	101	no_inc: ST	OUT, RESULT	; Store OUT to RESULT
	102	BR	look	; Branch to "look."
	103			
	104			
2100	105	cseg	AT 2100H	
	106			
2100	107	val_table:		
2100 000000200034004C	108	DCW	0000H, 2000H, 3400H, 4C00H	; A random function
2108 005D006A0072007B	109	DCW	5D00H, 6A00H, 7200H, 7B00H	
2110 007B007D0076006D	110	DCW	7B00H, 7D00H, 7600H, 6D00H	
2118 005D004B00340022	111	DCW	5D00H, 4B00H, 3400H, 2200H	
2120 0010	112	DCW	1000H	
2122	113	inc_table:		
2122 00024001B0011001	114	DCW	0200H, 0140H, 01B0H, 0110H	; Table of incremental
212A D000B00060003000	115	DCW	00D0H, 00B0H, 0060H, 0030H	; differences
2132 200090FF70FF00FF	116	DCW	00020H, 0FF90H, 0FF70H, 0FF00H	
213A E0FE90FEE0FEE0FE	117	DCW	0FEEOH, 0FE90H, 0FEE0H, 0FEE0H	
2142	118			
	119	END		
ASSEMBLY COMPLETED. NO ERROR(S) FOUND.				

270061-67

A.2. Table Lookup 2 (Continued)



270061-69

PL/M-96 COMPILER PLMEX1: PLM-96 Example Code for Table Lookup
ASSEMBLY LISTING OF OBJECT CODE

```

0022          ; STATEMENT 14
0022          PLMEX: LD SP, #STACK
0026          LOOP: LD TEMP, IN_VAL
0026          R      SHR TEMP, #4H
0029          R      ; STATEMENT 15
002C          R      ADD TMP0, TEMP
0030          R      LD TABLE_LOW, TABLE[TMP0]
0035          R      ; STATEMENT 16
0035          R      LD TABLE_HIGH, TABLE+2H[TMP0]
003A          R      ; STATEMENT 17
003A          R      SUB TABLE_DIF, TABLE_HIGH, TABLE_LOW
003E          R      ; STATEMENT 18
0040          R      PUSH TABLE_DIF
0040          R      AND TMP0, IN_VAL, #OFH
0045          R      PUSH TMP0
0047          E      CALL DMPY
004A          R      SHRAL TMP0, #4H
004D          R      LD OUT_DIF+2H, TMP2
0050          R      LD OUT_DIF, TMP0
0053          R      ; STATEMENT 19
0053          R      LD TMP4, TABLE_LOW
0056          R      EXT TMP4
005B          R      ADD TMP4, TMP0
005B          R      ADDC TMP6, TMP2
005E          R      SHRAL TMP4, #4H
0061          R      LD OUT, TMP4
0064          R      ; STATEMENT 20
0064          R      LDB TMP0, #OFFH
0067          R      BC @0003
0069          R      CLRB TMP0
006B          R      @0003:

```

270061-70

A.3. PLM-96 Code with Expansion (Continued)

006B 9B1C00	CMPB	RO, THP0
006E D705	BNE	@0001
	;	STATEMENT 21
0070 A0200A	LD	RESULT, THP4
0073 2005	BR	@0002
	;	STATEMENT 22
0075	@0001:	
0075 A0080A	LD	RESULT, OUT
007B 070A	INC	RESULT
	;	STATEMENT 23
007A	@0002:	
007A 27AA	BR	LOOP
	;	STATEMENT 24
	END	
MODULE INFORMATION:		
CODE AREA SIZE	=	005AH 90D
CONSTANT AREA SIZE	=	0022H 34D
DATA AREA SIZE	=	0000H 0D
STATIC REGS AREA SIZE	=	0012H 18D
PL/M-96 COMPILER PLMEX1: PLM-96 Example Code for Table Lookup		
ASSEMBLY LISTING OF OBJECT CODE		
OVERLAYABLE REGS AREA SIZE	=	0000H 0D
MAXIMUM STACK SIZE	=	0006H 6D
48 LINES READ		
PL/M-96 COMPILATION COMPLETE	0 WARNINGS,	0 ERRORS
		270061-71

A.3. PLM-96 Code with Expansion (Continued)



MCS-96 MACRO ASSEMBLER			MULT APT: 16*16 multiply procedure for PLM-96		
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0					
SOURCE FILE: F3:MULT.A96					
OBJECT FILE: F3:MULT.OBJ					
CONTROLS SPECIFIED IN INVOCATION COMMAND: NDSB					
ERR LOC	OBJECT	LINE	SOURCE STATEMENT		
		1	\$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')		
		2			
		3			
001B		4	SP	EGU	1BH: word
		5			
0000		6	rseg		
		7	EXTRN	PLMREG	: long
		8			
0000		9	cseg		
		10			
		11	PUBLIC	DMPY	
		12			
		13			
0000 CC04	E	14	DMPY:	PLMREG+4	
0002 CC00	E	15	POP	PLMREG	
0004 FE6E1900	E	16	MUL	PLMREG,[SP]++	
		17			
000B E304	E	18	BR	[PLMREG+4]	
000A		19	END		
ASSEMBLY COMPLETED,			NO ERROR(S) FOUND.		

270061-72

270061-72

A.3. PLM-96 Code with Expansion (Continued)

SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation
INPUT FILES: F3:PLMEX1.OBJ, F3:MULT.OBJ, PLM96.LIB
OUTPUT FILE: F3:PLMOUT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:
ROM(2080H-3FFFH)

INPUT MODULES INCLUDED:
F3:PLMEX1.OBJ(PLMEX) 12/25/84
F3:MULT.OBJ(MULT) 12/25/84
PLM96.LIB(PLMREG) 11/02/83

SEGMENT MAP FOR F3:PLMOUT.OBJ(PLMEX):

TYPE	BASE	LENGTH	ALIGNMENT	MODULE NAME
***RESERVED*	0000H	001AH		
GAP	001AH	0002H		
REG	001CH	0008H	ABSOLUTE	PLMREG
REG	0024H	0012H	WORD	PLMEX
STACK	0036H	0006H	WORD	
GAP	003CH	2044H		
CODE	2080H	0003H	ABSOLUTE	PLMEX
GAP	2083H	0001H		
CODE	2084H	007CH	WORD	PLMEX
CODE	2100H	000AH	BYTE	MULT
GAP	210AH	DEF6H		



SYMBOL TABLE FOR : F3: PLMOUT.OBJ(PLMEX):		
ATTRIBUTES	VALUE	NAME
-----	----	----
REG WORD	0024H	PUBLICS:
REG INTEGER	0026H	IN_VAL
REG INTEGER	0028H	TABLE_LOW
REG INTEGER	002AH	TABLE_HIGH
REG INTEGER	002CH	TABLE_DIF
REG LONGINT	002EH	OUT
REG WORD	0030H	RESULT
REG ENTRY	0034H	OUT_DIF
REG LONG	2100H	TEMP
REG NULL	001CH	DMPY
REG NULL	003CH	PLMREG
REG NULL	1FC4H	MEMORY
		?MEMORY_SIZE
		MODULE: PLMEX
		MODULE: MULT
		MODULE: PLMREG
RL96 COMPLETED,	0 WARNING(S),	0 ERROR(S)

270061-74

A.3. PLM-96 Code with Expansion (Continued)



```

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0
SOURCE FILE: F3:PULSE A96
OBJECT FILE: F3:PULSE OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 1 $TITLE('PULSE A96: Measuring pulses using the HSI unit')
2 2
3 3 $INCLUDE(DEMO96.INC)
4 4 $nolist ; Turn listing off for include file
51 51
52 52 ; End of include file
53 53
54 54 rseg at 28H
55 55
56 56 HIGH_TIME: dsb 1
57 57 LOW_TIME: dsb 1
58 58 PERIOD: dsb 1
59 59 HI_EDGE: dsb 1
60 60 LO_EDGE: dsb 1
61 61
62 62
63 63
64 64 cseg at 2080H
65 65
66 66
67 67 LD SP, #100H
68 68 IOCO, #00000001B ; Enable HSI 0
69 69 LDB HSI_MODE, #00001111B ; HSI 0 look for either edge
70 70
71 71 wait: ADD PERIOD, HIGH_TIME, LOW_TIME
72 72 JBS IOS1, 6, contin ; If FIFO is full
73 73 JBC IOS1, 7, wait ; Wait while no pulse is entered
74 74
75 75 contin: LDB AL, HSI_STATUS ; Load status; Note that reading
76 76 ; HSI_TIME clears HSI_STATUS
77 77
78 78 LD BX, HSI_TIME ; Load the HSI_TIME
79 79
80 80 JBS AL, 1, hsi_hi ; Jump if HSI 0 is high
81 81
82 82 hsi_lo: ST BX, LO_EDGE
83 83 SUB HIGH_TIME, LO_EDGE, HI_EDGE
84 84 BR wait
85 85
86 86
87 87 hsi_hi: ST BX, HI_EDGE

```

270061-75

A.4. Pulse Measurement



20A9	48302E2A	88	SUB	LOW_TIME, HI_EDGE, LO_EDGE	270061-76
20AD	27DB	89	BR	wait	
20AF		90			
		91	END		
ASSEMBLY COMPLETED,		NO ERROR(S) FOUND.			

A.4. Pulse Measurement (Continued)





209D 3B2E05	88	JBS	HSI_S0,0,a_rise
20A0 3A2E0F	89	JBS	HSI_S0,2,a_fall
20A3 201A	90	BR	no_cnt
20A5 4B2C2B32	91		
20A9 4B2A2B30	92	a_rise: SUB	LOW_TIME, TIME, LAST_FALL
20AD A02B2A	93	SUB	PERIOD, TIME, LAST_RISE
20B0 200B	94	LD	LAST_RISE, TIME
	95	BR	increment
20B2 4B2A2B34	96		
20B6 4B2C2B30	97	a_fall: SUB	HIGH_TIME, TIME, LAST_RISE
20BA A02B2C	98	SUB	PERIOD, TIME, LAST_FALL
	99	LD	LAST_FALL, TIME
20BD 0736	100		
20BF 27CC	101	increment: INC	COUNT
	102		
20C1	103	no_cnt: BR	wait
	104		
	105	END	
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.			

270061-78

A.5. Enhanced Pulse Measurement (Continued)



```

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0
SOURCE FILE: F3:HSDRV.A96
OBJECT FILE: F3:HSDRV.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('HSDRV.A96: Driver module for HSO PWM program')
2
3 HSDRV MODULE MAIN, STACKSIZE(8)
4
5
6 PUBLIC HSO_ON_0, HSO_OFF_0
7 PUBLIC HSO_ON_1, HSO_OFF_1
8 PUBLIC HSO_TIME, HSO_COMMAND
9 PUBLIC SP, TIMER1, IOSO
10
11 $INCLUDE(DEMO96.INC)
12 $nolist ; Turn listing off for include file
13 $=1 ; End of include file
14
15 rseg at 28H
16
17 EXTRN OLD_STAT : byte
18
19 HSO_ON_0: dsb 1
20 HSO_OFF_0: dsb 1
21 HSO_ON_1: dsb 1
22 HSO_OFF_1: dsb 1
23 count: dsb 1
24
25 cseg at 2080H
26
27 EXTRN wait : entry
28
29 strt: DI SP, #100H
30 LD ANDB OLD_STAT, IOSO, #0FH
31 XORB OLD_STAT, #0FH
32
33 initial: LD CX, #0100H
34
35 loop: LD AX, #1000H
36 SUB BX, AX, CX
37 LD AX, CX
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87

```

270061-79

A.6. PWM Using the HSO



209B C02B1C	88	ST	AX, HSO_ON_0
209E C02A20	89	ST	BX, HSO_OFF_0
	90		
20A1 0B011C	91	SHR	AX, #1
20A4 0B0120	92	SHR	BX, #1
20A7 C02C1C	93	ST	AX, HSO_ON_1
20AA C02E20	94	ST	BX, HSO_OFF_1
	95		
20AD EF0000	96	CALL	wait
	97		
20B0 0722	98	INC	CX
20B2 89000F22	99	CMP	CX, #00F00H
20B6 D7DB	100	BNE	loop
	101		
20B8 27D2	102	BR	initial
	103		
20BA	104	END	
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.			

270061-80

A.6. PWM Using the HSO (Continued)



```

SERIES-111 MCS-96 MACRO ASSEMBLER, V1 0
SOURCE FILE: F3.HSOMOD A96
OBJECT FILE: F3.HSOMOD OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('HSOMOD A96: 8096 PWM PROGRAM MODIFIED FOR DRIVER')
2 $PAGEWIDTH(130)
3
4 ; This program will provide 3 PWM outputs on HSD pins 0-2
5 ; The input parameters passed to the program are:
6 ;
7 ; HSO_ON_N HSD on time for pin N
8 ; HSO_OFF_N HSD off time for pin N
9
10 ; Where: Times are in timer1 cycles
11 ; N takes values from 0 to 3
12
13
14
15
16 ; NOTE: Use this file to replace the declaration section of
17 ; the HSD PWM program from "INCLUDE(DEMO96.INC)" through
18 ; the line prior to the label "wait". Also change the last
19 ; branch in the program to a "RET".
20
21 RSEG
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

D_STAT: DSB 1
extrn HSO_ON_0:word, HSO_OFF_0:word
extrn HSO_ON_1:word, HSO_OFF_1:word
extrn HSO_TIME:word, HSO_COMMAND:byte
extrn TIMER1:word, IOSO:byte
extrn SP:word

public OLD_STAT
OLD_STAT: dsb 1
NEW_STAT: dsb 1

cseg
PUBLIC wait
wait: JBS IOSO, 6, wait ; Loop until HSD holding register
NOP ; is empty

; For operation with interrupts 'store_stat' would be the
; entry point of the routine
; Note that a DI or PUSHF might have to be added.

```

270061-81

A.6. PWM Using the HSO (Continued)

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-82

```

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0
SOURCE FILE  F3 SP A96
OBJECT FILE  F3 SP OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND. NOSB

ERR LOC  OBJECT      LINE  SOURCE STATEMENT
      1  1  $TITLE('SP.A96: SERIAL PORT DEMO PROGRAM')
      2  2
      3  3
      4  4  $INCLUDE(DEMO96.INC)
      5  5  $nolist , Turn listing off for include file
      6  6  $nolist , End of include file
      7  7
      8  8
      9  9
     10 10
     11 11
     12 12
     13 13
     14 14
     15 15
     16 16
     17 17
     18 18
     19 19
     20 20
     21 21
     22 22
     23 23
     24 24
     25 25
     26 26
     27 27
     28 28
     29 29
     30 30
     31 31
     32 32
     33 33
     34 34
     35 35
     36 36
     37 37
     38 38
     39 39
     40 40
     41 41
     42 42
     43 43
     44 44
     45 45
     46 46
     47 47
     48 48
     49 49
     50 50
     51 51
     52 52
     53 53
     54 54
     55 55
     56 56
     57 57
     58 58
     59 59
     60 60
     61 61
     62 62
     63 63
     64 64
     65 65
     66 66
     67 67
     68 68
     69 69
     70 70
     71 71
     72 72
     73 73
     74 74
     75 75
     76 76
     77 77
     78 78
     79 79
     80 80
     81 81
     82 82
     83 83
     84 84
     85 85

0028
0029
002A
002B
002C
200C
200C 9C20
2080
2080 A100011B
2084 B12016

0027
0080
0026
2087 B1260E
208A B1800E

      1  $TITLE('SP.A96: SERIAL PORT DEMO PROGRAM')
      2
      3
      4  $INCLUDE(DEMO96.INC)
      5  $nolist , Turn listing off for include file
      6  $nolist , End of include file
      7
      8
      9
     10
     11
     12
     13
     14
     15
     16
     17
     18
     19
     20
     21
     22
     23
     24
     25
     26
     27
     28
     29
     30
     31
     32
     33
     34
     35
     36
     37
     38
     39
     40
     41
     42
     43
     44
     45
     46
     47
     48
     49
     50
     51
     52
     53
     54
     55
     56
     57
     58
     59
     60
     61
     62
     63
     64
     65
     66
     67
     68
     69
     70
     71
     72
     73
     74
     75
     76
     77
     78
     79
     80
     81
     82
     83
     84
     85

      CHR:      dsb      1
    SPTEMP:    dsb      1
    TEMPO:     dsb      1
    TEMP1:     dsb      1
    RCV_FLAG:  dsb      1

      cseg      at 200CH
      DCW      ser_port_int

      cseg      at 2080H
      LD        SP, #100H
      LDB      IOC1, #00100000B      ; Set P2.0 to TXD
      ; Baud rate = input frequency / (64*baud_val)
      ; baud_val = (input frequency/64) / baud rate

      baud_val equ 39      ; 39 = (12,000,000/64)/4800 baud
      BAUD_HIGH equ ((baud_val-1)/256) OR 80H      ; Set MSB to 1
      BAUD_LOW  equ (baud_val-1) MOD 256

      LDB      BAUD_REG, #BAUD_LOW
      LDB      BAUD_REG, #BAUD_HIGH

```

270061-83

A.7. Serial Port

208D B14911	86	LDB	SPCON, #01001001B	; Enable receiver, Mode 1
	87			; The serial port is now initialized
	88			
	89			
2090 C42807	90	STB	SBUF, CHR	; Clear serial Port
2093 B1202A	91	LDB	TEMPO, #00100000B	; Set TI-temp
	92			
2096 B1400B	93	LDB	INT_MASK, #01000000B	; Enable Serial Port Interrupt
2099 FB	94	EI		
209A 27FE	95	loop:		
	96	BR		; Wait for serial port interrupt
	97			
	98			
209C	99	ser_port_int:		
209C F2	100	PUSHF		
209D	101	rd_again:		
209D B01129	102	LDB	SPSTAT	; This section of code can be replaced
20A0 90292A	103	ORB	TEMPO, SPTEMP	; with "ORB TEMPO, SP_STAT" when the
20A3 716029	104	ANDB	SPTEMP, #01100000B	; serial port TI and RI bugs are fixed
20A6 D7F5	105	JNE	rd_again	; Repeat until TI and RI are properly cleared
	106			
20AB	107	get_byte:		
20AB 362A09	108	JBC	TEMPO, 6, put_byte	; If RI-temp is not set
20AB C42807	109	STB	SBUF, CHR	; Store byte
20AE 71BF2A	110	ANDB	TEMPO, #1011111B	; CLR RI-temp
20B1 B1FF2C	111	LDB	RCV_FLAG, #0FFH	; Set bit-received flag
	112			
20B4	113	put_byte:		
20B4 302C1B	114	JBC	RCV_FLAG, 0, continue	; If receive flag is cleared
20B7 352A15	115	JBC	TEMPO, 5, continue	; If TI was not set
20BA B02807	116	LDB	SBUF, CHR	; Send byte
20BD 71DF2A	117	ANDB	TEMPO, #1101111B	; CLR TI-temp
	118			
20C0 717F2B	119	ANDB	CHR, #0111111B	; This section of code appends
20C3 990D2B	120	CMPB	CHR, #0DH	; an LF after a CR is sent
20C6 D705	121	JNE	clr_rcv	
20CB B10A2B	122	LDB	CHR, #0AH	
20CB 2002	123	BR	continue	
	124			
20CD	125	clr_rcv:		
20CD 112C	126	CLRB	RCV_FLAG	; Clear bit-received flag
	127			
20CF	128	continue:		
20CF F3	129	POPF		
20D0 F0	130	RET		
	131			
20D1	132	END		
	133			
ASSEMBLY COMPLETED,		NO ERROR(S) FOUND.		

270061-84

A.7. Serial Port (Continued)

```

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0
SOURCE FILE: F3:ATOD.A96
OBJECT FILE: F3:ATOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('ATOD.A96: SCANNING THE A TO D CHANNELS')
2
3 $INCLUDE(DEMO96.INC)
4 $nolist ; Turn listing off for include file
52 ; End of include file
53
54 RSEG at 2BH
55
56 BL EQU BX:BYTE
57 DL EQU DX:BYTE
58
59 RESULT_TABLE:
60 RESULT_1: dsb 1
61 RESULT_2: dsb 1
62 RESULT_3: dsb 1
63 RESULT_4: dsb 1
64
65 cseg at 2080H
66
67
68
69 start: LD SP, #100H ; Set Stack Pointer
70 CLR BX
71
72 next: AADB AD_COMMAND, BL, #1000B ; Start conversion on channel
73 ; indicated by BL register
74
75 NOP ; Wait for conversion to start
76 NOP
77 check: JBS AD_RESULT_LO, 3, check ; Wait while A to D is busy
78
79 LDB AL, AD_RESULT_LO ; Load low order result
80 LDB AH, AD_RESULT_HI ; Load high order result
81
82 AADB DL, BL, BL ; DL=BL*2
83 LD8ZE DX, DL
84 ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2
85
86 INCB BL ; Increment DL modulo 4

```

270061-85

A.8. A to D Converter



20A2	710320	87	ANDB	BL, #03H	
20A5	27DF	88	BR	next	
20A7		89	END		
		90			
		91			
ASSEMBLY COMPLETED,		NO ERROR(S) FOUND.		270061-86	

A.8. A to D Converter (Continued)



APPENDIX B HSD AND A TO D UNDER INTERRUPT CONTROL

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0									
SOURCE FILE: F3:A2DHSO.A96									
OBJECT FILE: F3:A2DHSO.OBJ									
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB									
ERR LOC	OBJECT	LINE	SOURCE STATEMENT						
		1	\$TITLE ('A2DHSO.A96: GENERATING PWM OUTPUTS FROM A TO D INPUTS')						
		2							
		3	; This program will provide 3 PWM outputs on HSD pins 0-2						
		4	; and one on the PWM.						
		5	;						
		6	; The PWM values are determined by the input to the A/D converter.						
		7	;						
		8	;						
		9							
		10	\$INCLUDE(DEMO96.INC)						
		11	\$nolist ; Turn listing off for include file						
		59	; End of include file						
		60							
002B		61	RSEG AT 28H						
		62							
001E		63		DL	DL	DX:BYTE			
		64							
002B		65	ON_TIME:						
002B		66		PWM_TIME_1:		DSW	1		
002A		67		HSD_ON_0:		DSW	1		
002C		68		HSD_ON_1:		DSW	1		
002E		69		HSD_ON_2:		DSW	1		
		70							
0030		71	RESULT_TABLE:						
0030		72		RESULT_0:		DSW	1		
0032		73		RESULT_1:		DSW	1		
0034		74		RESULT_2:		DSW	1		
0036		75		RESULT_3:		DSW	1		
		76							
003B		77		NXT_ON_T:		DSW	1		
003A		78		NXT_OFF_0:		DSW	1		
003C		79		NXT_OFF_1:		DSW	1		
003E		80		NXT_OFF_2:		DSW	1		
0040		81		COUNT:		DSL	1		
0044		82		AD_NUM:		DSW	1		
0046		83		TMP:		DSW	1		
004B		84		HSD_PER:		DSW	1		
004A		85		LAST_LOAD:		DSB	1		
		86							
Channel being converted									
270061-87									

2000	87	cseg	AT 2000H	
2000 8020	88		DCW	start
2002 1021	89		DCW	Atod_do
2004 8020	90		DCW	start
2006 CC20	91		DCW	HSD_exe
	92			
	93			
	94	\$EJECT		
	95			
2080	96	cseg	AT 2080H	
	97			
2080 A1000118	98	start:	LD	SP, #10
2084 011C	99		CLR	AX
2086 051C	100	wait:	DEC	AX
2088 D7FC	101		JNE	wait
	102			
208A 1144	103		CLRB	AD_NUM
	104			
208C A1800028	105		LD	PWM_TIME
2090 A1000148	106		LD	HSD_PER
2094 A140002A	107		LD	HSD_ON
209B A180002C	108		LD	HSD_ON
209C A1C0002E	109		LD	HSD_ON
	110			
20A0 4500010A3B	111		ADD	NXT_ON
	112			
20A5 B13606	113		LDB	HSD_COMP
20AB A03B04	114		LD	HSD_TIME
20AB FD	115		NOP	
20AC FD	116		NOP	
20AD B12206	117		LDB	HSD_COMP
20B0 643B04	118		ADD	HSD_TIME
	119			
20B3 91074A	120		ORB	LAST_LD
20B6 B10A08	121		LDB	INT_MASK
20B9 B10A09	122		LDB	INT_PENL
20BC FB	123		EI	
	124			
20BD 91010F	125	loop:	ORB	Port1, #
20C0 65010040	126		ADD	COUNT, #
20C4 A40042	127		ADDC	COUNT+2,
20C7 71FE0F	128		ANDB	Port1, #
20CA 27F1	129		BR	loop
	130			
	131	\$EJECT		



270061-88

```

132 ..... HSD EXECUTED INTERRUPT .....
133 .....
134 .....
135 .....
136 .....
137 HSD_exec_int:
138   ORB      PUSHF
139   ORB      Port1, #00000010B      ; Set p1.1
140
141   SUB      TMP, TIMER1, NXT_ON_T
142   CMP      TMP, ZERO
143   JLT      set_off_times
144
145   set_on_times:
146   ADD      NXT_ON_T, HSD_PER
147   LDB      HSD_COMMAND, #00110110B      ; Set HSD for timer1, set pin 0,1
148   LD      HSD_TIME, NXT_ON_T
149
150   NOP
151   LDB      HSD_COMMAND, #00100010B      ; Set HSD for timer1, set pin 2
152   LD      HSD_TIME, NXT_ON_T
153
154   ORB      LAST_LOAD, #00000111B      ; Last loaded value was all ones
155
156   LDB      PWM_CONTROL, PWM_TIME_1      ; Now is as good a time as any
157   BR       check_done                  ; to update the PWM reg
158
159   check_done
160
161   set_off_times:
162   JBC      LAST_LOAD, 0, check_done
163
164   ADD      NXT_OFF_0, NXT_ON_T, HSD_ON_0
165   LDB      HSD_COMMAND, #00010000B      ; Set HSD for timer1, clear pin 0
166   LD      HSD_TIME, NXT_OFF_0
167
168   NOP
169   ADD      NXT_OFF_1, NXT_ON_T, HSD_ON_1
170   LDB      HSD_COMMAND, #00010001B      ; Set HSD for timer1, clear pin 1
171   LD      HSD_TIME, NXT_OFF_1
172
173   NOP
174   ADD      NXT_OFF_2, NXT_ON_T, HSD_ON_2
175   LDB      HSD_COMMAND, #00010010B      ; Set HSD for timer1, clear pin 2
176   LD      HSD_TIME, NXT_OFF_2
177
178   ANDB     LAST_LOAD, #11111000B      ; Last loaded value was all 0s
179
180   check_done:
181   ANDB     Port1, #11111101B      ; Clear P1.1

```

270061-89

APPENDIX C

SOFTWARE SERIAL PORT

```

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: :F3:SWPORT.A96
OBJECT FILE: :F3:SWPORT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 1 $TITLE('SWPORT.A96 : SOFTWARE IMPLEMENTED ASYNCHRONOUS SERIAL PORT')
2
3 ; This module provides a software implemented asynchronous serial port
4 ; for the 8096. HSD 5 is used for transmit data. HSI 2 is used for
5 ; receive data. Note: the choice of HSD 5 and HSI 2 is arbitrary.
6
7 $INCLUDE(DEMO96.INC)
8 $nolist ; Turn listing off for include file
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
10
```

```

2080      cseg at 2080h
2081  reset_loc:
2082      ; The 8096 starts executing here on reset, the program will initialize the
2083      ; the software serial port and run a simple test to exercise it
2084      ;
2085      di
2086      ld     sp, #0f0h
2087      push  #4800
2088      call  setup_serial_port
2089      ldb   int_mask, #01f01100b      ; serial, swt.hso.hsi
2090      ei
2091
2092      ;
2093      ;
2094      ;
2095      ;
2096      ;
2097      ;
2098      ;
2099      ;
2100      ;
2101      test1:
2102      ; A simple test of the serial port routines.
2103      ; While no characters are received an incrementing pattern is sent to the
2104      ; serial output. When a character is received the incrementing pattern
2105      ; "jumps" to the character received and proceeds from there.
2106      ;
2107      ;
2108      CR     equ    0DH
2109      ldb   char, #CR
2110      testloop:
2111      ldbz  ax, char
2112      push ax
2113      call char_out
2114
2115      cmpb char, #CR
2116      bne  nopause
2117      clr  ax
2118      pause:
2119      inc  ax
2120      bne  pause
2121      nopause:
2122
2123      incb char
2124
2125      test2:
2126      call csts
2127      al, 0
2128      cmpb be testloop
2129      call char_in
2130      ldb  char, al
2131      br  testloop

```

270061-92

```

0000      132      cseg
0000      133      ,
0000      134      setup_serial_port.
0000      135      ; Called on system reset to initiate the software serial port
0000      136      ;
0000      137      ,
0000      138      pop      cx      ; the return address
0000      139      pop      bx      ; the baud rate (in decimal)
0000      140      ld      dx, #0007h ; dx:ax:=500,000 (assumes 12 Mhz crystal)
0000      141      ld      ax, #0A120h
0000      142      divu     ax, bx      ; calculate the baud count (500,000/baudrate)
0000      143      st      ax, baud_count
0000      144      st      0, serial_out ; clear serial out
0000      145      ldb     ioctl, #01100000b ; Enable HSD 5 and Txd
0000      146      bbs     io50, 6, $      ; Wait for room in the HSD CAM
0000      147      add     txd_time, timer1, 20
0000      148      ldb     hso_command, #mark_command
0000      149      ld      hso_time, txd_time
0000      150      clrb    rcve_buf ; clear out the receive variables
0000      151      clrb    rcve_reg
0000      152      clrb    rcve_state
0000      153      call   init_receive ; setup to detect a start bit
0000      154      br     [cx]
0000      155      $reject
0000      156      ,
0000      157      ,
0000      158      char_out:
0000      159      ; Output character to the software serial port
0000      160      ,
0000      161      pop      cx      ; the return address
0000      162      pop      bx      ; the character for output
0000      163      ldb     (bx+1), #01h ; add the start and stop bits
0000      164      add     bx, bx      ; to the char and leave as 16 bit
0000      165      wait_for_xmit:
0000      166      cmp     serial_out, 0 ; wait for serial_out=0 (it will be cleared by
0000      167      bne     wait_for_xmit ; the hso interrupt process)
0000      168      st      bx, serial_out ; put the formatted character in serial_out
0000      169      br     [cx] ; return to caller
0000      170      ,
0000      171      csts:
0000      172      ; Returns "true" (ax<>0) if char_in has a character.
0000      173      ,
0000      174      clr     ax
0000      175      bbc     rcve_state, 0, csts_exit
0000      176      inc     ax
0000      177      csts_exit:
0000      178      ret
0000      179      ,
0000      180      char_in:

```

270061-93

```

181 ; Get a character from the software serial port
182 ;
183
184 R ; wait for character ready
185 R rcv_state,0,char_in
186 R ; set up a critical region
187 R rcv_state,#not(rxdy)
188 R al,rcv_buf
189 R ; leave the critical region
190 R
191 R $eject
192 R
193 R hso_isr
194 ; Fields the hso interrupts and performs the serialization of the data.
195 ; Note: this routine would be incorporated into the hso service strategy
196 ; for an actual system.
197
198 R at 2006h
199 R hso_isr ; Set up vector
200
201 R cseg
202 R pushf
203 R add
204 R serial_out,0 ; if character is done send a mark
205 R cmp
206 R be
207 R send_mark
208 R serial_out,#1 ; else send bit 0 of serial_out and shift
209 R shr
210 R bc
211 R send_space:
212 R hso_command,#space_command
213 R hso_time,txd_time
214 R hso_isr_exit
215 R send_mark:
216 R hso_command,#mark_command
217 R hso_time,txd_time
218 R
219 R hso_isr_exit:
220 R popf
221 R ret
222 R $eject
223 R
224 R init_receive:
225 R ; Called to prepare the serial input process to find the leading edge of
226 R ; a start bit
227 R
228 R ioc0,#00000000b ; disconnect change detector
229 R hsi_mode,#00100000b ; negative edges on HSI 2
230 R flush_fifo:
231 R iosi_save,iosl
232 R orl
233 R bbl
234 R al,hsi_status
235 R ax,hsi_time ; trash the fifo entry
236 R
237 R
238 R
239 R
240 R
241 R
242 R
243 R
244 R
245 R
246 R
247 R
248 R
249 R
250 R
251 R
252 R
253 R
254 R
255 R
256 R
257 R
258 R
259 R
260 R
261 R
262 R
263 R
264 R
265 R
266 R
267 R
268 R
269 R
270 R
271 R
272 R
273 R
274 R
275 R
276 R
277 R
278 R
279 R
280 R
281 R
282 R
283 R
284 R
285 R
286 R
287 R
288 R
289 R
290 R
291 R
292 R
293 R
294 R
295 R
296 R
297 R
298 R
299 R
300 R
301 R
302 R
303 R
304 R
305 R
306 R
307 R
308 R
309 R
310 R
311 R
312 R
313 R
314 R
315 R
316 R
317 R
318 R
319 R
320 R
321 R
322 R
323 R
324 R
325 R
326 R
327 R
328 R
329 R
330 R
331 R
332 R
333 R
334 R
335 R
336 R
337 R
338 R
339 R
340 R
341 R
342 R
343 R
344 R
345 R
346 R
347 R
348 R
349 R
350 R
351 R
352 R
353 R
354 R
355 R
356 R
357 R
358 R
359 R
360 R
361 R
362 R
363 R
364 R
365 R
366 R
367 R
368 R
369 R
370 R
371 R
372 R
373 R
374 R
375 R
376 R
377 R
378 R
379 R
380 R
381 R
382 R
383 R
384 R
385 R
386 R
387 R
388 R
389 R
390 R
391 R
392 R
393 R
394 R
395 R
396 R
397 R
398 R
399 R
400 R
401 R
402 R
403 R
404 R
405 R
406 R
407 R
408 R
409 R
410 R
411 R
412 R
413 R
414 R
415 R
416 R
417 R
418 R
419 R
420 R
421 R
422 R
423 R
424 R
425 R
426 R
427 R
428 R
429 R
430 R
431 R
432 R
433 R
434 R
435 R
436 R
437 R
438 R
439 R
440 R
441 R
442 R
443 R
444 R
445 R
446 R
447 R
448 R
449 R
450 R
451 R
452 R
453 R
454 R
455 R
456 R
457 R
458 R
459 R
460 R
461 R
462 R
463 R
464 R
465 R
466 R
467 R
468 R
469 R
470 R
471 R
472 R
473 R
474 R
475 R
476 R
477 R
478 R
479 R
480 R
481 R
482 R
483 R
484 R
485 R
486 R
487 R
488 R
489 R
490 R
491 R
492 R
493 R
494 R
495 R
496 R
497 R
498 R
499 R
500 R
501 R
502 R
503 R
504 R
505 R
506 R
507 R
508 R
509 R
510 R
511 R
512 R
513 R
514 R
515 R
516 R
517 R
518 R
519 R
520 R
521 R
522 R
523 R
524 R
525 R
526 R
527 R
528 R
529 R
530 R
531 R
532 R
533 R
534 R
535 R
536 R
537 R
538 R
539 R
540 R
541 R
542 R
543 R
544 R
545 R
546 R
547 R
548 R
549 R
550 R
551 R
552 R
553 R
554 R
555 R
556 R
557 R
558 R
559 R
560 R
561 R
562 R
563 R
564 R
565 R
566 R
567 R
568 R
569 R
570 R
571 R
572 R
573 R
574 R
575 R
576 R
577 R
578 R
579 R
580 R
581 R
582 R
583 R
584 R
585 R
586 R
587 R
588 R
589 R
590 R
591 R
592 R
593 R
594 R
595 R
596 R
597 R
598 R
599 R
600 R
601 R
602 R
603 R
604 R
605 R
606 R
607 R
608 R
609 R
610 R
611 R
612 R
613 R
614 R
615 R
616 R
617 R
618 R
619 R
620 R
621 R
622 R
623 R
624 R
625 R
626 R
627 R
628 R
629 R
630 R
631 R
632 R
633 R
634 R
635 R
636 R
637 R
638 R
639 R
640 R
641 R
642 R
643 R
644 R
645 R
646 R
647 R
648 R
649 R
650 R
651 R
652 R
653 R
654 R
655 R
656 R
657 R
658 R
659 R
660 R
661 R
662 R
663 R
664 R
665 R
666 R
667 R
668 R
669 R
670 R
671 R
672 R
673 R
674 R
675 R
676 R
677 R
678 R
679 R
680 R
681 R
682 R
683 R
684 R
685 R
686 R
687 R
688 R
689 R
690 R
691 R
692 R
693 R
694 R
695 R
696 R
697 R
698 R
699 R
700 R
701 R
702 R
703 R
704 R
705 R
706 R
707 R
708 R
709 R
710 R
711 R
712 R
713 R
714 R
715 R
716 R
717 R
718 R
719 R
720 R
721 R
722 R
723 R
724 R
725 R
726 R
727 R
728 R
729 R
730 R
731 R
732 R
733 R
734 R
735 R
736 R
737 R
738 R
739 R
740 R
741 R
742 R
743 R
744 R
745 R
746 R
747 R
748 R
749 R
750 R
751 R
752 R
753 R
754 R
755 R
756 R
757 R
758 R
759 R
760 R
761 R
762 R
763 R
764 R
765 R
766 R
767 R
768 R
769 R
770 R
771 R
772 R
773 R
774 R
775 R
776 R
777 R
778 R
779 R
780 R
781 R
782 R
783 R
784 R
785 R
786 R
787 R
788 R
789 R
790 R
791 R
792 R
793 R
794 R
795 R
796 R
797 R
798 R
799 R
800 R
801 R
802 R
803 R
804 R
805 R
806 R
807 R
808 R
809 R
810 R
811 R
812 R
813 R
814 R
815 R
816 R
817 R
818 R
819 R
820 R
821 R
822 R
823 R
824 R
825 R
826 R
827 R
828 R
829 R
830 R
831 R
832 R
833 R
834 R
835 R
836 R
837 R
838 R
839 R
840 R
841 R
842 R
843 R
844 R
845 R
846 R
847 R
848 R
849 R
850 R
851 R
852 R
853 R
854 R
855 R
856 R
857 R
858 R
859 R
860 R
861 R
862 R
863 R
864 R
865 R
866 R
867 R
868 R
869 R
870 R
871 R
872 R
873 R
874 R
875 R
876 R
877 R
878 R
879 R
880 R
881 R
882 R
883 R
884 R
885 R
886 R
887 R
888 R
889 R
890 R
891 R
892 R
893 R
894 R
895 R
896 R
897 R
898 R
899 R
900 R
901 R
902 R
903 R
904 R
905 R
906 R
907 R
908 R
909 R
910 R
911 R
912 R
913 R
914 R
915 R
916 R
917 R
918 R
919 R
920 R
921 R
922 R
923 R
924 R
925 R
926 R
927 R
928 R
929 R
930 R
931 R
932 R
933 R
934 R
935 R
936 R
937 R
938 R
939 R
940 R
941 R
942 R
943 R
944 R
945 R
946 R
947 R
948 R
949 R
950 R
951 R
952 R
953 R
954 R
955 R
956 R
957 R
958 R
959 R
960 R
961 R
962 R
963 R
964 R
965 R
966 R
967 R
968 R
969 R
970 R
971 R
972 R
973 R
974 R
975 R
976 R
977 R
978 R
979 R
980 R
981 R
982 R
983 R
984 R
985 R
986 R
987 R
988 R
989 R
990 R
991 R
992 R
993 R
994 R
995 R
996 R
997 R
998 R
999 R
1000 R

```

270061-94

00B8 717F00	R	231	andb iosi_save,#not(BOh) ; clear bit 7.
00B8 27EF		232	br flush_fifo
00B8		233	flush_fifo_done
00B8 B11015		234	ldb loc0,#00010000b ; connect HSI_2 to detector
0090 F0		235	ret
		236	
		237	
0091		238	;
		239	hsi_isr:
		240	; Fields interrupts from the HSI unit, used to detect the leading edge
		241	; of the START bit
		242	; Note: this routine would be incorporated into the HSI strategy of an actual
		243	; system.
		244	;
2004		245	cseg at 2004h
2004 9100	R	246	dcw hsi_isr ; setup the interrupt vector
		247	
0091		248	cseg
0091 F2		249	pushf
0092 C81C		250	ax
0094 B061C		251	al,hsi_status
0097 A0404	R	252	ldb sample_time,hsi_time
009A 341C15		253	ld al,4,exit_hsi
009D 3F15FD		254	bbs ioso.7,\$; wait for room in HSD holding reg
00A0 A081C	R	255	ld ax,baud_count ; send out sample command in 1/2
00A3 08011C		256	shr ax,#1 ; bit time
00A6 641C04	R	257	add sample_time,ax
00A9 B11806		258	hso_command,#sample_command
00AC C00404	R	259	st sample_time,hso_time
00AF B10015		260	ldb loc0,#00000000b ; disconnect hsi_2 from change detector
00B2		261	exit_hsi:
00B2 CC1C		262	pop ax
00B4 F3		263	popf
00B5 F0		264	ret
		265	\$elect
		266	;
00B6		267	software_timer_isr:
		268	; Fields the software timer interrupt, used to deserialize the incoming data
		269	; Note: this routine would be incorporated into the software timer strategy
		270	; in an actual system.
		271	;
200A		272	cseg at 200Ah
200A B600	R	273	dcw software_timer_isr ; setup vector
		274	
00B6		275	cseg
00B6 F2		276	pushf
00B7 901600	R	277	osi_save,osi
00B8 71FE00	R	278	orb iosi_save,#not(01h) ; clear bit 0
00B8 51FC0100	R	279	andb O.rcv_state,#0fch ; All bits except rxrdy and overrun=0
00C1 D70C		280	bne process_data

270061-95



00C3	281	process_start_bit:
00C3 350604	282	bbc hsi_status,5,start_ok
00C6 2FAE	283	call init_receive
00C8 2032	284	br software_timer_exit
00CA 910401	285	start_ok: orb rcve_state,#rip ; set receive in progress flag
00CD 2021	286	br schedule_sample
00CF	287	process_data:
00CF 3F010E	288	bbc rcve_state,7,check_stopbit
00D2 180103	289	rcve_reg,#1
00D5 350603	290	bbc hsi_status,5,datazero
00D8 918003	291	orb rcve_reg,#80h ; set the new data bit
00DB	292	datazero:
00DB 751001	293	addb rcve_state,#10h ; increment bit count
00DE 2010	294	br schedule_sample
00E0	295	check_stopbit:
00E0 3506FD	296	bbc hsi_status,5,\$; DEBUG ONLY
00E3 800302	297	ldb rcve_buf,rcve_reg
00E6 910101	298	orb rcve_state,#1rdr
00E9 710301	299	andb rcve_state,#03h ; Clear all but ready and overrun bits
00EC 2F8B	300	call init_receive
00EE 200C	301	br software_timer_exit
00F0	302	schedule_sample:
00F0 3F15FD	303	bbs ios0,7,\$; wait for holding reg empty
00F3 B11806	304	ldb hso_command,#sample_command
00F6 640804	305	add sample_time,baud_count
00F9 C00404	306	st sample_time,hso_time
00FC	307	software_timer_exit:
00FC F3	308	popf
00FD F0	309	ret
00FE	310	end
	311	
	312	
	313	
	314	
	315	
	316	
	317	
ASSEMBLY COMPLETED,		NO ERROR(S) FOUND.

APPENDIX D

MOTOR CONTROL PROGRAM

```

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: :F3: MOTCON.A96
OBJECT FILE: :F3: MOTCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE ('MOTCON.A96: Motor Control Example Program')
2
3 ;
4 ; USE WITH C-STEP or later parts
5 ;
6 ; December 20, 1984
7 $INCLUDE(DEMO96.INC)
8 $olist ; Turn listing off for include file
9 =1
10 =1
11 ; End of include file
12
13 ; Initial Values
14
15
16 min_hsil_t equ 30 ; min period for PHA edges in mode1 before mode2
17
18 min_hsi_t equ 2*min_hsil_t
19 ; min period for PHA edges in mode0 before mode1
20
21 max_hsil_t equ 3*min_hsil_t + min_hsil_t/2
22 ; max period for PHA edges in mode1 before mode0
23
24
25 HSD0_dly_period equ 110 ; delay for HSD timer 0 (timed count of pulses)
26 ; min period for 5 T2 clocks before mode 1
27
28
29 sut1_dly_period equ 250 ; delay for software timer 1
30 sut2_dly_period equ 250 ; delay for software timer 2
31
32 max_power equ Offh
33
34 max_brake equ Offh
35
36 maximum_hold equ 080H
37
38 brake_pnt equ 1200
39
40 position_pnt equ 100
41
42 velocity_pnt equ 16
43
44
45 RSEG at 024H
46
47
48
49 tmp:
50 timer_2:
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85

```



002C	tmr2_old:	dsl 1
0030	position:	dsl 1
0034	des_pos:	dsl 1
0038	pos_err:	dsl 1
003C	delta_p:	dsl 1
0040	time:	dsl 1
0044	des_time:	dsl 1
0048	time_err:	dsl 1
95	\$EJECT	
96		
97	last_time_err:	dsw 1
98	last_pos_err:	dsw 1
99	pos_delta:	dsw 1
0002	time_delta:	dsw 1
0006	last_pos:	dsw 1
000A	last_time:	dsw 1
000E	last2_time:	dsw 1
0012	boost:	dsw 1
0016	tmp1:	dsw 1
001A	out_ptr:	dsw 1
001E	offset:	dsw 1
0022	nxt_pos:	dsw 1
0026	rpwr:	dsw 1
002A	old_t2:	dsw 1
002E		
0032	direct:	dsw 1 , 1=forward, 0=reverse
0036	pwm_dir:	dsw 1
003A	hsi_so:	dsw 1
003E	last_stat:	dsw 1
0042	pwm_pwr:	dsw 1
0046	tr_col:	dsw 1 , COLLECT TRACE IF TR_COL=00
004A	main_dly:	dsw 1
004E		
0052	max_pwr:	dsw 1
0056	max_brk:	dsw 1
005A	max_hold:	dsw 1
005E	vel_pnt:	dsw 1
0062	brk_pnt:	dsw 1
0066	pos_pnt:	dsw 1
006A	HS00_dly:	dsw 1
006E	swt1_dly:	dsw 1
0072	swt2_dly:	dsw 1
0076	min_hsi:	dsw 1
007A	max_hsi1:	dsw 1
007E		
0082		
0086		
0100	dseg at 100H	

0100	mode_view:	dsb	1	
0102	count_out:	dsw	1	
0104	err_view:	dsw	1	
136				
137	count_out:	dsb	1	
138	err_view:	dsw	1	
139				
140				
141				
142	\$reject			
143				
144				
145				
146				
147				
148				
149				
150				
151				
152				
153				
154				
155				
156				
157	cseg	at	2000H	
158				
159				
160				
161				
162				
163				
164				
165				
166				
167	atod_done_int:			
168	hsi_o_int:			
169	ser_port_int:			
170	external_int:			
171				
172	cseg	at	2080H	
173				
174	init:	ld	sp, #0FOH	
175		ldb	pwm_control, #OFFH	
176				
177		clrb	direct	
178		ld	tmp1, #6000	
179	delay:	dec	tmp1	
180		djnz	direct, \$	
181		cmp	tmp1, zero	
182		jgt	delay	
183				
184		ldb	port1, #OFFH	
185		ldb	port2, #OFFH	
2000				
2000 0022				
2002 1020				
2004 0424				
2006 8022				
2008 1020				
200A 2022				
200C 1020				
200E 1020				
2010				
2010				
2010				
2010				
2080				
2080 A1F00018				
2084 B1FF17				
2087 1168				
2089 A170175C				
208D 055C				
208F E068FD				
2092 88005C				
2095 D2F6				
2097 B1FF0F				
209A B1FF10				

270061-99

209D B12516	186	ldb	IOC1.#00100101B ; Disable HSD 4,HSD 5, HSI_INT=first, ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
20A0 71FC0F	187		
20A3 B19903	188	andb	Port1.#11111100B ; clear P1.O.1 (set mode 0)
20A6 B15715	189	ldb	HSI_mode.#10011001B ; set hsi.1,3 --, hsi.0,2 +
	190	ldb	IOC0.#01010111B ; Enable all hsi
	191		; T2 CLOCK=T2CLK, T2RST=T2RST
	192		; Clear timer2
	193		
	194	\$eject	
	195	ld	zero.hsi_time
20A9 A00400	196	clr	time
20AC 0140	197	clr	time+2
20AE 0142	198	clr	timer_2
20B0 012B	199	clr	timer_2+2
20B2 012A	200	clr	position
20B4 0130	201	clr	position+2
20B6 0132	202	clr	last_pos
20B8 0154	203	clr	last_pos
20BA 0134	204	clr	des_pos
20BC 0136	205	clr	des_pos+2
20BE 0144	206	clr	des_time
20C0 0146	207	clr	des_time+2
20C2 A00A56	208	ld	last1_time,Timer1
20C5 49000B55B	209	sub	last2_time,last1_time,#BOOH
20CA 116D	210	clrb	ios1_bak
20CC 1109	211	clrb	int_pending
20CE A1F0015E	212	ld	out_ptr.#iFOH
20D2 A13C00B2	213	ld	min_hsil.#min_hsil_t
20D6 A11E00B4	214	ld	min_hsil.#min_hsil_t
20DA A16900B6	215	ld	max_hsil.#max_hsil_t
20DE A16E007C	216	ld	HSD0_dly,#HSD0_dly_period
20E2 A1FA007E	217	ld	swt1_dly,#swt1_dly_period
20E6 A1FA00B0	218	ld	swt2_dly,#(swt2_dly_period)
20EA A1FF0070	219	ld	max_pwr.#max_power
20EE A1FF0072	220	ld	max_brk.#max_brake
20F2 A1800074	221	ld	max_hold.#maximum_hold
20F6 A180047B	222	ld	brk_pnt.#brake_pnt
20FA A164007A	223	ld	pos_pnt.#position_pnt
20FE A1100076	224	ld	vel_pnt.#velocity_pnt
2102 A1002962	225	ld	next_pos.#pos_table
2106 B0006C	226	ldb	pwm_pwr.zero
2109 B10169	227	ldb	pwm_dir.#01h ; FORWARD
	228		
210C B12D0B	229	ldb	int_mask.#00101101B ; Enable tmr_ovf, hsi, swt, HSD.interrupts
210F B13006	230	ldb	hso_command.#30H ; set HSD_0
2112 447C0A04	231	add	hso_time,timer1,HSD0_dly
2116 FD	232	nop	
2117 FD	233	NOP	
211B B13906	234	ldb	hso_command.#39H ; set swt_1
211B 447E0A04	235	add	hso_time,timer1,swt1_dly

270061-A0

```

211F FD      nop
2120 FD      nop
2121 B13A06   hso_command,#3AH      , set swt_2
2124 44800A04 hso_time,timer1,swt2_dly
240         time,timer1
241         tmr2_old,timer2
242         ld
243         ei
244         br      main_prog
245
246         $reject
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285

2200
2200
2200 F2
2201 90166D
2204 356D05
2207 0742
2209 71DF6D
220C
220C F3
220D F0

2220
2220
2220 F2
2221 90166D
2224 306D03
2227 71FE6D
222A 316D06
222D 71FD6D

236         nop
237         nop
238         ldb
239         add
240
241         ld
242         ld
243         ei
244         br      main_prog
245
246         $reject
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285

CSEG AT 2200H
timer_ovf_int
pushf
orb      ios1_bak,IOS1
chk_t1:  jbc      ios1_bak,5,tmr_int_done
inc      timer2
andb     ios1_bak,#1101111B      , clear bit 5
tmr_int_done
popf
ret
; End of timer interrupt routine

SOFTWARE TIMER INTERRUPT SERVICE ROUTINE
CSEG AT 2220H
soft_tmr_int
pushf
orb      ios1_bak,IOS1
chk_swto:
jbc      ios1_bak,0,chk_sut1
andb     ios1_bak,#1111110B      ; Clear bit 0 - end swto
call     swto_expired
chk_sut1:
jbc      ios1_bak,1,chk_sut2
andb     ios1_bak,#1111101B      ; Clear bit 1

```

270061-A1

2230 EFC0D3	286	call	swt1_expired	
2233 326D06	287	chk_sw2:		
2236 71FB6D	288	jbc	ios1_bak,2,chk_sw2	
2239 EF4401	289	andb	ios1_bak,#1111011B	; Clear bit 2
223C 346D03	290	call	swt2_expired	
223F 71F76D	291	chk_sw3:		
	292	jbc	ios1_bak,4,swt_int_done	
	293	andb	ios1_bak,#1110111B	; Clear bit 3
	294	call	swt3_expired	
	295	swt_int_done:		
2242 F3	296	popf		
2243 F0	297	ret		
	298			; END OF SOFTWARE TIMER INTERRUPT ROUTINE
	299			
	300	\$rept		
	301			
	302			
	303			SOFTWARE TIMER ROUTINE ()
	304			NOW USING HSO 0 TO TRIGGER
	305			
	306			
2280	307	CSEG AT 2280H		
	308			
2280	309	hso_exec_int.		; Check mode -- Update position in mode 2
	310			
2280 F2	311	PUSHF		
2281 B13006	312	lbb	HSO_COMMAND,#30H	
2284 447C0A04	313	add	HSO_TIME,TIMER1,HSO0_dly	
	314			
228B 91200F	315	orb	port1,#00100000B	; set P1.5
228B A0C2B	316	ld	Timer_2,TIMER2	
228E 390F1B	317	jbs	Port1,1,in_mode2	
	318			
2291 48662B5C	319	in_mode1:	tmp1,Timer_2,old_t2	; Check count difference in tmp1
2295 8902005C	320	sub	tmp1,#2	
2299 D94C	321	cnp	end_sw0	
229B	322	jh		
229B 300F49	323	set_mode0:		
229E 71FC0F	324	jbc	Port1,0,end_sw0	; if already in mode 0
22A1 B15515	325	andb	Port1,#1111100B	; Clear P1.0, P1.1 (set mode 0)
22A4 B006B	326	lbb	IDCO,#01010101B	; enable all HSI
22A7 203E	327	lbb	last_stat,zero	
	328	br	end_sw0	
	329			
22A9 4B2C2B3C	330	in_mode2:		
22AD A02B2C	331	sub	delta_p,timer_2,tmr2_old	; get timer2 count difference
	332	ld	tmr2_old,timer_2	
22B0 306B0B	333	jbc	direct,0,in_rev	
	334			
	335			

270061-A2

22B3 643C30	336	in_fwd: add	position, delta_p
22B6 A40032	337	addc	position+2, zero
22B9 2006	338	br	chk_mode
22BB 683C30	339		
22BE A80032	340	in_rev: sub	position, delta_p
	341	subc	position+2, zero
22C1	342	chk_mode:	
22C1 4B62B5C	343	sub	tmp1, Timer_2, old_t2
22C5 8905005C	344	sub	
22C9 D21C	345	cmp	tmp1, #5 ; Check count difference in tmp1
	346	jgt	; set mode1 if count is too low
	347		; count <= 5
22CB	348	set_model:	
22CB 71FDOF	349	andb	Port1, #1111101B ; Clear P1.1, set P1.0 (set mode 1)
22CE 91010F	350	orb	Port1, #00000001B
22D1 B10515	351	ldb	IDCO, #00000101B ; enable HSI 0 and 1
22D4 A00400	352	ld	zero, HSI_TIME
22D7 4B840A56	353	sub	last1_time, Timer1, min_hsi1
	354		; set up so (time-last2_time)>min_hsi1 on next HSI
	355	\$EJECT	
	356		
22DB	357	clr_hsi:	
22DB A00400	358	ld	ZERO, HSI_TIME
22DE 717F6D	359	andb	iosi_bak, #0111111B ; clear bit 7
22E1 90166D	360	orb	iosi_bak, ios1
22E4 3F6DF4	361	jbs	iosi_bak, 7, clr_hsi ; If hsi is triggered then clear hsi
	362		
22E7	363	end_swt0:	
22E7 A02B66	364	ld	old_t2, TIMER_2
22EA 71DF0F	365	andb	port1, #1101111B ; clear P1.5
22ED F3	366	POPF	
22EE F0	367	ret	
	368		
	369		
	370		
	371		
	372		
	373		
	374		
23B0	375	CSEG AT 23B0H	
	376		
23B0	377	swt2_expired:	
23B0 F2	378	pushf	
23B1 B13A06	379	ldb	hso_command, #3AH ; set swt_2
23B4 44B00A04	380	add	hso_time, timer1, swt2_dly
	381		
23B8 91040F	382	orb	port1, #00000100B ; set port 1.2
23B8 89FF075E	383	cmp	out_ptr, #7ffh
23BF D104	384	bnh	pulsing
2391 A1F0015E	385	ld	out_ptr, #1f0H

270061-A3

```

2395      386      pulsing: jbc      tr_col,0,swt2_done
2396      387
2397      388
2398      389      position+2,[out_ptr]+      ; position high, position low
2399      390      st
2400      391      st
2401      392      direct,[out_ptr]+
2402      393      pwm_pwm,[out_ptr]+
2403      394
2404      395      ; store 8 bytes externally
2405      396
2406      397      swt2_done:
2407      398      tmp1,timer1,last1_time
2408      399      cmp tmp1,#1800H
2409      400      jnh      ; keep (Timer1-last1_time)<2000H
2410      401      add      last1_time,#1000H
2411      402      swt2_ret:
2412      403      andb      port1,#11111011B      ; clear port1.2
2413      404      popf
2414      405      ret
2415      406
2416      407      $EJECT
2417      408      ;
2418      409      ;
2419      410      ;
2420      411      ;
2421      412      ;
2422      413      ;
2423      414      ; This routine keeps track of the current time and position of the motor.
2424      415      ; The upper word of information is provided by the timer overflow routine.
2425      416
2426      417      CSEG AT 2400H
2427      418      now_mode_1: br      in_mode_1      ; used to save execution time for
2428      419      no_int1: br      no_int      ; worst case loop
2429      420
2430      421      hsi_data_int: pushf
2431      422      orb      port1,#01000000B      ; set P1.6
2432      423      andb      ios1_bak,#01111111B      ; Clear ios1_bak.7
2433      424      orb      ios1_bak,ios1
2434      425      jbc      ios1_bak,7,no_int1      ; If hsi is not triggered then
2435      426      ; jump to no_int
2436      427      get_values:
2437      428      ld      timer_2,TIMER2
2438      429      andb      hsi_s0,HSI_STATUS,#01010101B
2439      430      ld      time, HSI_TIME
2440      431
2441      432      jbs      port1,0,now_mode_1      ; jump if in mode 1
2442      433
2443      434      In_mode_0:
2444      435      jbs      hsi_s0,0,a_rise

```

2421 3A6A2C	436		hsl_s0.2, a_fall	
2424 3C6A4D	437	jbs	hsl_s0.4, b_rise	
2427 3E6A5A	438	jbs	hsl_s0.6, b_fall	
242A 2094	439	br	no_cnt	
242C A05658	440		last2_time, last1_time	
242F A04056	441	a_rise: ld	last1_time, time	
2432 685840	442	ld	time, last2_time	
2435 888240	443	sub	time, min_hsl	
2438 D906	444	cmp	tst_statf	
243A 91010F	445	jh		
243D 810515	446	; set model-	Port1, #000000001B	; Set P1.0 (in mode 1)
2440	447	orb	IDCO, #00000101B	; Enable HSI 0 and 1
2440 3E6B58	448	ldb		
2443 3C6B67	449	tst_statf:		
2446 3A6B50	450	jbs	last_stat, 6, going_fud	
2449 98006B	451	jbs	last_stat, 4, going_rev	
244C DF46	452	jbs	last_stat, 2, change_dir	
244E 27B2	453	cmpb	last_stat, zero	
	454	je	first_time	; first time in mode0
	455	br	no_int1	
2450 A05658	456		last2_time, last1_time	
2453 A04056	457	a_fall: ld	last1_time, time	
2456 685840	458	ld	time, last2_time	
2459 888240	459	sub	time, min_hsl	
245C D906	460	cmp	tst_statf	
	461	jh		
245E 91010F	462	; set model-	Port1, #000000001B	; Set P1.0 (in mode 1)
2461 B10515	463	orb	IDCO, #00000101B	; Enable HSI 0 and 1
	464	ldb		
2464	465	\$EJECT		
2464 3C6B37	466	tst_statf:		
2467 3E6B43	467	jbs	last_stat, 4, going_fud	
246A 3B6B2C	468	jbs	last_stat, 6, going_rev	
246D 98006B	469	jbs	last_stat, 0, change_dir	
2470 DF22	470	cmpb	last_stat, zero	
2472 2057	471	je	first_time	; first time in mode0
	472	br	no_int	
2474 3B6B27	473		last_stat, 0, going_fud	
2477 3A6B33	474	b_rise: jbs	last_stat, 2, going_rev	
247A 3E6B1C	475	jbs	last_stat, 6, change_dir	
247D 98006B	476	jbs	last_stat, 4, change_dir	
2480 DF12	477	cmpb	last_stat, zero	
2482 2047	478	je	first_time	; first time in mode0
	479	br	no_int	
2484 3A6B17	480		last_stat, 2, going_fud	
2487 3B6B23	481	b_fall: jbs	last_stat, 0, going_rev	
248A 3C6B0C	482	jbs	last_stat, 6, change_dir	
248D 98006B	483	jbs	last_stat, 4, change_dir	
2490 DF02	484	cmpb	last_stat, zero	
	485	je	first_time	; first time in mode0

270061-A5

2492 2037	486		br	no_int	
2494	487				
2494 C46B6A	488	first_time:	stb	hsi_s0.last_stat	
2497 2072	489		br	done_chk	, add delta position
	491				
2499	492	change_dir:		direct	
2499 1268	493		notb	direct	
249B 306B0F	494	no_inc:	jbc	direct,0,going_rev	
	496				
249E 914010	497	going_fwd:		PORT2,#01000000B	; set P2.6
24A1 B10168	498		orb	direct,#01	; direction = forward
24A4 65010030	499		ldb	position,#01	
24AB A40032	500		add	position+2,zero	
24AB 200D	501		addc	st_stat	
24AD	502		br		
24AD 718F10	503	going_rev:		PORT2,#10111111B	; clear P2.6
24B0 B10068	504		andb	direct,#00	; direction = reverse
24B3 69010030	505		ldb	position,#01	
24B7 AB0032	506		sub	position+2,zero	
	507		subc		
24BA	508				
24BA C46B6A	509	st_stat:	stb	hsi_s0.last_stat	
24BD	510	load_last:			
24BD A02B2C	511		ld	tmr2_oldd,timer_2	
24C0 717F6D	512	no_cnt:	andb	ios1_bak,#01111111B	; clr bit 7
24C3 90166D	513		orb	ios1_bak,ios1	
24C6 376D02	514		jbc	ios1_bak,7,no_int	
24C9 2746	515	again:	br	get_values	
	516				
24CB 718F0F	517	no_int:	andb	port1,#10111111B	; Clear P1.6
24CE F3	518		popf		
24CF F0	519		ret		
	520				; end of hsi_data_interrupt routine
	521				; Routine for mode 1 follows and then returns to "load_last"
	522	\$EJECT			
	523				
	524				
24D0	525	In_mode_1:			, mode 1 HSI routine
	526				
24D0 51506A5C	527		andb	tmp1,hsi_s0,#01010000B	
24D4 D7EA	528		jne	no_cnt	
24D6	529	cmp_time:			; Procedure which sets mode 1 also
	530				; sets times to pass the tests
24D6 A05658	531		ld	last2_time,last1_time	
24D9 A04056	532		ld	last1_time,time	
	533				
24DC 4858405C	534	cmp1:	sub	tmp1,time,last2_time	
24E0 88845C	535		cmp	tmp1,min_hsi1	

270061-A6

```

24E3 D914          536          Jh          check_max_time
24E5          537          set_mode_2:
24E5 91020F          538          orb
24E8 B10015          539          ldb
24E8 A00400          540          mt_hsi: ld
24EE 717F6D          541          andb
24F1 90166D          542          ios1_bak, #0111111B
24F4 3F6DF4          543          orb
24F7 2012          544          ios1_bak, ios1
24F9          545          done_chk
24F9 4858405C          546          br
24FD 88665C          547          check_max_time:
2500 D109          548          tmp1, time, last2_time
2502          549          cmp
2502 71FC0F          550          jnh
2503 B15515          551          done_chk
2508 B0006B          552          set_mode_0:
2508 482C283C          553          andb
250F 30680B          554          ldb
2512 643C30          555          last_stat, zero
2515 A40032          556          done_chk:
2518 27A3          557          delta_p, timer_2, tmr2_old
251A          558          direct, 0, add_rev
251D A80032          559          add_fud:
2520 279B          560          add
2520          561          position, delta_p
2520          562          position+2, zero
2520          563          load_last5
2520          564          add_rev:
2520          565          sub
2520          566          position, delta_p
2520          567          position+2, zero
2520          568          load_last5
2520          569          $reject
2520          570          ..... SOFTWARE_TIMER_ROUTINE_1 .....
2520          571          .....
2520          572          .....
2520          573          .....
2520          574          CSEG AT 2600H
2520          575          swt1_expired:
2520          576          pushf
2520          577          orb
2520          578          port1, #10000000B
2520          579          int_mask, #00001101B
2520          580          enable HSI, Tcvf, HSO
2520          581          ldb
2520          582          HSO_COMMAND, #39H
2520          583          HSO_TIME, TIMER1, swt1_dly
2520          584          add
2520          585

```

270061-A7

260E A0464A	586	ld	time_err+2,des_time+2	, Calculate time & position error
2611 A0363A	587	ld	pos_err+2,des_pos+2	
2614 4B404448	588	sub	time_err,des_time,time	; values are set
2618 4B424A	589	subc	time_err+2,time+2	
261B 4B30343B	590	sub	pos_err,des_pos,position	
261F 4B323A	591	subc	pos_err+2,position+2	
2622 FB	592	EI		
2623 4B484C52	593			
2627 A04B4C	594	sub	time_delta,last_time_err,time_err	
262A 4B384E50	595	ld	last_time_err,time_err	
262E A03B4E	596	sub	pos_delta,last_pos_err,pos_err	
	597	ld	last_pos_err,pos_err	
	598			
	599			
	600			
	601			
	602	Time_err = Desired time to finish - current time		
	603	Pos_err = Desired position to finish - current position		
	604	Pos_delta = Last position error - Current position error		
	605	Time_delta = Last time error - Current time error		
	606	note that errors should get smaller so deltas will be		
	607	positive for forward motion (time is always forward)		
	608			
	609			
2631 8B003A	610	chk_dir:	pos_err+2,zero	
2634 D60D	611	cmp	go_forward	
	612	jge		
	613			
2636 033B	614	go_backward:	pos_err	; Pos_err = ABS VAL (pos_err)
2638 B10069	615	neg	pwm_dir,#00h	
263B 89FFF3A	616	ldb	pos_err+2,#0fffh	
263F D70A	617	cmp	ld_max	
2641 200D	618	jne	chk_brk	
	619	br		
	620			
2643 B10169	621	go_forward:	pwm_dir,#01h	
2646 8B003A	622	ldb	pos_err+2,zero	
2649 DF05	623	cmp	chk_brk	
	624	je		
	625	\$EJECT		
	626			
264B B0706C	627	ld_max:	pwm_pwr,max_pwr	
264E 2031	628	br	chk_sanity	
	629			
2650 8B7A3B	630	Chk_brk:	Position_Error now = ABS(pos_err)	
2653 D11E	631	cmp	pos_err,pos_pnt	
2655 8B7B3B	632	jnh	hold_position	; position_error<position_control_point
	633	cmp	pos_err,brk_pnt	

270061-A8

270061-A9

26AB FA	684	p2bkwd: DI	port2,#01111111B	; clear P2.7
26AC 717F10	685	andb	pwm_control,rpwr	
26AF B06417	686	lbb		
26B2 FB	687	EI		
26B3 2008	688	br		
26B5 FA	689	p2fwd: DI	port2,#10000000B	; set P2.7
26B6 91B010	690	orb	pwm_control,rpwr	
26B9 B06417	691	lbb		
26BC FB	692	EI		
26BD	693			
26BD B8004A	694	purset:	time_err+2,zero	; do pos_table when err is negative
26C0 D225	695	cmp	end_p	
	696	jgt		
	697	br	end_p	
	698	;;		
26C2 89202962	699	cmp	nxt_pos,*(32+pos_table)	; jump if lower
26C6 DE06	700	jlt	get_vals	
26C8 A1002962	701	ld	nxt_pos,pos_table	
26CC 0142	702	clr	time+2	
26CE	703	get_vals:		
26CE A26334	704	ld	des_pos,[nxt_pos]+	
26D1 A26336	705	ld	des_pos+2,[nxt_pos]+	
26D4 A26346	706	ld	des_time+2,[nxt_pos]+	
26D7 A26370	707	ld	max_pwr,[nxt_pos]+	
26DA A07072	708	ld	max_brk,max_pwr	
26DD 646034	709	ld	des_pos,offset	
26E0 A40036	710	.add	des_pos+2,zero	
26E3 4B30344E	711	addc	last_pos_err,des_pos,position	
	712	sub		
26E7 717F0F	713			
	714	end_p: andb	port1,#01111111B	; clear P1.7
26EA F3	715			
26EB F0	716	popf		
	717	ret		
	718			
	719	*EJECT		
	720			
	721			
	722			
	723			
	724			
	725			
2800	726			
	727			
	728			
2800 90166D	729	CSEG at 2800H		
2803 366D09	730	MAIN_PROG:	iosl_bak,iosl	
2806 71BF6D	731	orb	iosl_bak,6,control	
2809 95100F	732	jbc	iosl_bak,#10111111B	; clear iosl_bak.6
280C EFF5FB	733	andb	iosl_bak,#10111111B	; Compl Bit P1.4
		xorb	Port1,#00010000B	; prevent lockup
		call	HSI_DATA_INT	

270061-B0

```

280F 912D0B          control: 734      orb      int_mask,#00101101B      ; enable hsi, hso, swt, tovf interrupts
280F 912D0B          nop      735
2812 FD            nop      736
2813 FD            nop      737
2814 FD            nop      738
2815 E06FFD        d jnz     739      main_dly,$
2818 FD            nop      740
2819 95080F        xorb     741      port1,#00001000B      ; compliment p1.3
281C 27E2          BR       742      MAIN_PROG
2900              CSEQ AT 2900H
2900              pos_table:
2900 00000000        dcl      748      00000000H      ; position 0
2904 20080000        dcw     749      0020H, 0080H      ; next time, power
2908 00C00000        dcl      750      0000C000H      ; position 1
290C 40040000        dcw     751      0040H, 0040H      ; next time, power
2910 00000000        dcl      752      00000000H      ; position 2
2914 600C0000        dcw     753      0060H, 00C0H      ; next time, power
2918 0080FFFF        dcl      754      0080H, 0080H      ; position 3
291C 80080000        dcw     755      0080H, 0080H      ; next time, power
2920 00080000        dcl      756      00008000H      ; position 4
2924 58080000        dcw     757      0058H, 0080H      ; next time, power
2928 00300000        dcl      758      00003000H      ; position 5
292C 7000FF00        dcw     759      0070H, 00FFH      ; next time, power
2930 00000000        dcl      760      00000000H      ; position 6
2934 9000F000        dcw     761      0090H, 00F0H      ; next time, power
2938 00000000        dcl      762      00000000H      ; position 7
293C 9100F000        dcw     763      0091H, 00F0H      ; next time, power
2940              END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-B1



INTEL CORPORATION, 2200 Mission College Blvd., Santa Clara, CA 95052; Tel. (408) 765-8080

INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511

