AP-466

APPLICATION NOTE

Using the 80C196KB

ROBIN SHEER
EMD APPLICATIONS

November 1991

Order Number: 272116-001

Using the 80C196KB

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

1.0 INTRODUCTION

The MCS®-96 family members are all high performance microcontrollers with a 16-bit CPU and at least 230 bytes of on chip RAM. The Intel MCS-96 family of 16-bit embedded controllers easily handles high speed calculations and fast input/output (I/O) operations. Typical applications using the MCS-96 products include closed-loop control and mid-range digital signal processing. Modems, motor control system, printers, engine control system, photocopiers, anti-lock brakes, air conditioner control systems, disk drives and medical instrumentation all use MCS-96 products.

The 80C196KB is a CHMOS member of the MCS-96 family. All of the MCS-96 components share a common instruction set and architecture. However, the CHMOS components have enhancements to provide higher performance with lower power consumption. To further decrease power usage, idle and power-down modes are available on these devices. The 80C196KB contains a dedicated I/O subsystem and can perform 16-bit arithmetic instructions including multiply and divide operations.

This application note briefly describes the 80C196KB, and provides software examples using its key features. For further information on the 80C196KB and its use consult the sources listed in the bibliography. Figure 1-1 shows a block diagram of the 80C196KB. Included in this application note are descriptions of the CPU and architecture, the interrupt structure and the peripherals. These peripherals include a Pulse Width Modulation output, an A/D Converter, a Serial Port and High Speed I/O Unit with two 16-bit timer/counters.

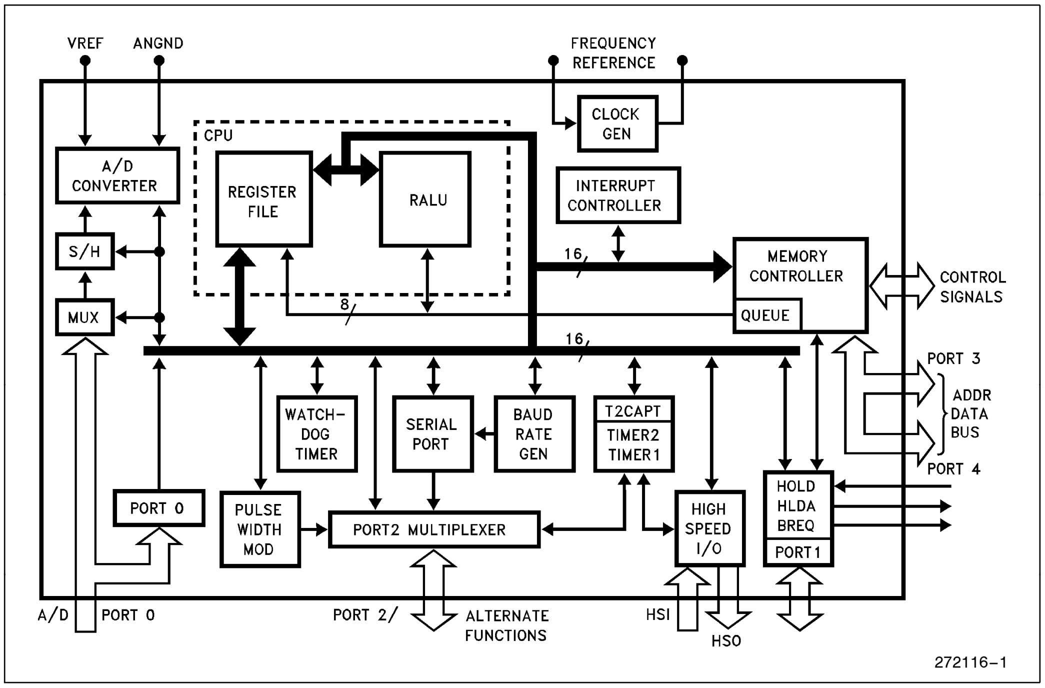


Figure 1-1. 80C196KB Block Diagram

2.0 THE CPU

The major components of the 80C196KB CPU are the Register File and the Register/Arithmetic Logic Unit (RALU). The Register File contains 256 internal register locations (00H through 0FFH), all of which remain alive during power-down mode. Locations 00H through 17H are the I/O control registers or Special Function Registers (SFRs). Locations 18H and 19H contain the stack pointer, which can serve as general purpose RAM when not performing stack operations. The remaining 230 bytes serve as general purpose RAM, accessible as bytes, words or double-words.

Calculations performed by the 80C196KB take place in the RALU. The RALU shown in Figure 2-1 contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. The RALU operates directly on the Register File, thus eliminating accumulator bottleneck and providing for direct control of I/O operations through the SFRs.

The SFRs control all the 80C196KB peripheral devices except Ports 3 and 4. Figure 2-2 shows the layout of these registers. Three SFR windows exist on the 80C196KB. The value in the Window Select Register (WSR) determines the SFR window; WSR = 0 selects Window 0 and WSR = 15 selects Window 15. Window 0 consists of 24 SFRs. Some of these registers serve one function when read and another function when written. The read-only registers in Window 0 become write-only registers in Window 15; and the write-only registers in Window 0 become read-only registers in Window 15. Figure 2-3 contains descriptions of the SFRs.

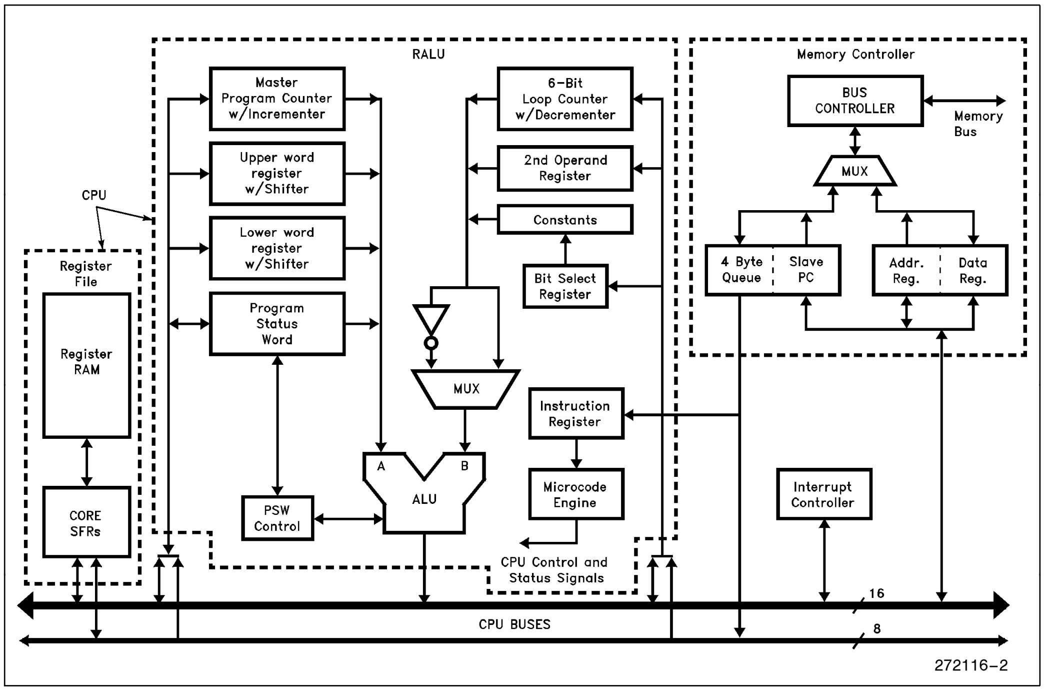


Figure 2-1. Block Diagram of the Register File, RALU, Interrupt Controller and Memory Controller

	WSR = 0		WSR = 15	
	WHEN READ	WHEN WRITTEN	WHEN READ	WHEN WRITTEN
19H	STACK POINTER	STACK POINTER	STACK POINTER	STACK POINTER
18H				
17H	*IOS2	PWM_CONTROL	PWM_CONTROL	*IOS2
16H	IOS1	IOC1	IOC1	IOS1
15H	IOS0	IOC0	IOC0	IOS0
14H	*WSR	*WSR	*WSR	*WSR
13H	*INT_MASK1	*INT_MASK1	*INT_MASK1	*INT_MASK1
12H	*INT_PEND1	*INT_PEND1	*INT_PEND1	*INT_PEND1
11H	*SP_STAT	*SP_CON	*SP_CON	*SP_STAT
10H	PORT2	PORT2	RESERVED**	RESERVED**
0FH	PORT1	PORT1	RESERVED**	RESERVED**
0EH	PORT0	BAUD RATE	RESERVED**	RESERVED**
0DH	TIMER2(HI)	TIMER2(HI)	T2CAPTURE(HI)	T2CAPTURE(HI)
0CH	TIMER2(LO)	TIMER2(LO)	T2CAPTURE(LO)	T2CAPTURE(LO)
0BH	TIMER1(HI)	*IOC2	*IOC2	TIMER1(HI)
0AH	TIMER1(LO)	WATCHDOG	WATCHDOG	TIMER1(LO)
09H	INT_PEND	INT_PEND	INT_PEND	INT_PEND
08H	INT_MASK	INT_MASK	INT_MASK	INT_MASK
07H	SBUF(RX)	SBUF(TX)	SBUF(TX)	SBUF(RX)
06H	HSI_STATUS	HSO_COMMAND	HSO_COMMAND	HSI_STATUS
05H	HSI_TIME(HI)	HSO_TIME(HI)	HSO_TIME(HI)	HSI_TIME(HI)
04H	HSI_TIME(LO)	HSO_TIME(LO)	HSO_TIME(LO)	HSI_TIME(LO)
03H	AD_RESULT(HI)	HSI_MODE	HSI_MODE	AD_RESULT(HI)
02H	AD_RESULT(LO)	AD_COMMAND	AD_COMMAND	AD_RESULT(LO)
01H	ZERO_REG(HI)	ZERO_REG(HI)	ZERO_REG(HI)	ZERO_REG(HI)
00H	ZERO_REG(LO)	ZERO_REG(LO)	ZERO_REG(LO)	ZERO_REG(LO)

NOTES:
*New or changed register function from 8096BH
**Reserved registers should not be written or read

Figure 2-2. Special Function Registers

Register	Description
ZERO__REG	Zero Register - Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.
AD__RESULT	A/D Result Hi/Low - Low and high order results of the A/D converter
AD__COMMAND	A/D Command Register - Controls the A/D
HSI__MODE	HSI Mode Register - Sets the mode of the High Speed Input unit.
HSI__TIME	HSI Time Hi/Lo - Contains the time at which the High Speed Input unit was triggered.
HSO__TIME	HSO Time Hi/Lo - Sets the time or count for the High Speed Output to execute the command in the Command Register.
HSO__COMMAND	HSO Command Register - Determines what will happen at the time loaded into the HSO Time registers.
HSI__STATUS	HSI Status Registers - Indicates which HSI pins were detected at the time in the HSI Time registers and the current state of the pins. In Window 15 - Writes to pin detected bits, but not current state bits.
SBUF(TX)	Transmit buffer for the serial port, holds contents to be outputted. Last written value is readable in Window 15.
SBUF(RX)	Receive buffer for the serial port, holds the byte just received by the serial port. Writable in Window 15.
INT__MASK	Interrupt Mask Register - Enables or disables the individual interrupts.
INT__PEND	Interrupt Pending Register - Indicates that an interrupt signal has occurred on one of the sources and has not been serviced. (also INT__PENDING)
WATCHDOG	Watchdog Timer Register - Written periodically to hold off automatic reset every 64K state times. Returns upper byte of WDT counter in Window 15.
TIMER1	Timer 1 Hi/Lo - Timer1 high and low bytes.
TIMER2	Timer 2 Hi/Lo - Timer2 high and low bytes.
IOPORT0	Port 0 Register - Levels on pins of Port 0. Reserved in Window 15.
BAUD__RATE	Register which determines the baud rate, this register is loaded sequentially. Reserved in Window 15.
IOPORT1	Port 1 Register - Used to read or write to Port 1. Reserved in Window 15
IOPORT2	Port 2 Register - Used to read or write to Port 2. Reserved in Window 15
SP__STAT	Serial Port Status - Indicates the status of the serial port.
SP__CON	Serial Port Control - Used to set the mode of the serial port.
IOS0	I/O Status Register 0 - Contains information on the HSO status. Writes to HSO pins in Window 15.
IOS1	I/O Status Register 1 - Contains information on the status of the timers and of the HSI.
IOC0	I/O Control Register 0 - Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.
IOC1	I/O Control Register 1 - Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.
PWM__CONTROL	Pulse Width Modulation Control Register - Sets the duration of the PWM pulse.
INT__PEND1	Interrupt Pending register for the 8 new interrupt vectors (also INT__PENDING1)
INT__MASK1	Interrupt Mask register for the 8 new interrupt vectors
IOC2	I/O Control Register 2 - Controls new 80C196KB features
IOS2	I/O Status Register 2 - Contains information on HSO events
WSR	Window Select Register - Selects register window

Figure 2-3. Special Function Register Descriptions

3.0 THE ARCHITECTURE

The 80C196KB supports 106 instructions. This instruction set includes bit operations, byte operations, word operations, double-word operations (unsigned 32-bit) long operations (signed 32-bit), flag manipulations as well as jump and call instructions. All the standard logical and arithmetic instructions function as both byte and word operations. The Jump Bit Set and Jump Bit Clear instructions can operate on any of the SFRs or bytes in the register file. These fast bit manipulations allow for rapid I/O functions.

Byte and word operations make-up most of the 80C196KB instruction set. The assembly language for the 80C196KB (ASM-96) uses a "B" suffix on a mnemonic for a byte operation, otherwise the mnemonic refers to a word operation. One, two or three operand forms exist for many of the instructions.

A one operand instruction has the form:

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

A two operand instruction has the form:

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

A three operand instruction has the form:

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

Long and double-word operations include shifts, normalize, multiply and divide. The divide instruction functions as a 32-bit by 16-bit divide that generates a 16-bit quotient and 16-bit remainder. The word multiply operates as a 16-bit by 16-bit multiply with a 32-bit result. Both operations can function in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 3.0 μs (at 16 MHz) for divide and 1.75 μs (at 16 MHz) for multiply. The normalize instruction and sticky bit flag provide hardware support for the software floating point package (FPAL-96).

3.1 Addressing Modes

The 80C196KB instruction set supports the following addressing modes: register-direct, indirect, indirect with auto-increment, immediate, short-indexed and long-indexed. These modes increase the flexibility and overall execution speed of the 80C196KB. Each instruction uses at least one of the addressing modes. These modes and formats are shown in Figure 3-1.

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 = Instruction Mnemonic
Dest = Destination Register
Src1, Src2 = Source Registers
addr = Word register used in computing the address of an operand
offs = Offset used in computing the address of an operand

Figure 3-1. Instruction Format

The register-direct and immediate addressing modes execute faster than the other addressing modes. The register-direct addressing mode provides access to the addresses in the register file and the SFRs. The indexed modes provide for direct access to the remainder of the 64K address space. Immediate addressing uses 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. The indirect auto-increment mode increments a word address by one after a byte operation and two after a word operation. This addressing mode provides easy access into look-up tables.

The long-indexed addressing mode provides direct access to any of the locations in the 64K address space. This mode forms the address of the operand by adding a 16-bit 2's complement value to the contents of a word register. Indexing with the zero register allows "direct" addressing to any location. The short-indexed addressing mode forms the address of the operand by adding an 8-bit 2's complement value to the contents of a word register.

The multiple addressing modes of the 80C196KB 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. Table 3-1 lists the mnemonics and their functions. The MCS-96 Macro Assembler Users Guide, listed in the bibliography, contains additional information on 80C196KB assembly language.

Table 3-1. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	–	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	–	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	–	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	–	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	–	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	–	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	–	
MUL/MULU	2	D,D + 2 ← D × A	–	–	–	–	–	–	2
MUL/MULU	3	D,D + 2 ← B × A	–	–	–	–	–	–	2
MULB/MULUB	2	D,D + 1 ← D × A	–	–	–	–	–	–	3
MULB/MULUB	3	D,D + 1 ← B × A	–	–	–	–	–	–	3
DIVU	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	2
DIVUB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	3
DIV	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	
DIVB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	
AND/ANDB	2	D ← D AND A	✓	✓	0	0	–	–	
AND/ANDB	3	D ← B AND A	✓	✓	0	0	–	–	
OR/ORB	2	D ← D OR A	✓	✓	0	0	–	–	
XOR/XORB	2	D ← D (ecxl. or) A	✓	✓	0	0	–	–	
LD/LDB	2	D ← A	–	–	–	–	–	–	
ST/STB	2	A ← D	–	–	–	–	–	–	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	–	–	–	–	–	–	3,4
LDBZE	2	D ← A; D + 1 ← 0	–	–	–	–	–	–	3,4
PUSH	1	SP ← SP − 2; (SP) ← A	–	–	–	–	–	–	
POP	1	A ← (SP); SP + 2	–	–	–	–	–	–	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H; I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LJMP	1	PC ← PC + 16-bit offset	–	–	–	–	–	–	5
BR[indirect]	1	PC ← (A)	–	–	–	–	–	–	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	–	–	–	–	–	–	5

Table 3-1. Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RET	0	PC $\leftarrow$ (SP); SP $\leftarrow$ SP + 2	–	–	–	–	–	–	
J (conditional)	1	PC $\leftarrow$ PC + 8-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
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/ DJNZW	1	D $\leftarrow$ D − 1; If D $\neq$ 0 then PC $\leftarrow$ PC + 8-bit offset	–	–	–	–	–	–	5 10
DEC/DECB	1	D $\leftarrow$ D − 1	✓	✓	✓	✓	↑	–	
NEG/NEGB	1	D $\leftarrow$ 0 − D	✓	✓	✓	✓	↑	–	
INC/INCB	1	D $\leftarrow$ D + 1	✓	✓	✓	✓	↑	–	
EXT	1	D $\leftarrow$ D; D + 2 $\leftarrow$ Sign (D)	✓	✓	0	0	–	–	2
EXTB	1	D $\leftarrow$ D; D + 1 $\leftarrow$ Sign (D)	✓	✓	0	0	–	–	3
NOT/NOTB	1	D $\leftarrow$ Logical Not (D)	✓	✓	0	0	–	–	
CLR/CLRB	1	D $\leftarrow$ 0	1	0	0	0	–	–	
SHL/SHLB/SHLL	2	C $\leftarrow$ msb - - - - - lsb $\leftarrow$ 0	✓	✓	✓	✓	↑	–	7
SHR/SHRB/SHRL	2	0 $\rightarrow$ msb - - - - - lsb $\rightarrow$ C	✓	✓	✓	0	–	✓	7
SHRA/SHRAB/SHRAL	2	msb $\rightarrow$ msb - - - - - lsb $\rightarrow$ C	✓	✓	✓	0	–	✓	7
SETC	0	C $\leftarrow$ 1	–	–	1	–	–	–	
CLRC	0	C $\leftarrow$ 0	–	–	0	–	–	–	

Table 3-1. Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
CLRVT	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
PUSHA	1	SP ← SP-2; (SP) ← PSW; PSW ← 0000H; SP ← SP-2; (SP) ← IMASK1/WSR; IMASK1 ← 00H	0	0	0	0	0	0	
POPA	1	IMASK1/WSR ← (SP); SP ← SP+2 PSW ← (SP); SP ← SP+2	✓	✓	✓	✓	✓	✓	
IDLPD	1	IDLE MODE IF KEY=1; POWERDOWN MODE IF KEY =2; CHIP RESET OTHERWISE	–	–	–	–	–	–	
CMPL	2	D-A	✓	✓	✓	✓	↑	–	
BMOV	2	[PTR_HI]+ ← [PTR_LOW]+ ; UNTIL COUNT=0	–	–	–	–	–	–	

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 word.
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 $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.
10. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen (-) means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction.

3.2 Program Status Word

The Program Status Word (PSW) is a collection of Boolean flags which contain information concerning the state of the user's program. The high byte of the PSW contains status flags and the low byte contains an interrupt mask register. The PSW high byte is shown in Figure 3-2. Table 3-2 contains descriptions of the status flags.

Table 3-2. Status Flag Descriptions

Flag	Name	Function
ST	Sticky Bit	Indicates whether any 1's were lost due to a right shift operation; primarily used for floating-point routines.
I	Interrupt Enable	Master control for 80C196KB interrupts
C	Carry Flag	Set if there is a carry (or no borrow), and otherwise cleared, as a result of an ADD or SUB instruction.
VT	Overflow Trap Flag	Set whenever overflow flag is set; cleared only by a CLRVT, JVT or JNVT instruction.
V	Overflow Flag	Set if result is out of range for signed arithmetic operation.
N	Negative Flag	Holds the algebraically correct sign as the result of an operation.
Z	Zero Flag	Set if the result of an operation is zero.

PSW:	15	14	13	12	11	10	9	8
	Z	N	V	VT	C	X	I	ST

Figure 3-2. The Program Status Word Register (High Byte)

4.0 INTERRUPTS

There are 28 different interrupt sources available on the 80C196KB. The 28 sources vector through 18 locations or interrupt vectors. The vector names and their sources are shown in Figure 4-1, and their locations are listed in Table 4-1. The four registers that control the interrupt system are: INT__PEND, INT__PEND1, INT__MASK, INT__MASK1. The Program Status Word (PSW) contains a global disable bit, I, which is set or cleared using the EI or DI instructions. Figure 4-2 shows a block diagram of the interrupt structure.

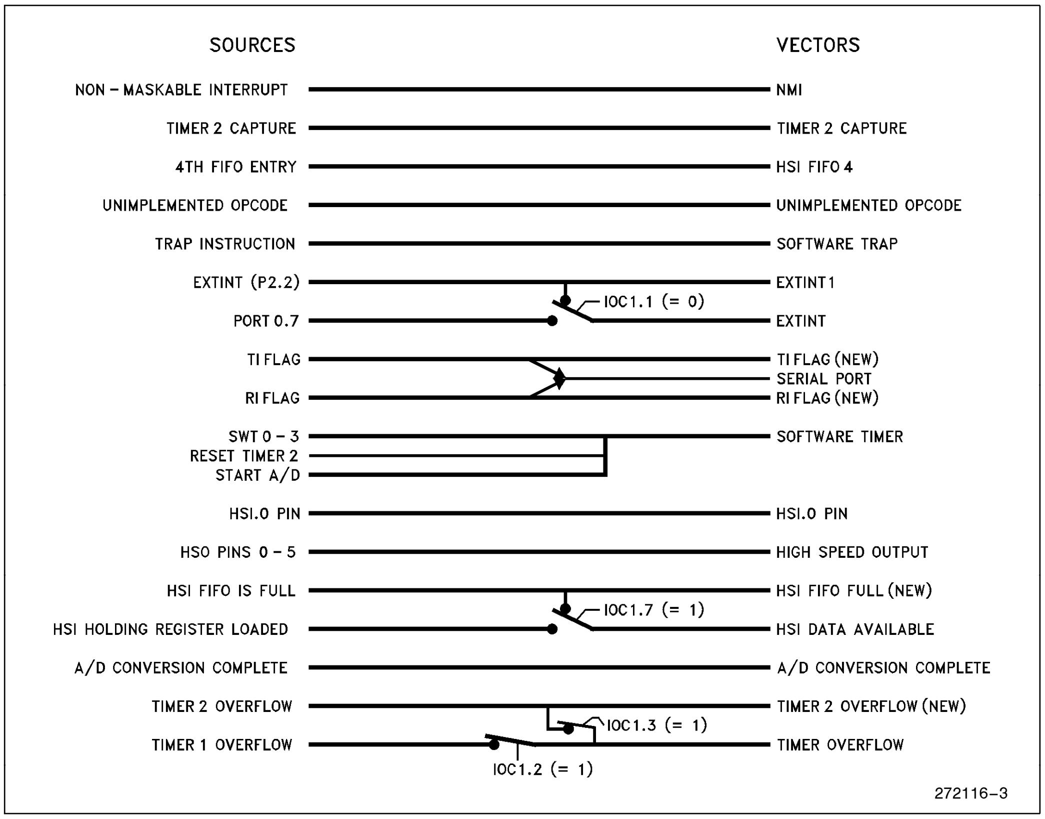


Figure 4-1. 80C196KB Interrupt Sources

Table 4-1. 80C196KB Interrupt Vector Locations

Number	Vector Name	Vector Location	Priority
INT15	NMI	203EH	15
INT14	HSI FIFO Full	203CH	14
INT13	EXTINT1	203AH	13
INT12	TIMER2 Overflow	2038H	12
INT11	TIMER2 Capture	2036H	11
INT10	4th Entry into HSI FIFO	2034H	10
INT09	RI	2032H	9
INT08	TI	2030H	8
SPECIAL	Unimplemented Opcode	2012H	N/A
SPECIAL	Trap	2010H	N/A
INT07	EXTINT	200EH	7
INT06	Serial Port	200CH	6
INT05	Software Timer	200AH	5
INT04	HSI.0 Pin	2008H	4
INT03	High Speed Outputs	2006H	3
INT02	HSI Data Available	2004H	2
INT01	A/D Conversion Complete	2002H	1
INT00	Timer Overflow	2000H	0

NOTE:
Priority 15 = highest, priority 0 = lowest

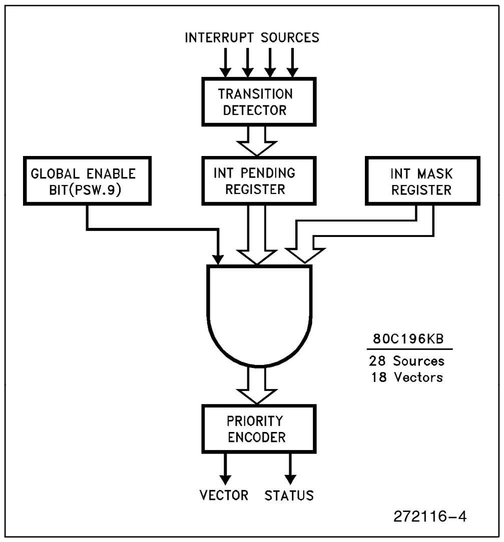
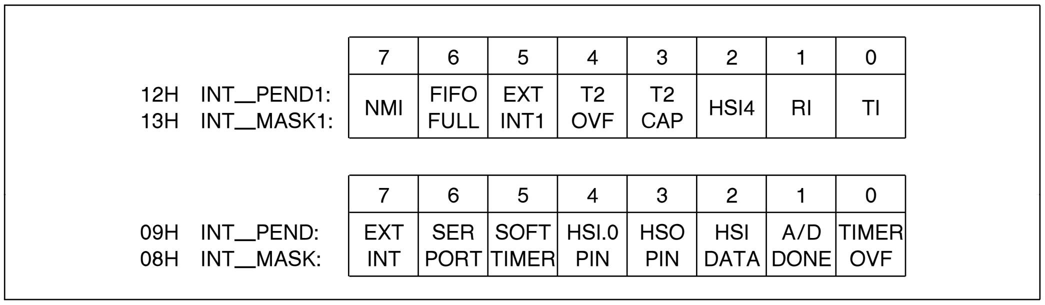


Figure 4-2. 80C196KB Interrupt Structure Block Diagram

Three special interrupts are available on the 80C196KB: the external Non-Maskable Interrupt (NMI), TRAP and Unimplemented Opcode. The external NMI pin generates an unmaskable interrupt for implementation of critical interrupt routines. The TRAP instruction is useful for developing custom software debuggers or generating software interrupts. The Unimplemented Opcode Interrupt generates an interrupt upon execution of unimplemented opcodes. This provides software recovery from random execution during hardware or software failures.

When the hardware detects one of the sixteen interrupts it sets the corresponding bit in one of two interrupt pending registers (INT_PEND and INT_PEND1). Individual interrupts are enabled or disabled by setting or clearing bits in the mask registers (INT_MASK and INT_MASK1). A one in any bit position will enable the corresponding interrupt source and a zero will disable it. The interrupt mask and pending registers are shown in Figure 4-3.



	7	6	5	4	3	2	1	0
12H INT_PEND1: 13H INT_MASK1:	NMI	FIFO FULL	EXT INT1	T2 OVF	T2 CAP	HSI4	RI	TI

	7	6	5	4	3	2	1	0
09H INT_PEND: 08H INT_MASK:	EXT INT	SER PORT	SOFT TIMER	HSI.0 PIN	HSO PIN	HSI DATA	A/D DONE	TIMER OVF

Figure 4-3. Interrupt Mask and Pending Registers

The priority encoder looks at all the interrupts that are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Table 4-1 (15 is highest, 0 is lowest). When the interrupt controller decides to process an interrupt, it executes a "call" to an Interrupt Service Routine (ISR). The address of the ISR is contained in the corresponding interrupt vector location. The interrupt controller clears the associated pending bit then pushes the return address onto the stack. The ISR should use the PUSHA instruction to save the PSW, INT__MASK, INT__MASK1 and WSR on the stack. The PUSHA instruction also clears the PSW and interrupt mask registers, disabling all interrupts. The ISR software must then implement the interrupt priority structure desired for that routine by enabling only the desired interrupts. At the end of the ISR, the POPA instruction restores the PSW, INT__MASK, INT__MASK1 and WSR to their original states and restores the original priority structure. In most cases an Interrupt Service Routine will have the basic structure shown below.

```
INT_VECTOR:  PUSHA                         ; Save the PSW, INT_MASK,
                                           ;INT_MASK1, and WSR
             LDB INT_MASK, #xxxxxxxxB      ;Set-up New Interrupt
             LDB INT_MASK1,#xxxxxxxxB      ;Priorities
             EI                            ;Enable Interrupts Again
             -
             -                             ;Service the Interrupt
             -
             POPA                          ;Restore
             RET
```

5.0 TIMERS/COUNTERS

The 80C196KB has two 16-bit timers, Timer1 and Timer2, shown in Figure 5-1. Timer1 is readable in Window 0 and writable in Window 15 while Timer2 is readable and writable in Window 0. The 80C196KB also includes separate, dedicated timers for the baud rate generator and watchdog timer. The watchdog timer is an internal timer that can be used to reset the system if the software fails to operate properly.

The Timer1 value is incremented by the 80C196KB internal clock every 8 state times. (A state time is 2 oscillator periods, or 0.167 μs with a 12 MHz crystal.) Timer1 generates a Timer Overflow Interrupt (INT00) when crossing the 0FFFFH/0000H boundary. I/O Control Register 1 (IOC1) controls the Timer1 overflow interrupt. As shown in Figure 5-2, setting IOC1.2 enables Timer1 overflow to INT00. The status of Timer1 Overflow Interrupt is read in I/O Status Register 1 (IOS1) shown in Figure 5-3.

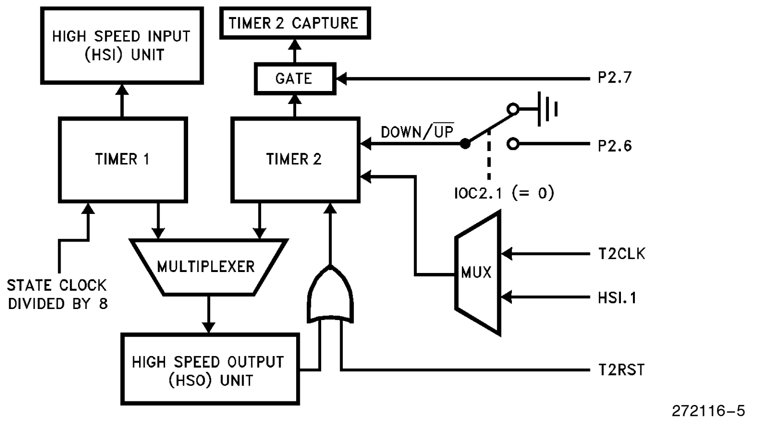


Figure 5-1. Timer Block Diagram

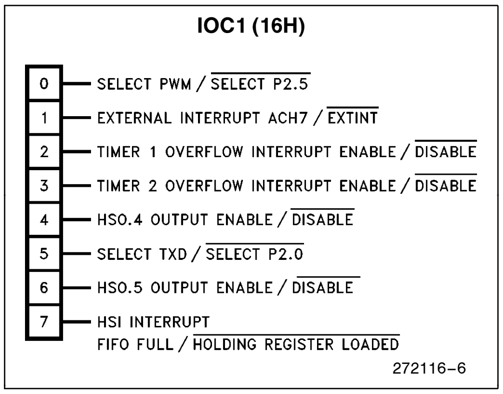

IOC1 (16H)
0 SELECT PWM / SELECT P2.5
1 EXTERNAL INTERRUPT ACH7 / EXTINT
2 TIMER 1 OVERFLOW INTERRUPT ENABLE / DISABLE
3 TIMER 2 OVERFLOW INTERRUPT ENABLE / DISABLE
4 HSO.4 OUTPUT ENABLE / DISABLE
5 SELECT TXD / SELECT P2.0
6 HSO.5 OUTPUT ENABLE / DISABLE
7 HSI INTERRUPT FIFO FULL / HOLDING REGISTER LOADED
272116-6

Figure 5-2. I/O Control Register 1 (IOC1)

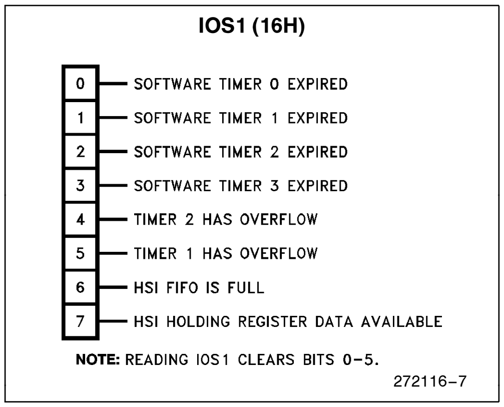

IOS1 (16H)
0 SOFTWARE TIMER 0 EXPIRED
1 SOFTWARE TIMER 1 EXPIRED
2 SOFTWARE TIMER 2 EXPIRED
3 SOFTWARE TIMER 3 EXPIRED
4 TIMER 2 HAS OVERFLOW
5 TIMER 1 HAS OVERFLOW
6 HSI FIFO IS FULL
7 HSI HOLDING REGISTER DATA AVAILABLE
NOTE: READING IOS1 CLEARS BITS 0–5.
272116-7

Figure 5-3. I/O Status Register 1 (IOS1)

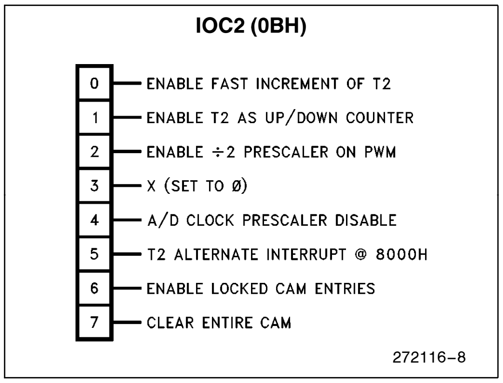

IOC2 (0BH)
0 ENABLE FAST INCREMENT OF T2
1 ENABLE T2 AS UP/DOWN COUNTER
2 ENABLE ÷2 PRESCALER ON PWM
3 X (SET TO Ø)
4 A/D CLOCK PRESCALER DISABLE
5 T2 ALTERNATE INTERRUPT @ 8000H
6 ENABLE LOCKED CAM ENTRIES
7 CLEAR ENTIRE CAM
272116-8

Figure 5-4. I/O Control Register 2 (IOC2)

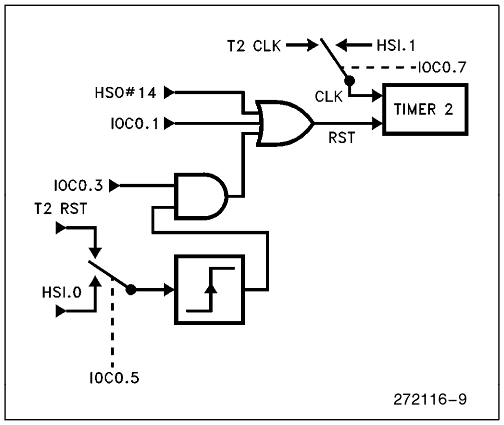


Figure 5-5. Timer2 Clock and Reset Options

I/O Control Register 1 (IOC1) and I/O Control Register 2 (IOC2) shown in Figure 5-4 determine the function of Timer2. Timer2 is driven by an external clock. Bit 7 of IOC0 controls whether the T2CLK pin or the HSI.1 pin function as the Timer2 clock input. Timer2 increments or decrements on every positive and negative transition. Bit 0 of IOC2 determines the maximum rate at which Timer2 can receive these transitions. When IOC2.0 = 1 the maximum transition speed is once per state time, and when IOC2.0 = 0 the maximum transition speed is once every 8 state times (Fast Increment Mode). Setting bit 1 of IOC2 enables Timer2 to function as an up/down counter. The T2UPDN pin determines the direction of Timer2 as an up/down counter; when T2UPDN = 1 Timer2 counts down and when T2UPDN = 0 Timer2 counts up. There are two possible external Timer2 reset sources. IOC0.3 enables the external reset function and IOC0.5 determines whether the T2RST pin or the HSI.0 pin will act as the reset source (Figure 6-4). It is also possible to reset Timer2 internally using the High Speed Output Unit or by clearing the Timer2 SFR. Figure 5-5 shows the Timer2 clock and reset options and Table 5-1 lists the Timer2 control bits.

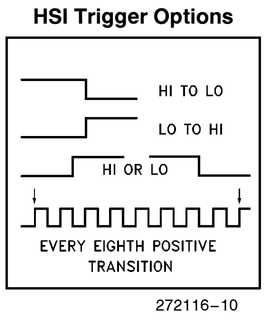
Table 5-1. Timer2 Control Bits

	Bit = 1	Bit = 0
IOC0.1	Reset Timer2 each write	No action
IOC0.3	Enable external reset	Disable
IOC0.5	HSI.0 is ext. reset source	T2RST is reset source
IOC0.7	HSI.1 is T2 clock source	T2CLK is clock source
IOC1.3	Enable Timer2 overflow int.	Disable overflow interrupt
IOC2.0	Enable fast increment	Disable fast increment
IOC2.1	Enable downcount feature	Disable downcount
P2.6	Count down if IOC2.1 = 1	Count up
IOC2.5	Interrupt on 7FFFH/8000H	Interrupt on 0FFFFH/0000H
P2.7	Capture Timer2 into T2CAPTURE on rising edge	

Timer2 can generate three interrupts: The Timer Overflow Interrupt (INT00), The Timer2 Overflow Interrupt (INT12), and The Timer2 Capture Interrupt (INT11). IOC1 determines whether Timer1 and/or Timer2 will generate INT00. Timer2 generates an overflow interrupt when crossing the 0FFFFH/0000H boundary or the 7FFFH/8000H boundary as determined by IOC2.5. A Timer2 overflow interrupts through INT00 if IOC1.3 and INT__MASK.0 are set. Alternatively, Timer2 interrupts through INT12 if INT__MASK1.3 is set. Bit 4 of I/O Status Register 1 (IOS1.4), shown in Figure 5-3, indicates that status of Timer2 Overflow Interrupt.

6.0 HIGH SPEED INPUT UNIT

The High Speed Input Unit (HSI) can record times of external events with an 8 state time (1.33 μs at 12 MHz) resolution. It can capture the value of Timer1 when an event takes place on one of the four HSI lines (HSI.0 through HSI.3). The four types of events that can trigger a capture are: rising edges only, falling edges only, rising or falling edges, or every eighth rising edge. As shown in Figure 6-2, the four input lines are independently configurable via the HSI__MODE register. This register determines the capture modes of the four inputs. A block diagram of the HSI unit is shown in Figure 6-1.



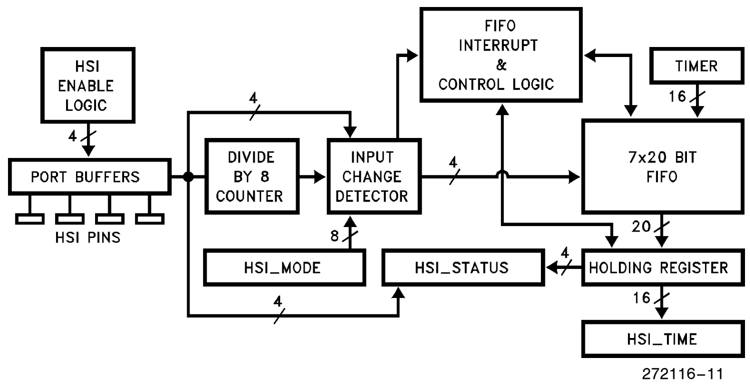


Figure 6-1. High Speed Input Unit

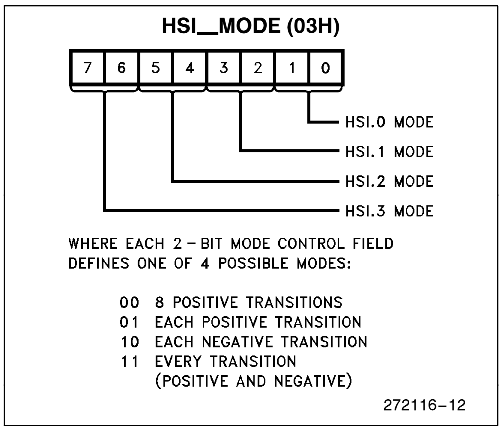
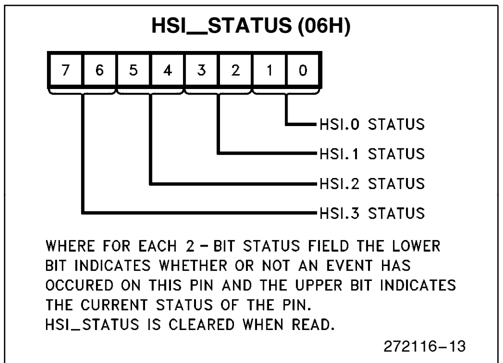

HSI_MODE (03H)
7 6 5 4 3 2 1 0
HSI.0 MODE
HSI.1 MODE
HSI.2 MODE
HSI.3 MODE
WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:
00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)
272116-12

Figure 6-2. High Speed Input Mode Register (HSI_MODE)

The HSI unit stores the Timer1 value and 4 status bits in a 7 x 20 level FIFO and holding register. It is possible to store 8 entries, 7 in the FIFO and 1 in the holding register. The HSI unit will not store events occurring after the FIFO is full. The HSI holding register contains the earliest entry placed in the FIFO. Reading the holding register unloads one level of the FIFO. The HSI unit then places the next entry into the holding register.

The contents of the HSI holding register are obtained by first reading the HSI_STATUS register and then the HSI_TIME register. The HSI_TIME register returns the event time tag. The HSI_STATUS register returns a status and an input bit for each of the four HSI lines (see Figure 6-3). The status bit indicates which line(s) caused the event and the input bit indicates the **current** input level of the line, not the level when the event occurred. Reading the HSI_TIME register unloads one level of the FIFO.


HSI_STATUS (06H)
7 6 5 4 3 2 1 0
HSI.0 STATUS
HSI.1 STATUS
HSI.2 STATUS
HSI.3 STATUS
WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.
HSI_STATUS IS CLEARED WHEN READ.
272116-13

NOTE:
HSI_STATUS is cleared when read.

Figure 6-3. High Speed Input Status Register (HSI_STATUS)

To start the HSI use the following steps: 1) Flush the FIFO, 2) Enable the HSI interrupts, 3) Initialize and enable the HSI pins. The following section of code will flush the FIFO:

```
FLUSH:  LD ZERO_REG,        ;Unload one level of
        HSI_TIME            the FIFO
        SKIP ZERO_REG       ;Wait 4 state times
        SKIP ZERO_REG       ;Wait 4 state times
        JBS IOS1, 7, FLUSH  ;Check whether FIFO
                            is empty
```

I/O Control Register 0 (IOC0), shown in Figure 6-4, can individually enable or disable the four HSI lines (HSI.0 through HSI.3). Disabling an input line disconnects it from the FIFO, changing its function from an HSI line to a general purpose input line. However, the corresponding HSI_STATUS input bits indicate the current state of the line regardless of whether the line functions as an HSI input line or as a general purpose input line.

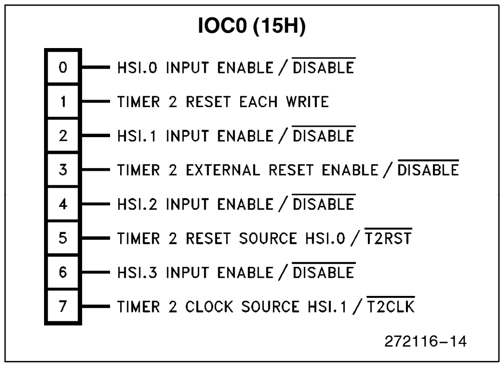

IOC0 (15H)
0 HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1 TIMER 2 RESET EACH WRITE
2 HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3 TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4 HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5 TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6 HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7 TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$
272116-14

Figure 6-4. I/O Control Register 0 (IOC0)

The HSI unit can generate three interrupts: The HSI Data Available Interrupt (INT02), the HSI__FIFO__4 Interrupt (INT10) and the HSI FIFO FULL Interrupt (INT14). Bit 7 of I/O Control Register 1 (IOC1) controls the INT02 source. If IOC1.7 = 0 loading the holding register will cause INT02; otherwise if IOC1.7 = 1 loading the sixth entry into the FIFO (not including the holding register) will cause INT02. After INT02 occurs bits 6 and 7 of I/O Status Register 1 (IOS1) indicate which source caused the interrupt. The sources for INT10 and INT14 are independent of IOC1. Loading the fourth entry into the FIFO causes INT10 and loading the sixth entry into the FIFO causes INT14. Note if IOC1.7 is set, loading the sixth entry into the FIFO will cause both INT02 and INT14.

7.0 HIGH SPEED OUTPUT UNIT

The HSO unit can trigger events at specified times based on Timer1 or Timer2. These programmable events include: starting an A/D conversion, resetting Timer2, generating up to four software time delays and setting or clearing one or more of the 6 output lines (HSO.0 through HSO.5). The HSO unit stores pending events and their specified times in a CAM (Content Addressable Memory) file. Figure 7-1 shows a block diagram of the HSO unit.

The CAM file is the main component of the HSO. This file stores up to eight commands. Each CAM register is 24 bits wide. Sixteen bits specify the action time, and 8 bits specify the nature of the action and whether Timer1 or Timer2 is the reference. Timer2 transitions should not occur faster than once every 8 state times when it is used as a reference for the HSO. Commands for the HSO first enter the HSO holding register. They then enter the CAM when an empty CAM register is available. Commands must be in the CAM to execute; commands in the holding register will not execute. It takes one state time to compare each CAM location, so 8 state times (1.33 μs with a 12 MHz clock) are necessary for a complete CAM search. The HSO unit triggers the specified event when it finds a time match.

Writing to the HSO__COMMAND register and the HSO__TIME register loads the HSO holding register. When the next opening in the CAM file is available the contents of the HSO holding register move into it. The HSO__COMMAND register shown in Figure 7-2 specifies the event type, whether an interrupt is to occur, and the reference timer. The I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. If IOSO.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOSO.7 equals 0, the holding register is empty. The holding register must be empty before writing the action time to the HSO__TIME registers. If the holding register is not empty, writing to the HSO will overwrite the current holding register value. Always write the command byte first, followed by the time word.

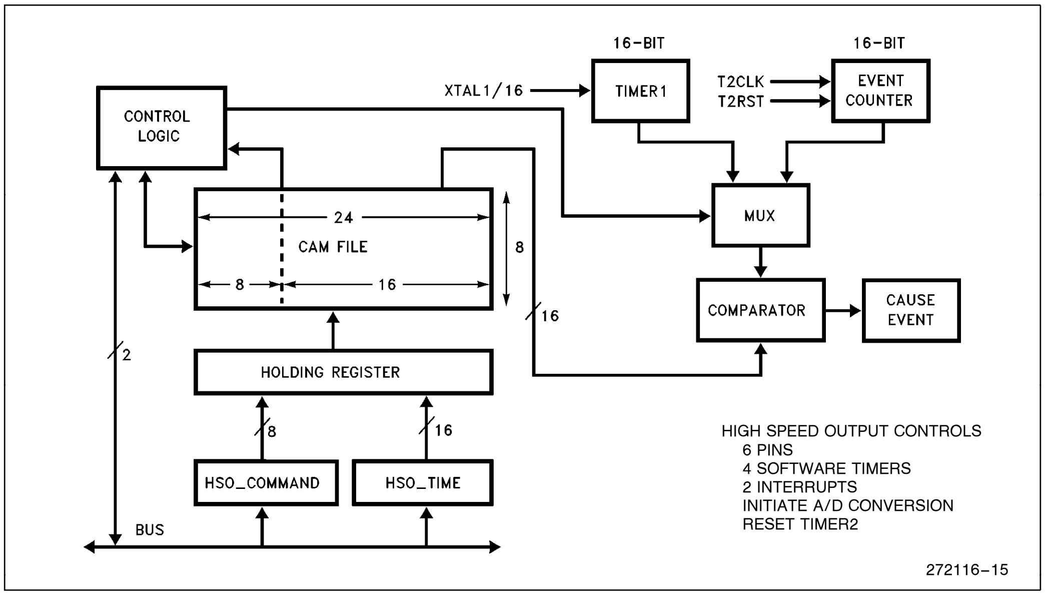


Figure 7-1. High Speed Output Block Diagram

HSO_COMMAND (06H)

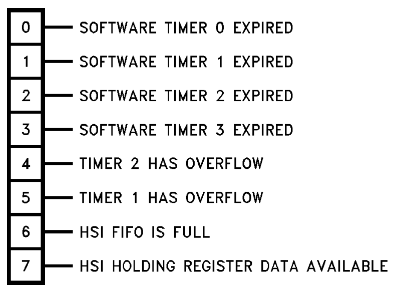
	7	6	5	4	3	2	1	0
HSO_ COMMAND	CAM LOCK	TMR2/ $\overline{\text{TMR1}}$	SET/ CLEAR	INT/ $\overline{\text{INT}}$	CHANNEL			

CAM Lock	— Locks event in CAM if this is enabled by IOC2.6 (ENA_LOCK)	
TMR/$\overline{\text{TMR1}}$	— Events Based on Timer2/Based on Timer1 if 0	
SET/$\overline{\text{CLEAR}}$	— Set HSO pin/Clear HSO pin if 0	
INT/$\overline{\text{INT}}$	— Cause interrupt/No interrupt if 0	
CHANNEL: (in Hex)	0–5:	HSO pins 0–5 separately
	6:	HSO pins 0 and 1 together
	7:	HSO pins 2 and 3 together
	8–B:	Software Timers 0–3
	C–D:	Unflagged Events (Do not use for future compatibility)
	E:	Reset Timer2
	F:	Start A to D Conversion

Figure 7-2. High Speed Output Command Register (HSO_Command)

An entry placed in the CAM remains there until its execution unless a chip reset occurs or the CAM clear bit (IOC2.7) is set. It is possible to cancel an external pending event by writing the opposite event with the same time tag to the CAM. However, both events remain in the CAM until their time tag is matched or the CAM is cleared. Setting bit 2 of IOC2 enables the CAM locking function. Setting the CAM lock bit (HSO_COMMAND.7) locks the command in the CAM; a locked CAM entry will execute whenever its time tag matches the reference time. Locked entries are useful in applications requiring periodic or repetitive events to occur. The HSO unit can generate multiple PWM's by locking CAM entries and using Timer2 as a reference. (See Software Example 4)

IOS1 (16H)



0 SOFTWARE TIMER 0 EXPIRED
1 SOFTWARE TIMER 1 EXPIRED
2 SOFTWARE TIMER 2 EXPIRED
3 SOFTWARE TIMER 3 EXPIRED
4 TIMER 2 HAS OVERFLOW
5 TIMER 1 HAS OVERFLOW
6 HSI FIFO IS FULL
7 HSI HOLDING REGISTER DATA AVAILABLE

272116–16

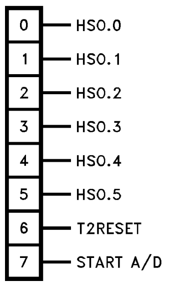
NOTE:
IOS1 is cleared when read.

Figure 7-3. I/O Status Register 1 (IOS1)

The HSO unit can generate two interrupts (providing HSO_COMMAND.4 is set): The High Speed Output interrupt (INT03) and The Software Timer interrupt (INT05). The High Speed Output interrupt occurs as a result of changes on one or more of the six output pins. The other HSO commands, triggering the A/D Converter, resetting Timer2 and setting a Software Timer Flag, generate INT05. The I/O Status Registers IOS1 and IOS2, shown in Figure 7-3 and Figure 7-4 indicate which event(s) caused a HSO interrupt.

IOS2 (17H)

INDICATES WHICH HSO EVENT OCCURRED



272116–17

NOTE:
IOS2 is cleared when read.

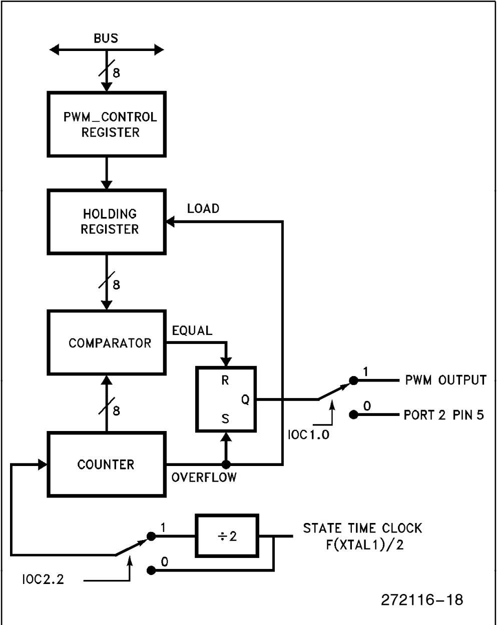
Figure 7-4. I/O Status Register 2 (IOS2)

The HSO unit can generate interrupts at preset times via four "Software Timers". Software Timer Flags are set in the I/O Status Register 1 (IOS1) at the preprogrammed times. If the interrupt bit in the HSO command register is set, a Software Timer Interrupt will also occur at the designated time. The interrupt service routine can then examine IOS1 to determine which software timer expired and caused the interrupt. The most common use of the software timers is to trigger interrupt routines that must occur at regular intervals.

8.0 PULSE WIDTH MODULATION OUTPUT

The Pulse Width Modulator of the 80C196KB, when used with external hardware, can provide useful signals for a variety of applications. The PWM output can perform digital to analog conversions and drive several types of motors which require a PWM waveform for more efficient operation. A block diagram of the PWM circuit is shown in Figure 8-1. Three registers control the PWM: I/O Control Register 1 (IOC1), I/O Control Register 2 (IOC2) and the PWM Register (PWM_CONTROL). The PWM output shares a pin with Port 2; setting IOC1.0 selects the PWM function rather then the standard port function.

The PWM output waveform is a variable duty cycle pulse that repeats every 256 state times (42.75 μs @ 12 MHz) or 512 state times (85.5 μs @ 12 MHz) if the prescaler bit (IOC2.2) is set. The PWM frequencies for different clock speeds are shown in Table 8-1. Writing a value between 0 and 255 to the PWM_CONTROL register will change the duty cycle. The PWM unit has an 8-bit counter that is incremented every state time or every other state time if the prescaler bit is set. When



• Duty Cycle Programmable in 256 Steps

Figure 8-1. PWM Block Diagram

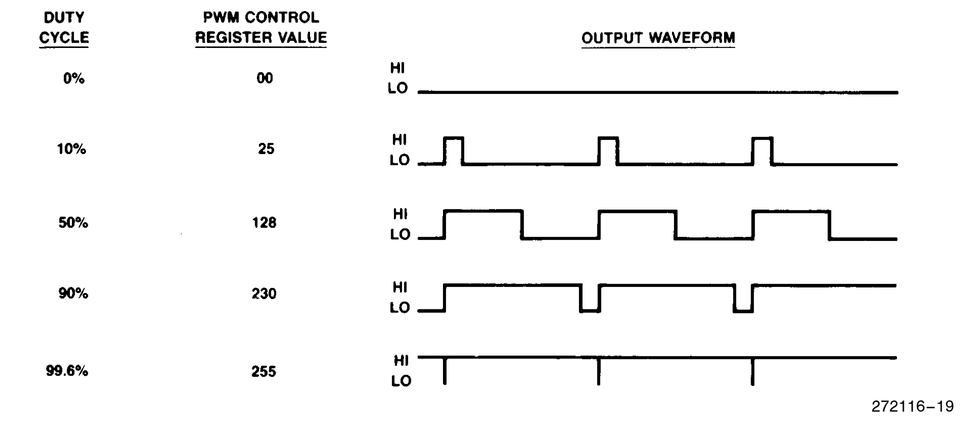


Figure 8-2. Typical PWM Outputs

the counter equals 0 the PWM output switches high; when the counter matches the value in the PWM_ CONTROL register the PWM output switches low; and when the counter overflows the PWM output switches high again. Typical output waveforms are shown in Figure 8-2. Values written to PWM_ CONTROL are loaded into a holding register when the counter overflows. This is so the compare circuit will not recognize a new value until the counter has expired, thus preventing missed PWM edges.

Table 8-1. PWM Frequencies

XTAL1 =	8 MHz	10 MHz	12 MHz
IOC2.2 = 0	15.6 KHz	19.6 KHz	23.6 KHz
IOC2.2 = 1	7.8 KHz	9.8 KHz	11.8 KHz

9.0 ANALOG OUTPUTS

Both the PWM output and the HSO unit can generate analog outputs. Either peripheral will generate a rectangular pulse train that varies in duty cycle and period. Filtering the output will create a smooth analog signal. This filtering is typically done after the signal is buffered to make it swing over the desired analog output voltage range. A block diagram of the type of circuit needed is shown in Figure 9-1. The filter can be a simple RC network or an active filter. Shown in Figure 9-2 is a circuit used for low output currents, (less than 100 μA or so). The PWM unit can generate these waveforms if a fixed period on the order of 42.75 μs or 85.5 μs (at 12 MHz) is acceptable. The HSO unit can generate waveforms with a period of up to 87.5 ms (using Timer1 at 12 MHz).

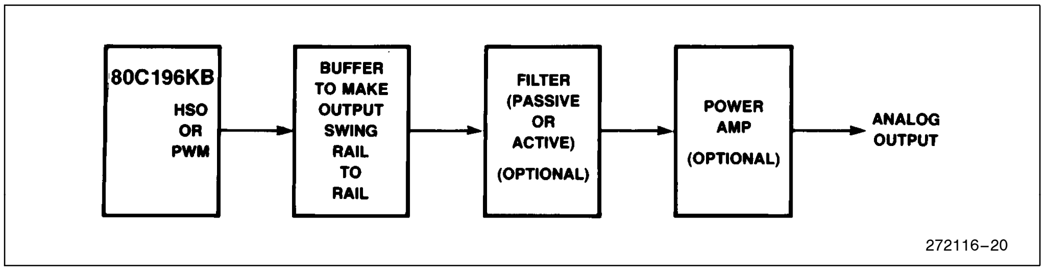

80C196KB
HSO OR PWM
BUFFER TO MAKE OUTPUT SWING RAIL TO RAIL
FILTER (PASSIVE OR ACTIVE) (OPTIONAL)
POWER AMP (OPTIONAL)
ANALOG OUTPUT
272116-20

Figure 9-1. D/A Buffer Block Diagram

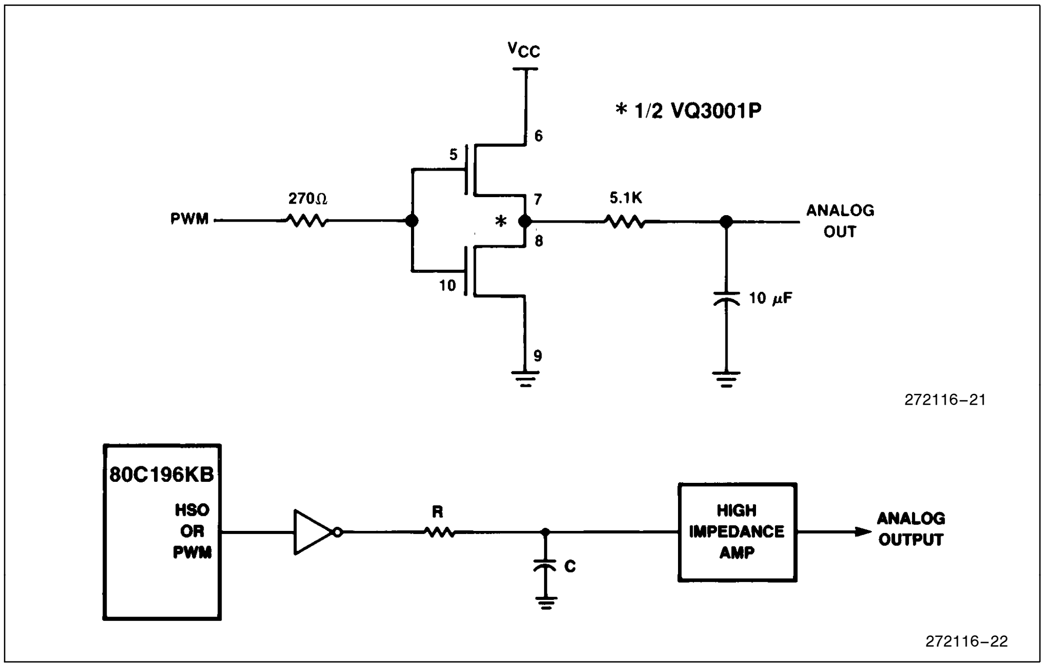


Figure 9-2. PWM to Analog Conversion Circuitry

10.0 ANALOG TO DIGITAL CONVERTER

The 80C196KB analog interface consists of a sample-and-hold, an 8 channel multiplexer, and a 10-bit analog-to-digital converter. A block diagram of the A/D converter is shown in Figure 10-1. Port 0, an input-only port, shares the analog inputs ACH0 through ACH7. The A/D Converter uses the successive approximation method to perform an A/D conversion on one input at a time. Three SFRs control the A/D Converter. The AD__COMMAND register controls which channel and when a conversion will start, and the AD__RESULT (low and high) registers store the 10-bit conversion result. Bit 4 of the I/O Control Register 2 (IOC2.4) controls the number of state times required for the conversion.

To set-up an analog-to-digital conversion load the desired analog input channel into the lower three bits of the AD__COMMAND register. The GO bit, bit 4 of the AD__COMMAND register, controls when the conversion will start. If the GO bit is set the conversion will start immediately, otherwise the HSO unit will trigger the conversion. The AD__COMMAND register is shown in Figure 10-2. The A/D result registers (AD__RESULT(hi) and AD__RESULT(lo)), shown in Figure 10-3 and Figure 10-4 contain the 10-bit conversion result. The AD__RESULT(hi) register contains the most significant 8 bits of the result. Bits 6 and 7 of the AD__RESULT(lo) register contain the remaining least significant bits (LSB's) of the result. Also, the lower four bits of the AD__RESULT(lo) register contain the A/D channel number and the A/D status as shown in Figure 10-3. The AD__RESULT(lo) status bit, when set, indicates that an A/D conversion is in progress. It takes 8 state times to set this bit after the start of an A/D conversion.

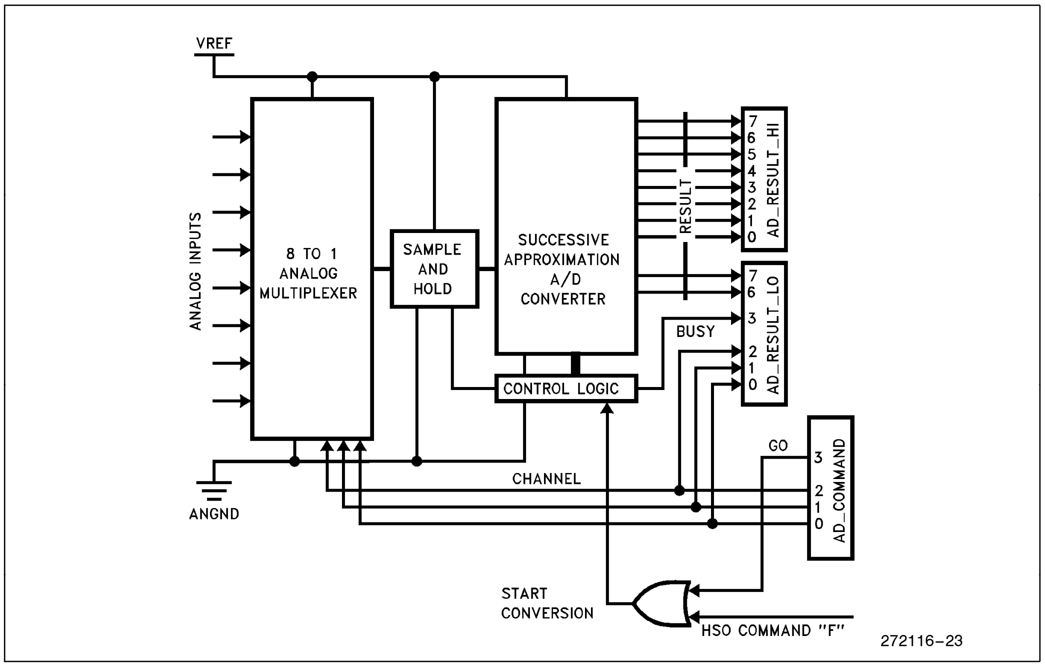


Figure 10-1. A/D Converter Block Diagram

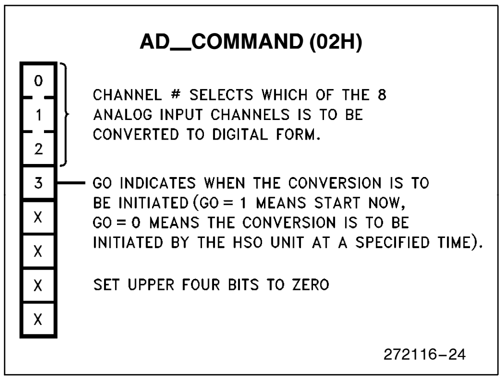

AD_COMMAND (02H)
0
1
2
3
X
X
X
X
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).
SET UPPER FOUR BITS TO ZERO
272116–24

Figure 10-2. A/D Command Register (AD_COMMAND)

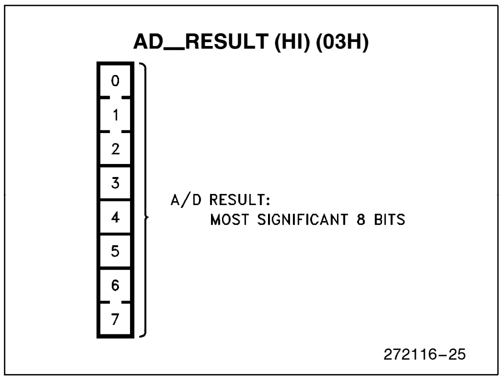


Figure 10-3. A/D Result High Register (AD_RESULT(HI))

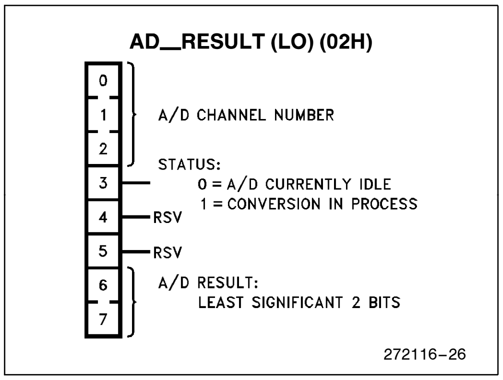


Figure 10-4. A/D Result Low Register (AD_RESULT(LO))

The clock prescaler bit of I/O Control Register 2 (IOC2.4) determines the number of state times required for an A/D conversion. High crystal frequencies require more states to complete a conversion to allow enough settling time for the internal comparator. When IOC2.4 = 1 the A/D conversion time is 91 state times (22.75 μs for an 8 MHz crystal) otherwise the A/D conversion time is 158 state times (26.33 μs for a 12 MHz crystal). An A/D Conversion Complete Interrupt (INT01) occurs on completion of an A/D conversion. It is possible to generate a Software Timer Interrupt (INT05) at the start of an A/D conversion by using the HSO unit to trigger the conversion.

11.0 SERIAL PORT

The Serial Port on the 80C196KB has one synchronous (Mode 0) and three asynchronous modes (Modes 1–3). The asynchronous modes are full duplex, meaning they can transmit and receive data simultaneously. The receiver on the 80C196KB is double buffered so the reception of a second byte may begin before the first byte is read. The transmitter is also double buffered and can generate continuous transmissions.

In the asynchronous modes, the TxD pin is the serial port transmission line and the RxD pin is the serial port reception line. Data to and from the serial port is transferred through the Serial Port Buffers. The Transmit Buffer SBUF(TX) contains data for transmission, the Receive Buffer SBUF(RX) stores the received data.

The Serial Port Control (SP_CON) register and the Serial Port Status (SP_STAT) register control the serial port. Bit 5 of the I/O Control Register 1 (IOC1), shown in Figure 5-2 enables the TxD pin for serial port use. Writing to location 11H in Window 0 accesses the SP_CON register while reading it accesses the SP_STAT register. The SP_CON register contains bits that: determine the Serial Mode (M1 and M2), enable parity (PEN), enable the receiver (REN), and determine the state and function of the 9th data bit when using Modes 2 and 3 (TB8). The SP_STAT register contains flags that indicate: receive Overrun Error (OE), Framing Error (FE), Transmitter Empty (TXE), Transmit Interrupt (TI), Receive Interrupt (RI), Receive Parity Error (RPE) and Receive Bit 8 (RB8). The SP_CON and SP_STAT registers are shown in Figure 11-1.

SP_CON:

7	6	5	4	3	2	1	0	
X	X	X	TB8	REN	PEN	M2	M1	11H

TB8	— Sets the ninth data bit for transmission. Cleared after each transmission. Not valid if parity is enabled.
REN	— Enables the receiver
PEN	— Enables the Parity function (even parity)
M2, M1	— Sets the mode. Mode0 = 00, Mode1 = 01, Mode2 = 10, Mode 3 = 11
X	— Reserved bit. Must be written as 0.

SP_STAT:

7	6	5	4	3	2	1	0	
RB8/ RPE	RI	TI	FE	TXE	OE	X	X	11H

RB8	— Set if the 9th bit is high on reception (parity disabled)
RPE	— Set if parity is enabled and a parity error occurred
RI	— Set after the last data bit is sampled
TI	— Set at the beginning of the STOP bit transmission
FE	— Set if no STOP bit is found at the end of a reception
TXE	— Set if two bytes can be transmitted
OE	— Set if the receiver buffer is overwritten

Reading SP_STAT clears Bits 2, 4, 5, 6 and 7

Figure 11-1. Serial Port Control and Status Registers (SP_CON and SP_STAT)

The most common use of Mode 0, the synchronous mode, is to expand the I/O capability of the 80C196KB using shift registers. In this mode the port outputs a set of 8 clock pulses on the TxD pin and either transmits or receives data synchronously on the RxD pin. Data is transferred 8 bits at a time with the LSB first. A diagram of the relative timing of these signals is shown in Figure 11-2. A schematic of a typical circuit that uses shift registers is shown in Figure 11-3. Since this circuit inverts the input data bits, software must re-invert them. The users software routine must control two pins (PX.X) to load data into the 74165 and to enable the shift clock on the 74164.

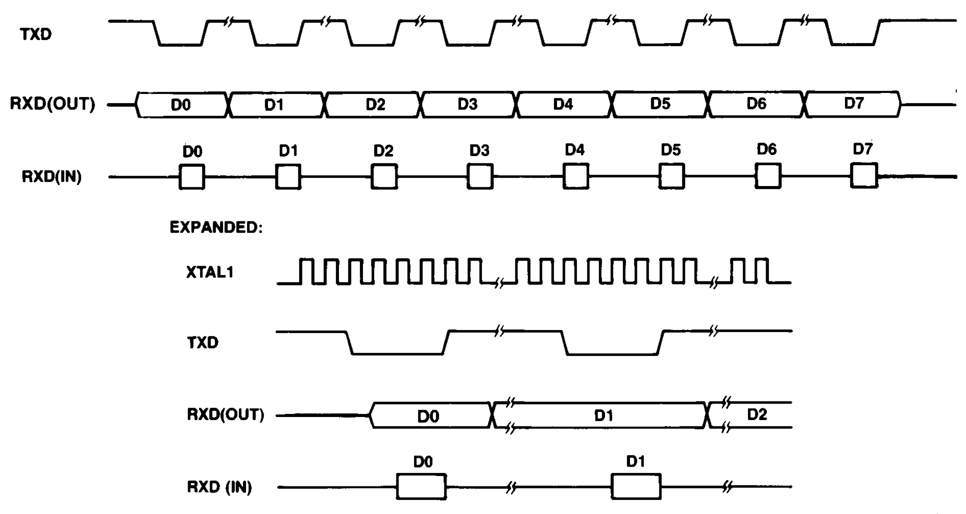


Figure 11-2. Mode 0 Timing

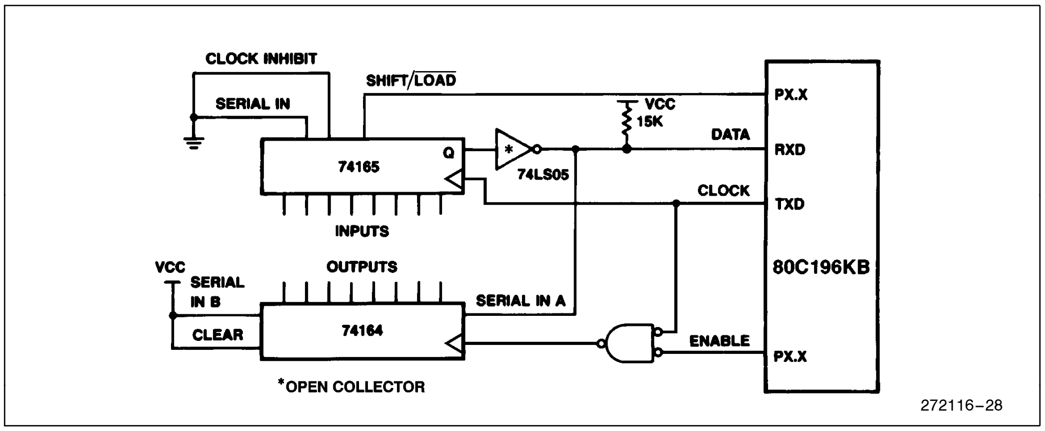


Figure 11-3. Typical Shift Register Circuit

Mode 1 is the standard asynchronous mode used for normal serial communication. The data frame used in this mode is shown in Figure 11-4. It consists of 10 bits: a start bit, 8 data bits (LSB first) and a stop bit. If parity is enabled (PEN = 1), an even parity bit is sent instead of the 8th data bit. Modes 2 and 3 are 9-bit modes commonly used for multi-processor communications. The data frame used in these modes, shown in Figure 11-4, consists of 11 bits: a start bit, nine data bits (LSB first) and a stop bit. Devices in Mode 2 will interrupt upon reception only if the 9th data bit is set. Devices in Mode 3 will always interrupt upon reception. Mode 3 also allows transmission of 8 data bits plus an even parity bit.

By making use of Modes 2 and 3 software can easily communicate between processors. Software sets the 9th data bit when sending an address or command for all the processors. In standby mode all the processors wait in Mode 2 for a byte with the 9th bit set. When they receive that byte, each processor determines if the next message is for them. The processor(s) that is to receive the message switches to Mode 3 and receives the information. Since the other processors remain in Mode 2, the software can send information with the 9th data bit cleared ensuring that only the previously addressed processor(s) will receive the information. This scheme minimizes the overall CPU time required for the serial port.

A typical connection diagram for multiprocessor communication is shown in Figure 11-5. This type of communication can connect peripherals to a desk top computer, an axis controller of a multi-axis machine, or any other group of microcontrollers.

The Serial Port sets the Transmit Interrupt (TI) and the Receive Interrupt (RI) flags in the SP__STAT register to indicate when operations are complete. TI is set when the last data bit is sent. RI is set when the last data bit is received (except in mode 2). In mode 2 the RI flag is set only when the 9th data bit of the reception is set. Reading the SP__STAT register clears the TI and RI flags. In response to the RI and TI flags the Serial Port generates three possible interrupts: the Transmit Interrupt (INT08), the Receive Interrupt (INT09) and the Serial Port Interrupt (INT06). Both the RI and TI flags generate INT06, which exists for compatibility with the 8096BH. Software should enable INT06 and disable both INT08 and INT09 for 8096BH compatibility. For normal operation software should disable INT06 and enable both INT08 and INT09.

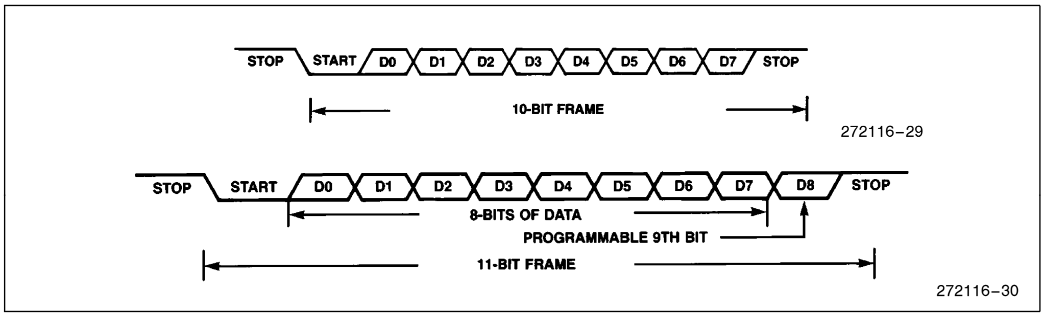


Figure 11-4. Serial Port Frames, Mode 1, 2 and 3

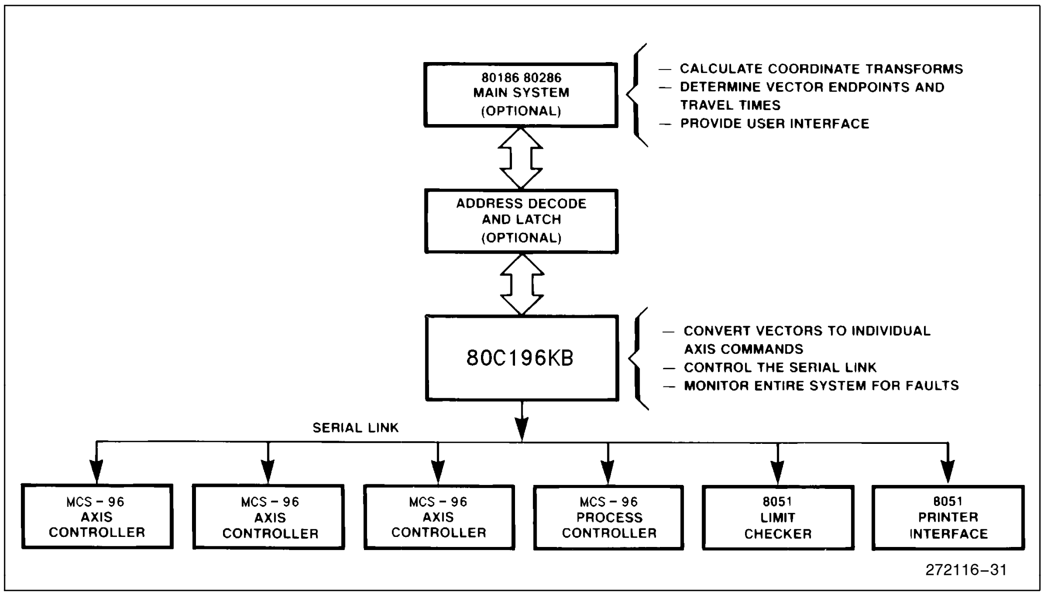


Figure 11-5. Multiprocessor Communication

The Baud Rate Register (BAUD__REG) controls the baud rates for the serial modes. This is a byte wide register that 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 byte. The most significant bit of the baud value determines the clock source for the baud rate generator. Setting this bit will select the XTAL1 pin as the clock, otherwise the T2CLK pin will function as the clock. To determine the baud value use the formulas shown in Figure 11-6. The baud values for common baud rates when using XTAL1 as the clock source are shown in Table 11-1. In most cases a serial link will work with up to 5.0% difference between baud rates.

Common baud rate values, using XTAL1 at 16 MHz, are shown below.

Table 11-1. Common Baud Rate Values

Baud Rate	Baud Register Value	
	Mode 0	Others
9600	8340H	8067H
4800	8682H	80CFH
2400	8D04H	81AOH
1200	9A0AH	8340H
300	E82BH	8D04H

Asynchronous Modes 1, 2 and 3:

$$\text{BAUD\_REG} = \frac{\text{XTAL1}}{\text{Baud Rate} * 16} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate} * 8}$$

Synchronous Mode 0:

$$\text{BAUD\_REG} = \frac{\text{XTAL1}}{\text{Baud Rate} * 2} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate}}$$

B must only equal 0 in modes 1, 2 or 3, when using XTAL1 as the clock source. Do not use B = 0 in mode 0.

Figure 11-6. Baud Rate Formulas

12.0 SOFTWARE EXAMPLES

This section contains 7 software examples that use the major 80C196KB peripherals. The first example is a Table Look-Up and Interpolation program that uses many of the 80C196KB code features. The following programs demonstrate the use of: the HSI Unit, the HSO Unit, the PWM Output and the A/D Converter.

To avoid repetitive declarations the examples use the "include" file (80C196KB.INC) shown in Listing 12-0. This file contains the definitions for the 80C196KB Special Function Registers (SFRs). The software examples were written for use on the 80C196KB evaluation board.

Listing 12-0.
Include File 80C196KB.INC

```
;************************************************
;  80C196.INC - DEFINITION OF SYMBOLIC NAMES FOR
;          THE I/O REGISTERS OF THE 80C196KB
;************************************************
;       80C196KB SFR's
R0              EQU    00H:WORD     ; R
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
WATCHDOG        EQU    0AH:BYTE     ;   W
TIMER1          EQU    0AH:WORD     ; R
IOC2            EQU    0BH:BYTE     ;   W
TIMER2          EQU    0CH:WORD     ; R
BAUD_RATE       EQU    0EH:BYTE     ;   W
IOPORT0         EQU    0EH:BYTE     ; R
IOPORT1         EQU    0FH:BYTE     ; R/W
IOPORT2         EQU    10H:BYTE     ; R/W
SP_CON          EQU    11H:BYTE     ;   W
SP_STAT         EQU    11H:BYTE     ; R
IPEND1          EQU    12H:BYTE     ; R/W
IMASK1          EQU    13H:BYTE     ; R/W
WSR             EQU    14H:BYTE     ; R/W
IOC0            EQU    15H:BYTE     ;   W
IOS0            EQU    15H:BYTE     ; R
IOC1            EQU    16H:BYTE     ;   W
IOS1            EQU    16H:BYTE     ; R
IOS2            EQU    17H:BYTE     ; R
PWM_CONTROL     EQU    17H:BYTE     ;   W
SP              EQU    18H:WORD     ; R/W
```

272116-32

12.1 Example 1—Table Look-Up and Interpolation

A good way to increase speed for many processing tasks is to use table look-up with interpolation. The program shown in Listing 12-1 uses 17 points at evenly spaced intervals to characterize a function. Example 1 stores the 17 points as output values in a table of words. These values correspond to 17 input values (0, 20D, 40D, . . . 340D): an input = 0 returns the first output word, an input = 20D returns the second output word, and so on. Listed in Table 12-1 are the corresponding input and output values. To compute the output value for any intermediate input value (i.e., 28D) the program uses a linear approximation based on the nearest function values. Given below is a description of the interpolation process.

Table 12-1. Table of Input and Output Values

Input Value	Output Value
00D	0000H
20D	1900H
40D	2EE0H
60D	41A0H
80D	5140H
100D	5DC0H
120D	6720H
140D	6D60H
160D	7080H
180D	7080H
200D	6D60H
220D	6720H
240D	5DC0H
260D	5140H
280D	41A0H
300D	2EE0H
320D	1900H
340D	0000H

The linear interpolation formula:

RESULT = OUT__LOW + (OUT__DIF/SCALE) * (IN__VAL − IN__LOW)

IN__VAL	= The intermediate input value.
SCALE	= The difference between the nearest table input values.
IN__LOW	= The nearest table input value that is lower than IN__VAL.
OUT__DIF	= The difference between the nearest table output values.
OUT__LOW	= The lower table output value.
OUT__HIGH	= The higher table output value.

For example, if IN__VAL = 68D (44H) then OUT__LOW = 41A0H, OUT__HIGH = 5140H, OUT__DIF = 0FA0H, SCALE = 20D (14H), and INT__LOW = 60D (3CH)

RESULT = 41A0H + 0FA0H/14H * (44H − 3CH) = 75C0H

Listing 12-1.
Table Look-Up and Interpolation - INTERP.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;*********************************************************************
;   INTERP.A96 -         TABLE-LOOKUP AND INTERPOLATION
;*********************************************************************
INTERPOLATION  MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:\INTEL\INCLUDE\80C196.INC)
$LIST

CSEG

;The table values below are 18 points for the function Y = 340*X - X*X
;Max_in_val equals the outer bound of the input values

TABLE:      DCW    0000H, 1900H, 2EE0H, 41A0H
            DCW    5140H, 5DC0H, 6720H, 6D60H
            DCW    7080H, 7080H, 6D60H, 6720H
            DCW    5DC0H, 5140H, 41A0H, 2EE0H
            DCW    1900H, 0000H

MAX_IN_VAL  EQU 340D
SCALE       EQU MAX_IN_VAL/(($-TABLE)/2)-1)

RSEG AT 30H
RSV:        DSW 8                           ;Reserved space in RISM

RSEG
VALUE:      DSL 1                           ;Temporary value
OUT_LOW:    DSW 1                           ;Low table output value
OUT_HIGH:   DSW 1                           ;High table output value
IN_LOW:     DSW 1                           ;Low table input value
RESULT:     DSW 1                           ;Interpolated output value
TAB_PTR:    DSW 1                           ;Pointer into output table
IN_VAL:     DSW 1                           ;Actual input must be less
                                            ;than max_in_val

CSEG AT 2080H
INIT:       LD SP, #100H                    ;Load stack pointer
            LD IN_VAL, #320D                ;Load actual value must be
                                            ;less than max_in_val
POINTER:    LD TAB_PTR, IN_VAL              ;Table ptr=actual value
            DIVB TAB_PTR, #SCALE            ;Table ptr = table ptr\scale
            AND TAB_PTR, #00FFH             ;Discard remainder

            MULB IN_LOW, TAB_PTR, #SCALE    ;Determine the lower table
                                            ;input
            SUB IN_VAL, IN_VAL, IN_LOW      ;In_val = in_low - in_val

OUT_VALUES: SHL TAB_PTR, #1                 ;Multiple pointer by 2
            LD OUT_LOW, TABLE[TAB_PTR]      ;Out_low = low output value
            LD OUT_HIGH, (TABLE+2)[TAB_PTR] ;Out_high = hi output value
            SUB OUT_HIGH, OUT_LOW           ;out_high = out_high-out_low

;Calculate result using interpolation formula -
;result = out_low + (out_high - out_low)\scale * (in_val - in_low)

CAL_RESULT: LD VALUE, IN_VAL                ;Value = (in_val - in_low)
            MUL VALUE, OUT_HIGH             ;value * (out_high-out_low)
            DIV VALUE, #SCALE               ;Value = value \ scale
            AND VALUE, #0000FFFFH           ;Discard remainder
            ADD RESULT, VALUE, OUT_LOW      ;Result = value + value
            BR $
END                                         ;Last line must be end
```

272116–33

12.2. Example 2— Using the High Speed Input Unit

A common use of the HSI Unit is to monitor a signal and measure the time between positive and negative transitions. Example 2 shown in Listing 12-2 uses the pins HSI.0 and HSI.1 to monitor a signal. HSI.0 captures negative transition times and HSI.1 captures positive transition times. The program then calculates and stores transition time differences in a time table (TIME__TABLE). The value of TABLE__SIZE determines the size of TIME__TABLE. Once TIME__TABLE contains the designated number of values the software disables the HSI pins. Note the program discards the first transition time capture because the signal is input on the HSI pins before they are enabled.

Listing 12-2.
Using The High Speed Input Unit - HSIA.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;*********************************************************************
; HSIA.A96 - Using The High Speed Input Unit
;*********************************************************************
HSIA   MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:INTEL\INCLUDE\80C196.INC)
$LIST

RSEG AT 30H
RSV:        DSW     8                       ;Reserved space in RISM

;HSI.0 and HSI.1 monitor the same signal, HSI.0 captures the time of
;every positive transition and HSI.1 captures the time of every negative
;transition.  Event1 and event2 store the times of each consecutive
;transition.  TIME_TABLE stores the times between each transition.  The
;size of TIME_TABLE is determined by TABLE_SIZE.  All time values are
;based on Timer1.  8 state times = 1 Timer1 count

DSEG
TABLE_SIZE  EQU     0AH                     ;Size of time array
TIME_TABLE: DSW     TABLE_SIZE              ;Array of event time
                                            ;differences
RSEG
EVENT1:     DSW     1                       ;Transition time
EVENT2:     DSW     1                       ;Transition time
TIME:       DSW     1                       ;Time between events
PTR:        DSW     1                       ;Time array pointer
SIZE:       DSW     1

CSEG AT 2004H
DCW         CAL_TIME                        ;Address of HSI Data
                                            ;Available
                                            ;Interrupt Service Routine
CSEG AT 2080H

;Load the stack pointer, disable interrupts, unmask HSI Data Available
;Interrupt, mask all other interrupts and disable HSI.0 and HSI.1

INIT:       LD SP, #100H                    ;Load stack pointer
            DI                              ;Disable interrupts
            LDB INT_MASK, #00000100B        ;Unmask HSI Data Available
            CLRB IMASK1                     ;Mask all other interrupts
            CLRB IOC0                       ;Disable HSI.0 and HSI.1
            LDB IOC1, #10000000B            ;Select HSI Data Available

;Flush the FIFO, enable HSI.0 to capture positive transitions and HSI.1
;to capture negative transitions

FLUSH:      CLR HSI_TIME                    ;Unload one FIFO level
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
            JBS IOS1, 7, FLUSH              ;If FIFO is not empty goto
                                            ;flush
            LDB HSI_MODE, #00001001B        ;Set HSI.0 to collect
                                            ;positive transitions and
                                            ;HSI.1 to collect negative
                                            ;transitions
            LD PTR, #TIME_TABLE             ;Initialize time table ptr
            CLR SIZE                        ;Initialize counter
            LDB IOC0, #00000101B            ;Enable HSI.0 and HSI.1

;Save first valid transition time and enable interrupts

WAIT1:      JBC IOS1, 7, WAIT1              ;Wait for capture
            CLR HSI_TIME                    ;Dump 1st capture - invalid
WAIT2:      JBC IOS1, 7, WAIT2              ;Wait for next capture
            LD EVENT1, HSI_TIME             ;Save transition time
            EI                              ;Enable interrupts
            BR $                            ;Wait here for interrupt

;HSI Data Available Interrupt Service Routine - Calculates the time
;between consecutive transitions, and stores the times in TIME_TABLE

CAL_TIME:   PUSHA                           ;Disable interrupts
            LD EVENT2, HSI_TIME             ;Collect transition time
            SUB TIME, EVENT2, EVENT1        ;Calculate time between
                                            ;transitions
            ST TIME, [PTR]+                 ;Store in time table
            LD EVENT1, EVENT2
            INC SIZE                        ;Increment storage counter
            CMP SIZE, #TABLE_SIZE           ;If size and table_size are
            BNE RETURN                      ;not equal goto return
            CLRB IOC0                       ;Disable HSI.0 and HSI.1
RETURN:     POPA                            ;Enable interrupts
            RET                             ;Return
END                                         ;The last line must be end
```

272116–34

12.3 Example 3—Using the High Speed Input Unit and the Pulse Width Modulation Output

As in the previous example, Example 3 shown in Listing 12-3 monitors a signal and captures positive and negative transition times. HSI.0 captures positive transition times and HSI.1 captures negative transition times. For every three consecutive transitions the software calculates the low time percentage. The program then generates a PWM output with a duty cycle equal to the low time percentage. Figure 12-1 shows various input signals and the resulting PWM outputs.

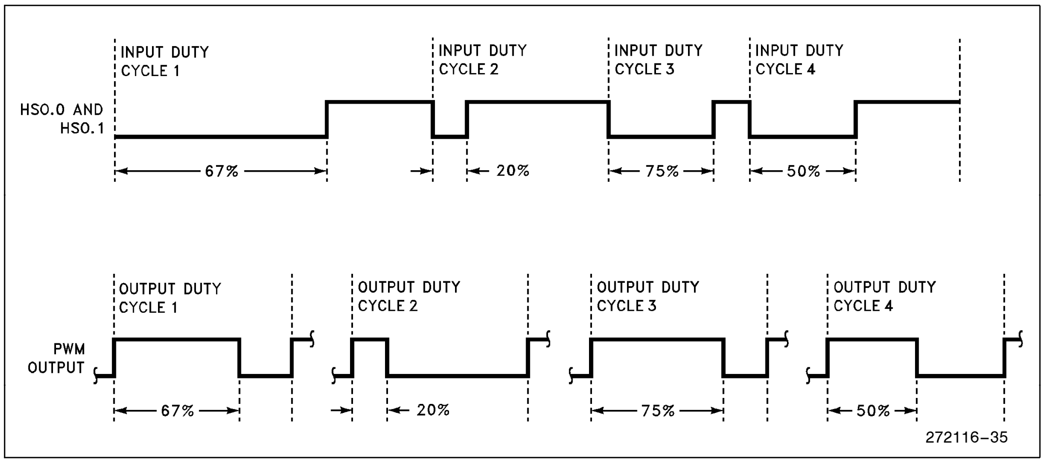


Figure 12-1. Example Input Signals and the Resulting PWM Outputs

Listing 12-3.
Using the High Speed Input Unit and The Pulse Width
Modulation Output--HSIB.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;**********************************************************************
; HSIB.A96 - Using The High Speed Input Unit
;**********************************************************************
HSIB   MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:INTEL\INCLUDE\80C196.INC)
$LIST

;HSI.0 and HSI.1 monitor the same signal, HSI.0 captures the time of
;every positive transition while HSI.1 captures the time of every
;negative transition.  Events 1, 2 and 3 store the times of each three
;consecutive transitions.  The time values are based on timer1.
;8 state times = 1 timer1 count.

RSEG AT 30
RSV:        DSW     8                       ;Reserved space in RISM
TIME1:      DSL     1                       ;Low time
TIME2:      DSW     1                       ;Period
EVENT1:     DSW     1                       ;Low transition time
EVENT2:     DSW     1                       ;High transition time
EVENT3:     DSW     1                       ;Low transition time

CSEG AT 2004H
DCW         CAL_TIME                        ;Address of HSI DATA
                                            ;AVAILABLE
                                            ;Interrupt Service Routine
CSEG AT 2080H
;Load the stack pointer, disable interrupts, disable HSI.0 and HSI.1

INIT:       LD SP, #100H                    ;Load stack pointer
            DI                              ;Disable interrupts
            LDB INT_MASK, #00000100B        ;Unmask HSI DATA AVAILABLE
            CLRB IMASK1                     ;Mask all other interrupts
            LDB IOC1, #10000001B            ;Bit 0=1 selects PWM output
                                            ;Bit 7=1 selects HSI Data
                                            ;Available Interrupt
            CLRB IOC0                       ;Disable HSI.0 and HSI.1

;Flush the FIFO, set-up HSI.0 to capture positive transitions and HSI.1
;to capture negative transitions and enable the HSI pins

FLUSH:      CLR HSI_TIME                    ;Unload one FIFO level
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
            JBS IOS1, 7, FLUSH              ;If FIFO is not empty goto
                                            ;flush
            LDB HSI_MODE, #00001001B        ;Set HSI.0 to collect
                                            ;positive transitions and
                                            ;HSI.1 to collect negative
                                            ;transitions
            LDB IOC0, #00000101B            ;Enable HSI.0 and HSI.1

;Event1 stores the time of the first valid negative transition.  Event2
;stores the next positive transition.

WAIT1:      JBC IOS1, 7, WAIT1              ;Wait for first capture
DUMP:       CLR HSI_TIME                    ;Dump capture
WAIT2:      JBC IOS1, 7, WAIT2              ;Wait for next capture
            JBS HSI_STATUS, 0, DUMP         ;If pos transition goto dump
            LD EVENT1, HSI_TIME             ;Save neg transition time
WAIT3:      JBC IOS1, 7, WAIT3              ;Wait for next capture
            LD EVENT2, HSI_TIME             ;Save time pos transition
            EI                              ;Enable interrupts
            BR $
;HSI Data Available Interrupt Service Routine - calculates the low time
;period, % low time and outputs a PWM.  Low time is the difference
;between every positive transition and the preceding negative transition
;Period is the difference between every negative transition.
;% low time = low time / period.  % low time is then used as the duty
;cycle for a PWM output.

CAL_TIME:   PUSHA                           ;Disables interrupts
            LD EVENT3, HSI_TIME             ;Save neg transition time
            SUB TIME1, EVENT2, EVENT1       ;Calculate low time
            SUB TIME2, EVENT3, EVENT1       ;Calculate period
            LD EVENT1, EVENT3
WAIT4:      JBC IOS1, 7, WAIT4              ;Wait for next capture
            LD EVENT2, HSI_TIME             ;Save pos transition time
CAL_PWM:    MUL TIME1, #256D                ;Determine % of the signal
            DIV TIME1, TIME2                 ;that is low, set-up pwm
            LDB PWM_CONTROL, TIME1           ;with duty cycle of same %
            POPA                            ;Enable interrupts
            RET                             ;Return
END
```

272116–36

12.4 Example 4—Using the High Speed Output Unit to Generate Multiple PWMs

Example 4 shown in Listing 12-4 shows the most common way to generate multiple PWMs using the HSO Unit. This module uses pins HSO.0, HSO.1 and HSO.2 as the PWM outputs. The commands that set each PWM, clear each PWM and reset Timer2 are locked in the CAM.

Since Timer2 is the reference timer, an external source must drive it. The three PWMs generated by this program are shown in Figure 12-2. It is possible to change the waveforms of these PWMs by simply changing the EQU statements at the beginning of the program.

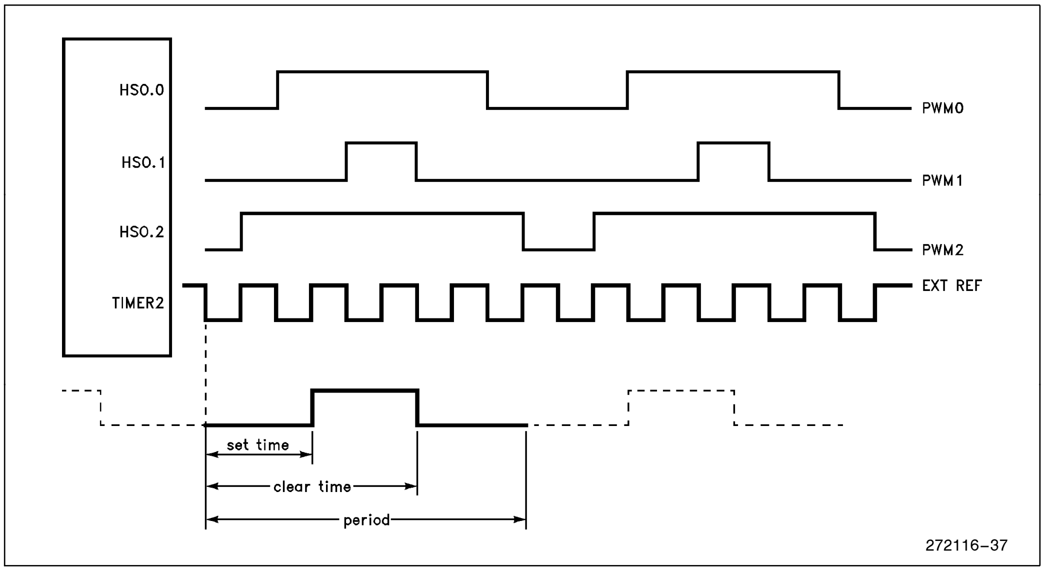


Figure 12-2. Example PWMs

Listing 12-4.
Using the High Speed Output Unit to Generate Multiple PWMs - HSOA.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;*************************************************************************
; HSOA.A96 - Using The High Speed Output Unit
;*************************************************************************
HSOA   MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:INTEL\INCLUDE\80C196.INC)
$LIST

RSEG AT 30H
RSV:        DSW     8                       ;Reserved space

;HSO.0, HSO.1 and HSO.2 are the output pins for PWM 0, PWM 1 and PWM 2
;The values below are based on Timer2. (1 = 1 Timer2 count)

PERIOD      EQU     0AH                     ;Period of the PWMs
PWM0_ST     EQU     2H                      ;PWM 0 set time
PWM0_CT     EQU     8H                      ;PWM 0 clear time
PWM1_ST     EQU     4H                      ;PWM 1 set time
PWM1_CT     EQU     6H                      ;PWM 1 clear time
PWM2_ST     EQU     1H                      ;PWM 2 set time
PWM2_CT     EQU     9D                      ;PWM 2 clear time

CSEG AT 2080H

;Load the stack pointer, clear CAM, enable the CAM locking function and
;disable interrupts

INIT:       LD SP, #100H                    ;Load stack pointer
            LDB IOC2, #11000000B            ;Clear CAM, enable locking
                                            ;function
            DI                              ;Disable interrupts

;Load CAM to output three PWM signals.  The commands are locked into
;the CAM and Timer2 which is reset every period is used as the time
;reference.

SET_0:      LDB HSO_COMMAND, #11100000B     ;Set hso.0 when
            LD HSO_TIME, #PWM0_ST           ;timer2 = pwm0_st
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
SET_1:      LDB HSO_COMMAND, #11100001B     ;Set hso.1 when
            LD HSO_TIME, #PWM1_ST           ;timer2 = pwm1_st
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
SET_2:      LDB HSO_COMMAND, #11100010B     ;Set hso.2 when
            LD HSO_TIME, #PWM2_ST           ;timer2 = pwm2_st
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
CLEAR_0:    LDB HSO_COMMAND, #11000000B     ;Clear hso.0 when
            LD HSO_TIME, #PWM0_CT           ;timer2 = pwm0_ct
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
CLEAR_1:    LDB HSO_COMMAND, #11000001B     ;Clear hso.1 when
            LD HSO_TIME, #PWM1_CT           ;timer2 = pwm1_ct
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
CLEAR_2:    LDB HSO_COMMAND, #11000010B     ;Clear hso.2 when
            LD HSO_TIME, #PWM2_CT           ;timer2 = pwm2_ct
            SKIP R0                         ;Wait 4 state times
            SKIP R0                         ;Wait 4 state times
RESET:      LDB HSO_COMMAND, #11001110B     ;Reset timer2 when
            LD HSO_TIME, #PERIOD            ;timer2 = period
            BR $
END
```

272116-38

12.5 Example 5—Using the High Speed Output Unit to Generate a Single PWM

Example 5 shown in Listing 12-5 shows another way to generate a single PWM using the HSO Unit. To use this example another module must set-up the PWM period and clear time. This program uses an HSO Interrupt (INT03) to force a call to an Interrupt Service Routine (ISR). The ISR then loads the commands in the CAM: a set command and a clear command. These CAM commands use Timer1 as the reference timer. Since this example does not use locked CAM entries, it is possible to alter the PWM output while the program is running by changing the period and clear time values.

Listing 12-5.
Using the High Speed Output Unit to Generate a Single PWM - HSOB.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;*********************************************************************
; HSOB.A96 - Using The High Speed Output Unit
;*********************************************************************
HSOB  MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:INTEL\INCLUDE\80C196.INC)
$LIST

RSEG AT 30H
RSV:        DSW     8                       ;Reserved space in RISM

;HSO.0 is the output pin for a PWM.  The signal starts high, has the
;designated PERIOD and is cleared at the designated CLEAR_TIME.  The
;PERIOD and CLEAR_TIME values are determined in another module.  These
;values are based on Timer1 (1 = 1 Timer1 count = 8 state times).

RSEG
EXTRN PERIOD, CLEAR_TIME                    ;Period and Clear time of PWM
TEMP_TIMER: DSW     1                       ;Time value used for the HSO
                                            ;set and clear commands

CSEG AT 2006H
DCW         OUTPUT                          ;Address of HSO
                                            ;Interrupt Service Routine

CSEG AT 2080H

;Load the stack pointer, clear CAM, unmask HSO interrupt,
;mask all other interrupts and enable interrupt

INIT:       LD SP, #100H                    ;Load stack pointer
            LDB IOC2, #11000010B            ;Clear the CAM
            LDB INT_MASK, #00001000B        ;Unmask HSO interrupt
            CLRB IMASK1                     ;Mask all other interrupts
            EI                              ;Enable interrupts

;Force a call to output routine

            LD TEMP_TIMER, TIMER1           ;Initialize temp_timer
            CALL OUTPUT
            BR $                            ;Wait here for interrupt

;HSO - Interrupt Service Routine to output signal using HSO.0.

OUTPUT:     PUSHA                           ;Disable interrupts
            ADD TEMP_TIMER, PERIOD          ;Temp_timer=temp_timer+period
SETO:       LDB HSO_COMMAND, #00110000B     ;Set HSO.0 when
            LD HSO_TIME, TEMP_TIMER         ;timer1 = temp_timer + period
            SKIP R0                         ;Wait four state times
            SKIP R0                         ;Wait four state times
CLEAR0:     LDB HSO_COMMAND, #00000000B     ;Clear HSO.0 when
            ADD HSO_TIME, TEMP_TIMER, CLEAR_TIME
                                            ;timer1=temp_timer + clr time
            POPA                            ;Enable interrupts
            RET                             ;Return
END
```

272116–39

12.6 Example 6—Using the A/D Converter

Example 6 shown in Listing 12-6 uses the HSO Unit to start an A/D conversion. Timer 2 is the reference timer so an external source must drive it. The value of SAMPLE_TIME determines how often the HSO will start an A/D Conversion. The Conversion Complete Interrupt forces a call to an Interrupt Service Routine (ISR). The ISR reads the A/D result and reloads the conversion command.

Listing 12-6.
Using the A/D Converter - AD.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;**********************************************************************
;   AD.A96 - Using the A/D Converter
;**********************************************************************

AD   MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:\INTEL\INCLUDE\80C196.INC)
$LIST

RSEG AT 30H
RSV:         DSW     8                  ;Reserved space in RISM

RSEG
SAMPLE_TIME EQU      170D               ;The HSO will start an A/D
                                        ;conversion every
                                        ;Timer2 = sample_time
SAMPLE:      DSW     1                  ;A/D conversion result
RESULT:      DSW     1                  ;A/D conversion result


CSEG AT 2002H
DCW          GET_RESULT                 ;Address of A/D Conversion
                                        ;Complete
                                        ;Interrupt Service Routine

;Load stack pointer, disable interrupts, enable CAM locking function,
;unmask only the A/D Conversion Complete Interrupt, and enable
;interrupts

CSEG AT 2080H
INIT:        LD   SP, #100H              ;Load stack pointer
             DI                         ;Disable interrupts
             LDB IOC2, #11000000B       ;Clear CAM, enable locking
                                        ;function
             LDB INT_MASK, #02H         ;Unmask A/D Conversion Complete
             CLRB IMASK1                ;Mask all other interrupts
             EI                         ;Enable interrupts

;Set-up an A/D conversion on A/D-channel 0 that the HSO unit will
;trigger.  Load the CAM with a command to start the A/D conversion and
;reset Timer2 every SAMPLE_TIME.

COMMAND:     LDB AD_COMMAND, #0         ;Conversion on AD0 started by
                                        ;HSO
START_CONV: LDB HSO_COMMAND, #11001111B ;AD Conversion, locked ,timer2
             LD HSO_TIME, #SAMPLE_TIME  ;Load conversion starting time
             SKIP R0                    ;Wait 4 state times
             SKIP R0                    ;Wait 4 state times
RESET_T2:    LDB HSO_COMMAND, #11001110B ;Reset timer2 when
             LD HSO_TIME, #SAMPLE_TIME  ;timer2 = sample_time
             BR $                       ;Wait here for interrupt

;A/D Conversion Complete Interrupt Service Routine - Read the AD_RESULT
;registers and reload the AD_COMMAND register

GET_RESULT: PUSHA                       ;Disable interrupts
             LD RESULT, AD_RESULT_LO    ;Load ad_result (result 10 MSBs
             SHR RESULT, #6             ;Shift result (result 10 LSBs)
             LD SAMPLE, RESULT          ;Save result in sample
             LDB AD_COMMAND, #0         ;Conversion on AD0 started by
                                        ;HSO
             POPA                       ;Enable interrupts
             RET                        ;Return
END
```

272116-40

12.7 Example 7—Using the Serial Port

Example 7 shown in Listing 12-7 uses Mode 1, the standard asynchronous mode of the 80C196KB Serial Port, to transmit the message "hello". The program transmits the first byte of the message then enables the TI (Transmit Interrupt). The transmission of the last data bit of the message byte causes a TI. The TI forces a call to an Interrupt Service Routine. The ISR then transmits the next byte of the message and the program control is returned to the main program where it waits for the next TI.

Listing 12-7.
Using the Serial Port - SP.A96

```
$DEBUG ERRORPRINT NOSYMBOLS
;*********************************************************************
;   SP.A96 - Using the Serial Port
;*********************************************************************
Serial_P  MODULE MAIN,STACKSIZE(20)
$NOLIST INCLUDE (C:\INTEL\INCLUDE\80C196.INC)
$LIST

RSEG AT 30H
RSV:        DSW 8                       ;Reserved Space in RISM

RSEG
MESS_PTR:   DSW 1                       ;Pointer into message

CSEG AT 2030H
DCW         TRANSMIT                    ;Address of TI
                                        ;Interrupt Service Routine
CSEG
MESSAGE1:   DCB         'HELLO'         ;Message to transmit
            DCB         0DH,0AH         ;Carriage return, line feed
MESS1_END   EQU         $               ;Address of end of message

CSEG AT 2080H

;Load stack pointer, enable TxD pin, unmask TI interrupt, mask all
;other interrupts, set serial port to mode 1, load the baud value for a
;baud rate of 9600 at 12 Mhz and set MSB of baud value to indicate that
;XTAL1 is the clock soure for the baud rate generator

INIT:       LD SP, #100H                ;Load stack pointer
            DI                          ;Disable interrupts
            LD IOC1, #20H               ;Enable TxD pin
                LDB IMASK1, #1              ;Unmask TI interrupt
            CLRB INT_MASK               ;Mask all other interrupts
            LDB SP_CON, #1              ;Set serial port to mode 1
            LDB BAUD_RATE, #77D         ;Load least significant byte of
                                        ;baud value
            LDB BAUD_RATE, #80H         ;Load most significant byte of
                                        ;baud value with MSB = 1 to
                                        ;indicate XTAL1 as the baud
                                        ;rate source
            EI                          ;Enable interrupts

;Send the first byte of message1, the following message bytes are sent
;in the TI interrupt service routine

            LD MESS_PTR, #MESSAGE1      ;Mess_ptr = address of message1
WAIT:       JBC SP_STAT, 3, WAIT        ;Wait till serial buffer is
                                        ;empty
            LDB SBUF, [MESS_PTR]+       ;Send first byte of message
            BR $                        ;Wait here for TI interrupt

;TI Interrupt Service Routine - Check for end of message, if not at
;end of message send the next byte of message1

TRANSMIT:   PUSHA                       ;Disable interrupts
            CMP MESS_PTR, #MESS1_END    ;If at message end goto return
            BE RETURN
            LDB SBUF, [MESS_PTR]+       ;Load next byte of message1
RETURN:     POPA                        ;Enable interrupts
            RET                         ;Return

END                                     ;Last line must be end
```

272116-41

13.0 HARDWARE EXAMPLES

Several different combinations of addressing and data bus modes exist on the 80C196KB. External memory is addressable through the AD0–AD15 lines. These lines form a multiplexed 16-bit address and data bus. The standard data bus mode uses a 16-bit data bus. Other data bus modes include an 8-bit only external bus mode, and a data bus mode that can switch dynamically between 8 bits and 16 bits. The address bus is always 16 bits wide. The address/data bus shares pins with ports 3 and 4.

An 8-bit system with EPROM and RAM is shown in Figure 13-1. The EPROM is addressable through the lower half of memory, and the RAM is addressable through the upper half. The diagram in Figure 13-2 shows a simple 16-bit system with 2 EPROMs. The first EPROM contains the even bytes while the second EPROM contains the odd bytes. Shown in Figure 13-3 is a system using the dynamic bus width. The two EPROMs contain the executable code and the single RAM provides for external data storage. Note the system uses AD15 as the Chip Select, and as input to the BUSWIDTH pin to select an 8-bit cycle.

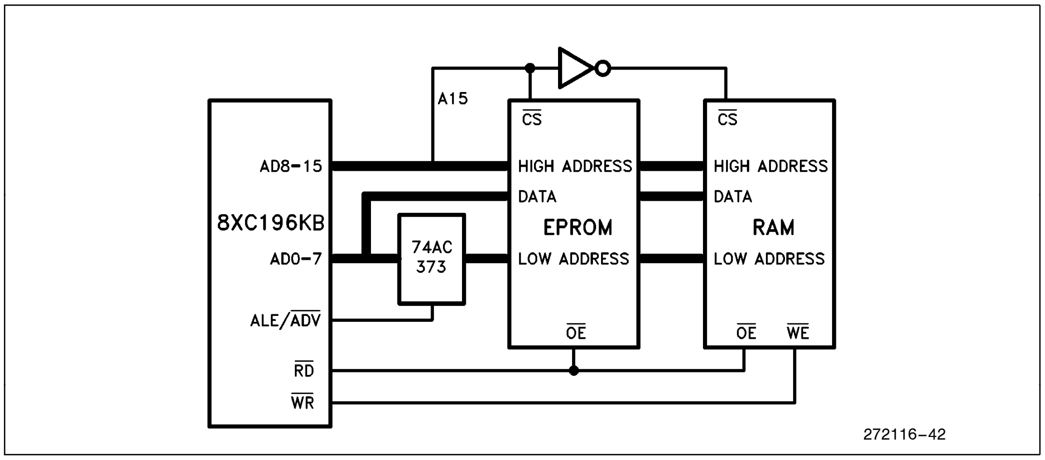


Figure 13-1. 8-Bit System with EPROM and RAM

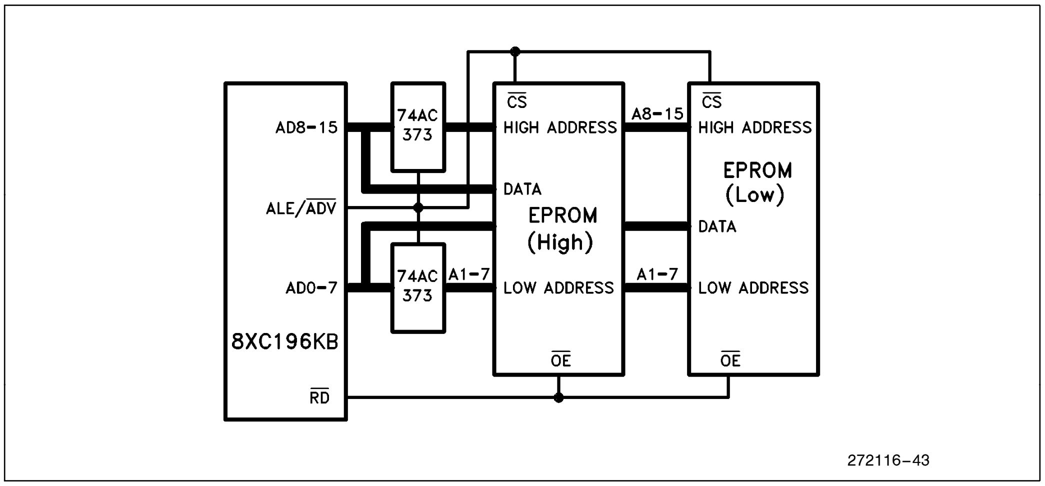


Figure 13-2. 16-Bit System with EPROM

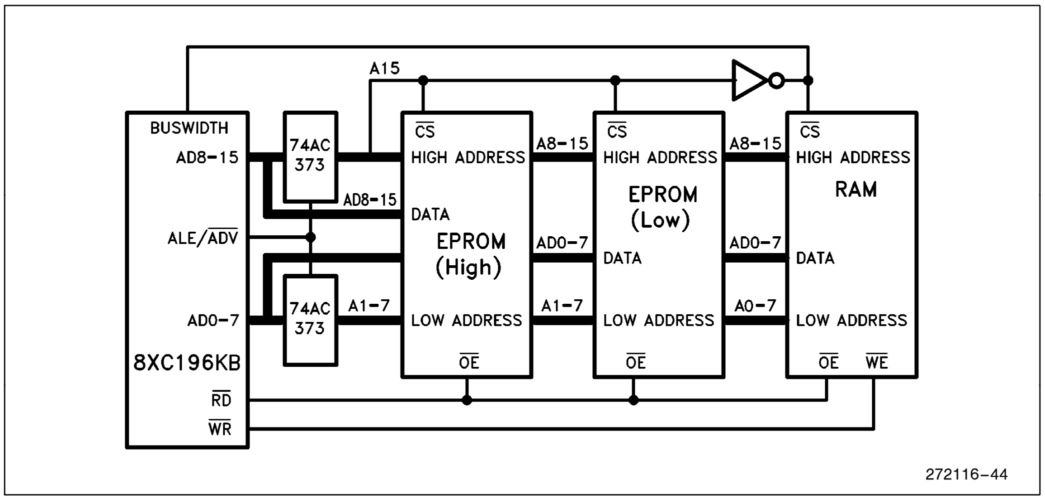


Figure 13-3. 16-Bit System with Dynamic Bus Width

Figure 13-4 is a schematic of a typical minimal system using the 80C196KB. The address/data bus is demultiplexed by latches U3 and U4. U5 and U7 are connected as odd and even bytes of 16-bit wide EPROM. Note that bus address line A0 is not required for these devices, as they are always addressed in increments of 2 bytes (A0 is a don't care). U6 is a byte-wide RAM chip.

Address mapping and BUSWIDTH is determined by U8, a PAL. The RESET signal is generated by an R–C network, U2A, U2B and D2. U2 is a Schmitt trigger which generates a fast edge from the R–C network slow rise time. D2 is used as a "wired or" gate to the $\overline{\text{RESET}}$ pin on the 80C196KB. This is necessary to avoid signal contention as the $\overline{\text{RESET}}$ pin is both an input and an output from the 80C196KB.

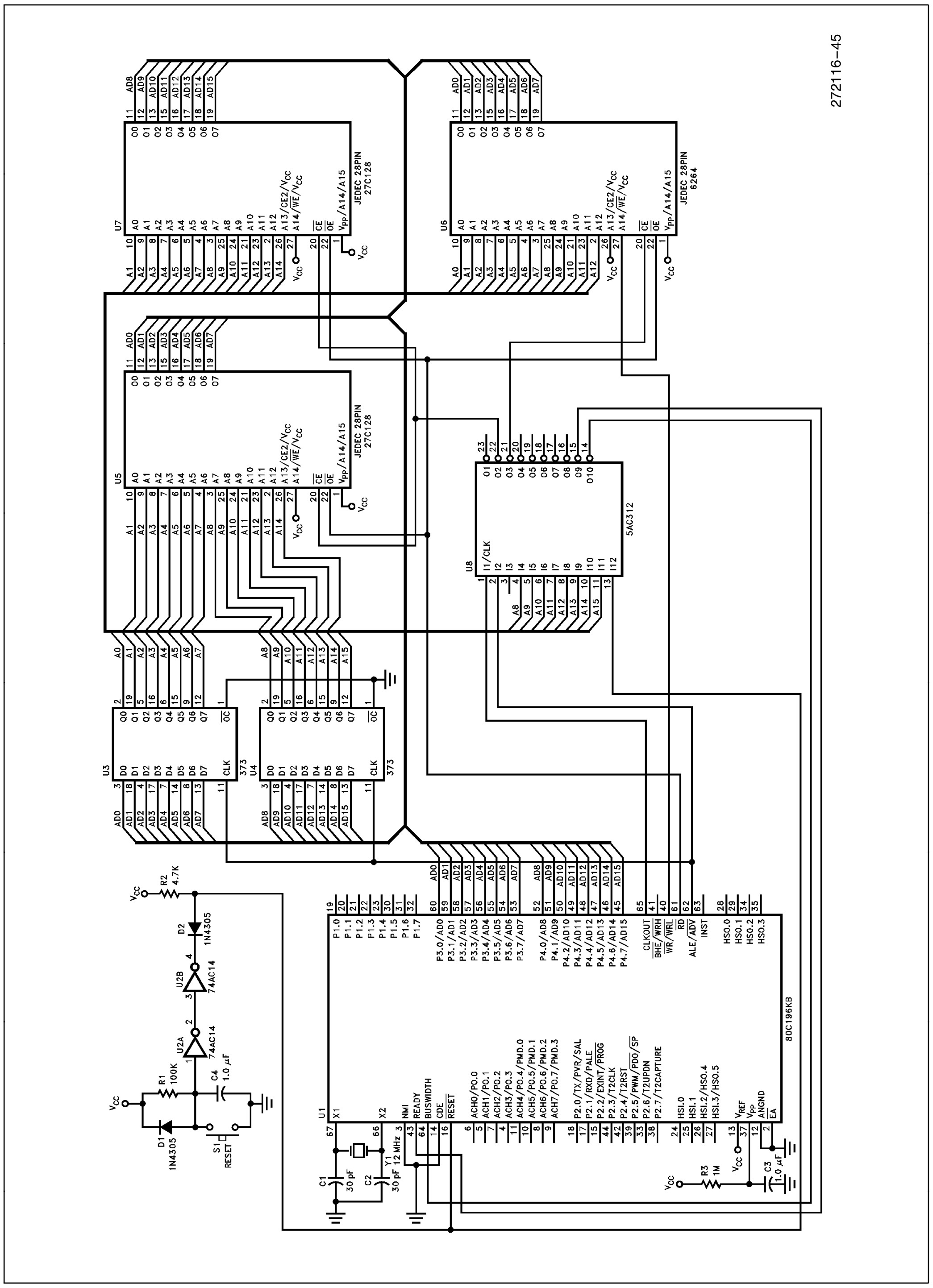


Figure 13-4. Schematic of 16-Bit System with Dynamic Bus Width

14.0 PORT RECONSTRUCTION

External memory systems for the 80C196KB use a multiplexed address/data bus (AD0–AD15) which shares pins with I/O Ports 3 and 4. Ports 3 and 4 are read and written at locations 1FFEH and 1FFFH. If $\overline{EA}$ is high, accessing locations 1FFEH and 1FFFH reads/writes to Ports 3 and 4. However, if $\overline{EA}$ is low, locations 1FFEH and 1FFFH act as memory locations, not ports. Thus, to use Ports 3 and 4 under these conditions requires a port reconstruction circuit. Shown in Figure 14-1 is a port reconstruction circuit that uses a memory mapped I/O technique.

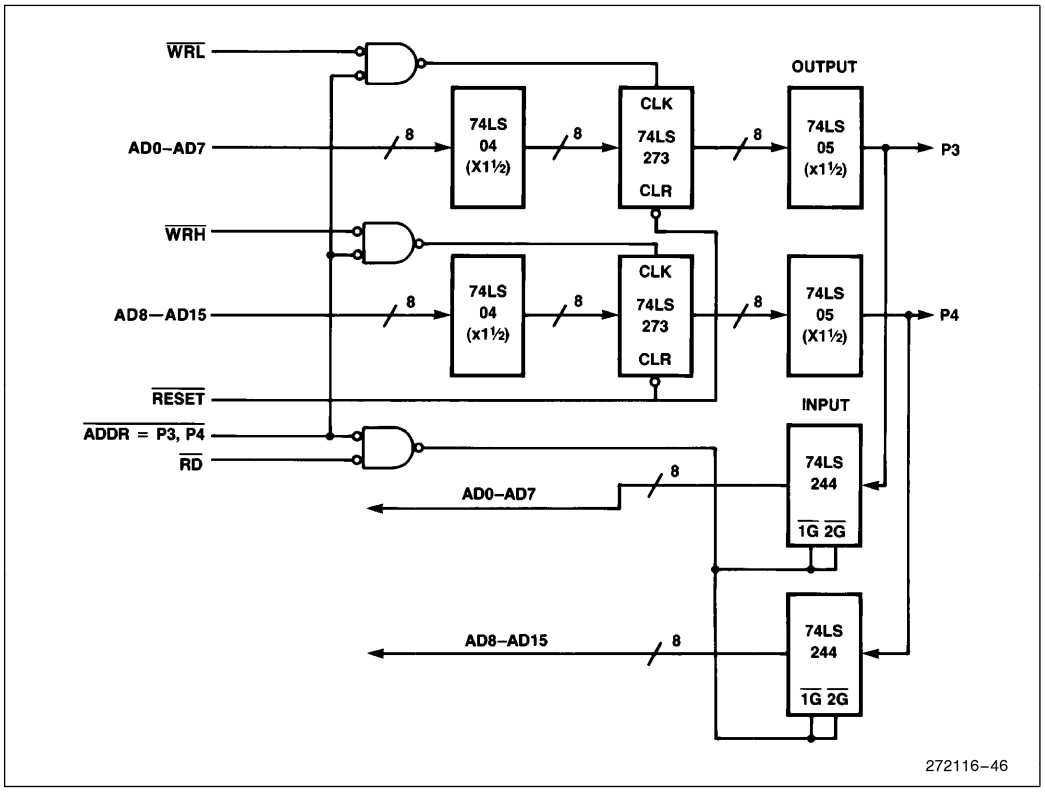


Figure 14-1. I/O Port Reconstruction

15.0 CONCLUSION

This application note presented an overview of the 80C196KB and provided software examples using its key features. Intel supports application development of its 80C196KB with a complete set of development languages and utilities. These tools include ACE196, a macroassembler (ASM96), a PL/M compiler (PLM96), a C compiler (iC96), linker/relocator program (RL96), floating point arithmetic library utility (FPAL96), a librarian utility (LIB96) and object-to-hex utility (OH96). ACE196 software is a PC-based expert system to guide you through detailed documentation training and includes: a hypertext manual, peripheral design modules and an assembler editor. Contact your local sales office for more information on the 80C196KB and its hardware and software development tools.

BIBLIOGRAPHY

1. "1991 16-Bit Embedded Controllers Handbook", Intel Corporation, 1990. Order Number 270646-003.
2. "1991 Embedded Applications Handbook", Intel Corporation, 1990. Order Number 270648-003.
3. AP-248, "Using the 8096, 1991 Embedded Applications Handbook", Intel Corporation, September 1987. Order Number 270648-003.
4. "MCS-96 Macro Assembler User's Guide for DOS Systems", Intel Corporation, 1990. Order Number 122350.
5. "iC-96 Compiler User's Guide for DOS Systems", Intel Corporation, 1990. Order Number 481194.