



AP-483

**APPLICATION
NOTE**

Application Examples Using the 8XC196MC/MD Microcontroller

**MASANORI DOI
GARY HARRIS
EMBEDDED CONTROL**

August 1993

Order Number: 272282-001

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

*Third-party brands and names are the property of their respective owners.

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

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

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

Table of Contents

CONTENTS	PAGE
1.0 UNDERSTANDING THE 8XC196MC/MD INTERRUPTS AND PTS	2
1.1 Interrupt Processing	2
1.1.1 Interrupt Controller	2
1.1.2 Peripheral Transaction Server (PTS)	2
1.2 Interrupt Latency	4
1.2.1 Calculating Latency	5
1.2.2 Standard Interrupt Latency	5
1.2.3 PTS Interrupt Latency	5
1.3 Interrupt Priorities	6
1.3.1 Modifying Interrupt Priorities	7
1.4 End-of-PTS Interrupts	7
1.5 Special Interrupts	8
1.5.1 Unimplemented Opcode	8
1.5.2 Software Trap	8
1.5.3 NMI	8
1.6 EPA Interrupts	8
1.6.1 Using the PTS to Service the EPA	9
1.7 Programming the Interrupts	9
1.7.1 Selecting Either PTS or Standard Interrupt Service	10
1.7.2 Enabling PTS Interrupts	10
1.7.3 Enabling Standard Interrupts	10
1.7.4 Interrupt Mask Registers	11
1.7.5 Interrupt Pending Registers	11
1.7.6 The Peripheral Interrupt (PI)	12
1.7.6.1 WG Interrupt	13
1.7.6.2 COMP5 Interrupt	13
1.7.7 The EXTINT Interrupt	13
1.7.8 The TOVF Interrupt	13
1.8 Configuring the PTS	14
1.8.1 Setting Up PTS Control Blocks	14
1.8.1.1 PTSCOUNT Register	14
1.8.1.2 PTSCON Register	14
1.8.2 Single Transfer Mode	16
1.8.2.1 Single Transfer Mode Example	16
1.8.3 Block Transfer Mode	16
1.8.3.1 Block Transfer Mode Example	17

CONTENTS

PAGE

1.8.4 A/D Scan Mode	17
1.8.4.1 PTS Cycles in A/D Scan Mode	18
1.8.4.2 A/D Scan Mode Example 1	18
1.8.4.3 A/D Scan Mode Example 2	19
1.8.5 Serial I/O Modes	19
1.8.5.1 PTSCOUNT Register	20
1.8.5.2 PTSCON Register	20
1.8.5.3 EPAREG Register	21
1.8.5.4 BAUD Register	21
1.8.5.5 PTSVEC1 Register	21
1.8.5.6 PORTREG Register	22
1.8.5.7 PORTMASK Register	22
1.8.5.8 PTSCON1 Register	22
1.8.5.9 DATA Register	23
1.8.5.10 SAMPTIME Register	23
1.8.6 PTS Asynchronous SIO Receive Mode	23
1.8.6.1 PTS Asynchronous SIO Transmit Mode	24
1.8.6.2 PTS Synchronous SIO Receive Mode	24
1.8.7 PTS Synchronous SIO Transmit Mode	25
2.0 SYNCHRONOUS SERIAL DATA TRANSMISSION PROGRAM EXAMPLE	26
2.1 Introduction	26
2.1.1 End-of-PTS Interrupt	26
2.2 Detailed Program Description	26
2.2.1 Constant Declarations (Lines 1–46)	26
2.2.2 Interrupt Vectors and CCB (Lines 47–66)	26
2.2.3 Main Program (Lines 67–135)	26
2.2.4 End-of-PTS Interrupt Routine	27
2.3 Top 5 Issues of Using the PTS/SSIO Transmit Mode	27
2.4 Program Example	28
3.0 SYNCHRONOUS SERIAL DATA RECEPTION PROGRAM EXAMPLE	33
3.1 Introduction	33
3.1.1 End-of-PTS Interrupt	34
3.2 Detailed Program Description	34
3.2.1 Constant Declarations (Lines 1–42)	34
3.2.2 Interrupt Vectors and CCB (Lines 43–66)	34
3.2.3 Main Program (Lines 67–135)	34
3.2.4 End-of-PTS Interrupt Routine	35
3.3 Top 5 Issues of Using the PTS/SSIO Receive Mode	35
3.4 Program Example	36
4.0 ASYNCHRONOUS SERIAL DATA TRANSMISSION PROGRAM EXAMPLE	41
4.1 Introduction	41

CONTENTS	PAGE
4.1.1 End-of-PTS Interrupt	41
4.2 Detailed Program Description	41
4.2.1 Constant Declarations (Lines 1–42)	41
4.2.2 Interrupt Vectors and CCB (Lines 43–66)	41
4.2.3 Main Program (Lines 67–135)	42
4.2.4 End-of-PTS Interrupt Routine	42
4.3 Top 5 Issues of Using the PTS/SIO Transmit Mode	43
4.4 Program Example	44
5.0 ASYNCHRONOUS SERIAL DATA RECEPTION PROGRAM SAMPLE	49
5.1 Introduction	49
5.1.1 End-of-PTS Interrupt	50
5.2 Detailed Program Description	50
5.2.1 Constant Declarations (Lines 1–42)	50
5.2.2 Interrupt Vectors and CCB (Lines 43–66)	50
5.2.3 Main Program (Lines 67–135)	50
5.2.4 End-of-PTS Interrupt Routine (Lines 133–176)	51
5.3 Top 5 Issues of Using the PTS/SIO Receive Mode	51
5.4 Program Example	52
6.0 UNDERSTANDING THE WAVEFORM GENERATOR	58
6.1 Introduction	58
6.2 Buffering	59
6.3 The Timebase Generator	60
6.3.1 Center Aligned Mode WG__COUNT Operation (Modes 0 and 1)	60
6.3.2 Edge Aligned Mode WG__COUNT Operation (Modes 2 and 3)	60
6.4 The WFG Control Register (WG__CON)	61
6.5 Center Aligned PWM Modes (Modes 0 and 1)	61
6.5.1 Power-Up Initialization	61
6.5.2 WFG Output Operation	62
6.5.3 Differences between Mode 0 and Mode 1	62
6.5.4 Formulas for Carrier Period and Duty Cycle	62
6.6 Edge Aligned PWM Modes (Modes 2 and 3)	63
6.6.1 Power-Up Initialization	63
6.6.2 WFG Output Operation	63
6.6.3 Differences between Modes 2 and 3	64
6.6.4 Formulas for Carrier Frequency and Duty Cycle	64
6.7 Interrupt Generation	64
6.7.1 The WG Interrupt	64
6.7.1.1 WG Interrupt Generation	65
6.7.2 The EXTINT Interrupt	65

CONTENTS	PAGE
6.8 Dead-Time Generator	66
6.8.1 Dead-Time Generator Operation	66
6.8.2 Dead-Time Calculation	67
6.8.3 Effect of Dead-Time on PWM Duty Cycle	67
6.9 Output Control Circuitry	67
6.9.1 PH Control Bits	68
6.9.2 Enabling and Disabling the WFG Outputs	68
6.9.3 Output Polarity	69
6.9.4 Output Schematic	69
6.10 Protection Circuitry	71
6.10.1 Protection Circuitry Operation	71
6.11 Software Example	72
7.0 3-PHASE INDUCTION MOTOR CONTROL PROGRAM SAMPLE	76
7.1 Introduction	76
7.1.1 Sine Look-Up Table	77
7.1.2 Frequency-to-Voltage Look-Up Table	78
7.1.3 Output Waveforms	79
7.2 Detailed Program Description	80
7.2.1 Constant Declarations (Lines 1–42)	80
7.2.2 Interrupt Vectors and CCB (Lines 43–57)	80
7.2.3 Main Program (Lines 58–93)	80
7.2.4 Polling Routine (Lines 94–126)	80
7.2.5 WG Interrupt Routine (Lines 127–211)	81
7.2.6 EXTINT Interrupt Routine (Lines 212–223)	81
7.2.7 Sine Look-Up Table (Lines 224–1429)	81
7.2.8 RAMDA Look-Up Table (Lines 1430–2235)	81
7.3 Top 5 Issues for the 3-Phase Induction Motor Control Example	81
7.4 Program Example	82
8.0 3-PHASE DC BRUSHLESS MOTOR CONTROL PROGRAM SAMPLE	90
8.1 Introduction	90
8.1.1 Drive Waveforms	91
8.1.2 Frequency-to-Voltage Look-Up Table	92
8.2 Detailed Program Description	93
8.2.1 Constant Declarations (Lines 1–27)	93
8.2.2 Interrupt Vectors and CCB (Lines 28–48)	93
8.2.3 Main Program Initialization (Lines 49–97)	93
8.2.4 Polling Routine (Lines 98–134)	94
8.2.5 WG Interrupt Routine (Lines 135–173)	94
8.2.6 CAPCOMP0 Interrupt Routine (Lines 174–225)	94
8.2.7 EXTINT Interrupt Routine (Lines 226–237)	95
8.2.8 Amplitude Look-Up Table (Lines 244–1044)	95

CONTENTS	PAGE
8.3 Top 4 Issues for the DC Brushless Motor Control Example	95
8.4 Program Example	96
9.0 QUADRI PHASE STEPPING MOTOR CONTROL PROGRAM SAMPLE	104
9.1 Introduction	104
9.1.1 Drive Waveforms	104
9.2 Detailed Program Description	107
9.2.1 Constant Declarations (Lines 1–29)	107
9.2.2 Interrupt Vectors and CCB (Lines 29–49)	107
9.2.3 Main Program Initialization (Lines 50–95)	107
9.2.4 Polling Routine (Lines 96–134)	107
9.2.5 CAPCOMP0 Interrupt Routine (Lines 131–186)	108
9.2.6 EXTINT Interrupt Routine (Lines 187–198)	108
9.2.7 Amplitude Look-Up Table (Lines 199–1005)	108
9.3 Top 3 Issues for the 4-Phase Stepper Motor Control Example	108
9.4 Program Example	109
10.0 THE FREQUENCY GENERATOR (MD ONLY)	115
10.1 Introduction	115
10.2 Using the Frequency Generator	116
10.3 Program to Transmit Data via Frequency Generator	118
10.3.1 Constant and Variable Declarations (Lines 1–81)	118
10.3.2 Main Program (Lines 82–132)	118
10.3.3 Compare3 Interrupt Routine	118
10.4 Program Example	119

CONTENTS

PAGE

List of Figures

Figure 1-1	Flow Diagram for PTS and Standard Interrupts	3
Figure 1-2	Standard Interrupt Response Time	5
Figure 1-3	PTS Interrupt Response Time	6
Figure 1-4	PTSSEL Register	10
Figure 1-5	INT__MASK and INT__MASK1 Registers	11
Figure 1-6	INT__PEND and INT__PEND1 Registers	11
Figure 1-7	PI Interrupt Sharing	12
Figure 1-8	PTS Control Blocks	14
Figure 1-9	PTSCON Register	15
Figure 1-10	PTS SIO Control Blocks	20
Figure 1-11	PTSCON Register in SIO Modes	20
Figure 1-12	PTSCON1 in Asynchronous Mode	22
Figure 1-13	PTSCON1 in Synchronous Mode	23
Figure 1-14	Asynchronous SIO Receive Mode Timing	24
Figure 1-15	Asynchronous SIO Transmit Mode Timing	24
Figure 1-16	Synchronous SIO Receive Mode Timing	25
Figure 1-17	Synchronous SIO Transmit Mode Timing	25
Figure 2-1	Flow Chart—SSIO Transmit Initialization	32
Figure 2-2	Flow Chart—SSIO Transmit Interrupt Routine	33
Figure 3-1	Flow Chart—SSIO Receive Initialization	39
Figure 3-2	Flow Chart—SSIO Receive Interrupt Routine	39
Figure 4-1	Flow Chart—ASIO Transmit Initialization	48
Figure 4-2	ASIO Transmit Interrupt Routine	49
Figure 5-1	Flow Chart—ASIO Receive Initialization	56
Figure 5-2	Flow Chart—ASIO Receive Interrupt Routine	57
Figure 6-1	Simplified WFG Block Diagram	58
Figure 6-2	Modes 0 and 1 Counter Operation	60
Figure 6-3	Modes 2 and 3 Counter Operation	61
Figure 6-4	WG__CON Register	61
Figure 6-5	WG__COUNT during Center Aligned PWM	62
Figure 6-6	WG__COUNT during Edge Aligned PWM	63
Figure 6-7	PI Interrupt Sharing	65
Figure 6-8	Dead-Time Generation	66
Figure 6-9	Effect of Dead-Time on WFG Duty Cycle	67
Figure 6-10	WG__OUT Register	68
Figure 6-11	WFG Output Schematic	70
Figure 6-12	Protection Circuitry	71
Figure 6-13	The WG__PROTECT Register	72

CONTENTS

PAGE

List of Figures

Figure 7-1	3-Phase Induction Motor System	77
Figure 7-2	Sine Look-Up Table	77
Figure 7-3	Raw Pointer Values	78
Figure 7-4	Modulation Depth	78
Figure 7-5	WFG Output Drive	79
Figure 7-6	WFG Output Waveform after Low-Pass Filter	79
Figure 7-7	Flow Chart—3-Phase A.C. Induction Motor Initialization	87
Figure 7-8	Flow Chart—3-Phase A.C. Induction Motor WG Interrupt Routine	88
Figure 8-1	3-Phase DC Brushless Motor System	90
Figure 8-2	DC Brushless Phase Drive Waveforms	91
Figure 8-3	WFG Output Waveform after Low-Pass Filter	92
Figure 8-4	Amplitude vs Modulation Frequency Function	92
Figure 8-5	Flow Chart—3-Phase D.C. Brushless Motor Initialization	101
Figure 8-6	Flow Chart—3-Phase D.C. Brushless Motor WG_COUNT Interrupt	102
Figure 8-7	Flow Chart—3-Phase D.C. Brushless Motor EPA Interrupt	103
Figure 9-1	4-Phase Stepper Motor System	104
Figure 9-2	4-Phase Stepper Drive Waveforms	104
Figure 9-3	Average Voltage Output Waveform at Motor	105
Figure 9-4	Amplitude vs Modulation Frequency Function	106
Figure 9-5	Flow Chart—4-Phase Stepper Motor Initialization	113
Figure 9-6	Flow Chart—4-Phase Stepper Motor EPA Interrupt	114
Figure 10-1	The Frequency Generator	115
Figure 10-2	FREQ_GEN Register	116
Figure 10-3	FREQ_CNT Register	116
Figure 10-4	Infrared Remote Control Block Diagram	116
Figure 10-5	Encoding of Zeros and Ones	117
Figure 10-6	Flow Chart—Frequency Generator Initialization	123
Figure 10-7	Interrupt Routine Flow Chart	124

CONTENTS

PAGE

List of Tables

Table 1-1	Interrupt Sources	6
Table 1-2	Interrupt and PTS Control and Status Registers	9
Table 1-3	EXTINT Mode Selection	13
Table 1-4	PTS Mode Select (PTSCON Bits 5–7)	14
Table 1-5	PTSCON Bits 0–4 for Single and Block Transfer Modes	15
Table 1-6	PTSCON Bits 0–4 for A/D Scan Mode	15
Table 1-7	Single Transfer Mode PTSCB	16
Table 1-8	Block Transfer Mode PTSCB	17
Table 1-9	A/D Scan Mode Command/Data Table	17
Table 1-10	Command/Data Table (Example 1)	18
Table 1-11	A/D Scan Mode PTSCB (Example 1)	18
Table 1-12	Command/Data Table (Example 2)	19
Table 1-13	A/D Scan Mode PTSCB (Example 2)	19
Table 1-14	PTSCON Bits 0–4 for SIO Modes	21
Table 1-15	PTSCON Bits 0–7 for Asynchronous SIO Mode	22
Table 1-16	PTSCON Bits 0–7 for Synchronous SIO Mode	23
Table 6-1	Conditions for Register Updates	59
Table 6-2	EXTINT Mode Selection	65
Table 6-3	WG Output Configuration	68
Table 6-4	Protection and Output Status	69
Table 6-5	Output Polarities, Modes 4–7	69
Table 8-1	WFG Output Sequencing	91
Table 9-1	WFG Output Sequencing	105

OVERVIEW

This application note gives several practical software examples using the unique features of the 8XC196MC micro-controller. Each section uses one of the 8XC196MC features to perform a “typical” application. A brief explanation of the peripheral used is given, followed by detailed analysis of a software example. The program source code is listed, along with figures and flow charts.

This application note is meant to be used in conjunction with the *8XC196MC USER'S MANUAL*, order number 272181.

The program examples were developed for several different hardware configurations, and may need to be modified for the end user's application.

All programs were assembled using the standard Intel ASM-96 assembler, along with the MC include file for register names. Additional include files are required to support the windowed register names and look-up tables. The program source code is available from the Intel bulletin board system:

The Intel Applications Bulletin Board System
(916) 356-3600 or (916) 356-3605

1200, 2400 Bps
8 DATA BITS
NO PARITY
1 STOP BIT

and

9600 Bps v.32, v.42
8 DATA BITS
NO PARITY
1 STOP BIT

All 8XC196MC files are under the embedded directory.

1.0 UNDERSTANDING THE 8XC196MC/MD INTERRUPTS AND PTS

A microcontroller's primary function is to provide real-time control of an instrument or device. The interrupt control circuitry within a microcontroller permits real-time events to control program flow. When an event generates an interrupt, the CPU services the interrupt before executing the next instruction. An internal peripheral, an external signal, or an instruction can request an interrupt. In the simplest case, the device receives the request, performs the service, and returns to the task that was interrupted.

1.1 Interrupt Processing

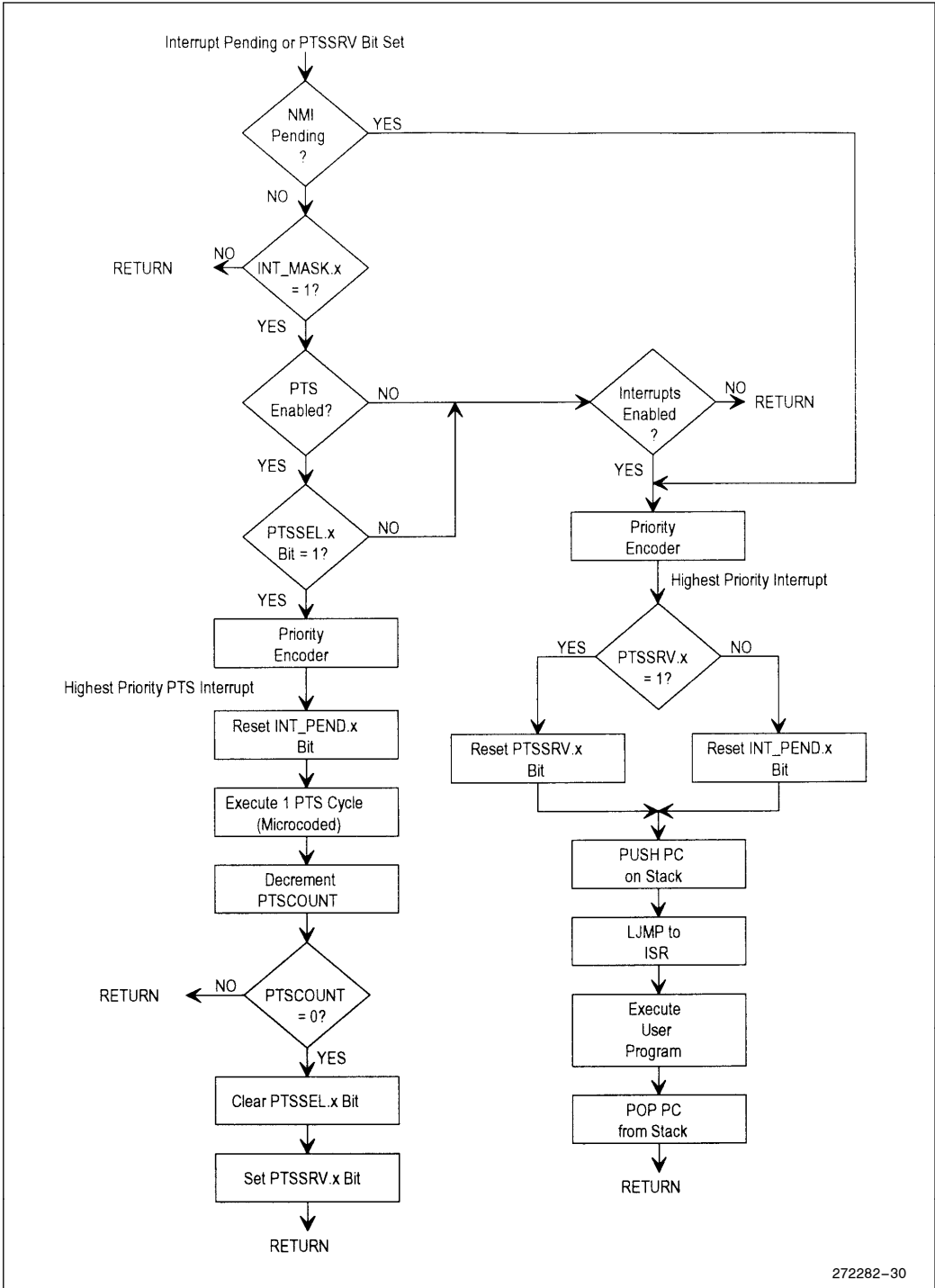
There are two options for interrupt service: software interrupt service routines via the Interrupt Controller and microcoded hardware interrupt processing via the Peripheral Transaction Server (PTS). You can select either option for each of the maskable interrupts. The nonmaskable interrupts (NMI, Software Trap, and Unimplemented Opcode) are always serviced by interrupt service routines. Figure 1-1 illustrates the interrupt processing flow.

1.1.1 INTERRUPT CONTROLLER

The Interrupt Controller generates vectors to software interrupt service routines. When the hardware detects an interrupt, it generates and executes an interrupt call. This pushes the contents of the program counter onto the stack and then loads the program counter with the contents of the appropriate interrupt vector. The CPU vectors to the address of the interrupt service routine, then executes the routine. Upon completion of the interrupt service routine, the program counter is reloaded from the stack and program execution continues.

1.1.2 PERIPHERAL TRANSACTION SERVER (PTS)

The Peripheral Transaction Server (PTS) is a microcoded hardware interrupt handler. It can be used in place of a standard interrupt service routine for each of the maskable interrupts. The PTS services interrupts with less overhead; it does not modify the stack or the PSW, and it allows normal instruction flow to continue. For these reasons, the PTS can service an interrupt in the time required to execute a single instruction.



272282-30

Figure 1-1. Flow Diagram for PTS and Standard Interrupts

The 8XC196MC/MD PTS operates in four special microcoded modes that enable it to complete specific tasks in much less time than an equivalent interrupt service routine can. See Section 1.8 for a description of the PTS modes.

Each PTS interrupt requires a block of data called the PTS Control Block (PTSCB). The PTS vector table entries point to the PTSCB. When a PTS interrupt occurs, the Priority Encoder selects the appropriate vector and fetches the PTS Control Block (PTSCB). The PTSCB determines the mode, the number of transfers per cycle (if applicable), the total number of cycles that will execute before the PTS requires servicing, and the source and/or destination of data transfers (if applicable). Each PTS interrupt generates one PTS cycle.

NOTE:

The PTSCB must be located in register RAM, and must be aligned on a quad-word boundary (evenly divisible by eight).

1.2 Interrupt Latency

Interrupt latency is the total delay between the time that the interrupt is generated and the time that the device begins executing the interrupt service routine or PTS cycle. A delay occurs between the time that the interrupt is generated and the time that it is acknowledged.

Acknowledgment is defined as when the CPU clears the interrupt pending bit that initiated the interrupt. If the interrupt occurs earlier than four state times before the end of the current instruction, it may not be acknowledged until after the next instruction finishes. This additional delay occurs because instructions are prefetched and prepared a few state times before they are executed. Thus, the maximum delay between interrupt generation and acknowledgment is four state times plus the execution time of the next instruction.

When a standard interrupt is acknowledged, the hardware clears the interrupt pending bit and forces a call to the address contained in the corresponding interrupt vector after completing the current instruction. The procedure that gets the vector and forces the call requires 11 state times. If the stack is in 16-bit 0 wait-state external RAM, the call requires an additional 2 state times. When a PTS interrupt is acknowledged, it immediately vectors to the PTSCB and begins executing the PTS cycle.

Execution of any of the following inhibits interrupts from being acknowledged until after the **next** instruction is executed:

- the signed prefix opcode (FE) for the two-byte, signed multiply and divide instructions
- the Unimplemented Opcode interrupt
- the Software Trap interrupt
- any PTS cycle (back-to-back PTS cycles are not possible)
- any of these eight *protected instructions*: DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, PUSHF.

All of these increase latency because an interrupt cannot occur until after the next instruction is executed.

1.2.1 CALCULATING LATENCY

The maximum latency occurs when the interrupt occurs too late for acknowledgment following the current instruction. The following worst-case calculation assumes that the current instruction is not a protected instruction. To calculate latency, add the following terms:

- Time for the current instruction to finish execution (4 state times).
 - If this is a protected instruction, the instruction that follows it must also execute before the interrupt can be acknowledged. Add the execution time of the instruction that follows a protected instruction.
- Time for the next instruction to execute. (The longest instruction, **NORML**, takes 39 state times. See Appendix A for instruction execution times.)
- For standard interrupts only, the response time to get the vector and force the call
 - 11 state times for an internal stack or 13 for an external stack

Please note that, depending on the number of words being transferred, the **BMOV** instruction could take longer than 39 state times to execute. In this case, the **BMOVI** instruction may be a better choice.

1.2.2 STANDARD INTERRUPT LATENCY

Figure 1-2 illustrates interrupt latency. The maximum latency for a standard interrupt is 56 state times (4 + 39 + 13). This delay time does not include the time needed to execute the first instruction in the interrupt service routine or the time to execute the instruction following a protected instruction.

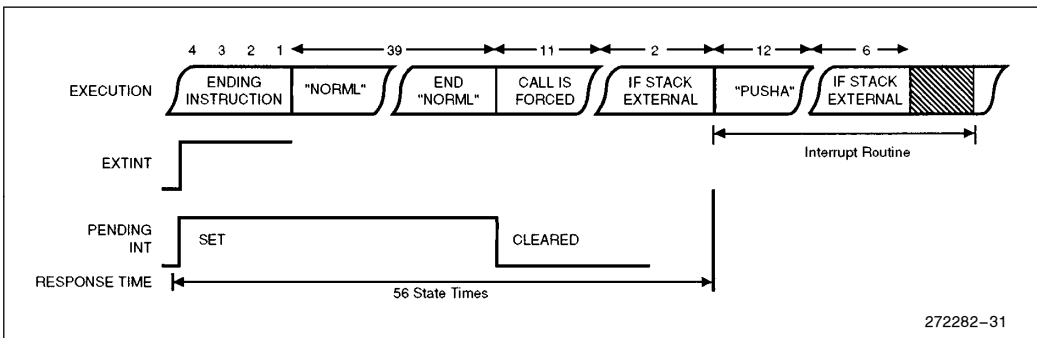


Figure 1-2. Standard Interrupt Response Time

1.2.3 PTS INTERRUPT LATENCY

The interrupt latency for a PTS interrupt is 43 state times (4 + 39). This delay time does not include the added delay if the PTS is disabled (**PSW.2** clear), if a protected instruction is being executed, or if a PTS request is already in progress.

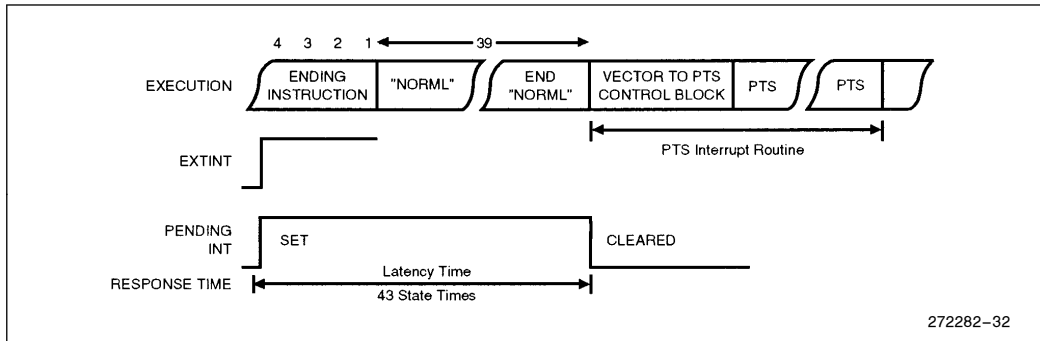


Figure 1-3. PTS Interrupt Response Time

1.3 Interrupt Priorities

Table 1-1 shows the default interrupt priorities (30 is highest and 0 is lowest). The Unimplemented Opcode and Software Trap interrupts are not prioritized; they go directly to the Interrupt Controller for servicing and thus vector immediately upon execution.

The Priority Encoder determines the priority of all other pending interrupt requests. NMI has the highest priority of all prioritized interrupts, PTS interrupts have the next highest priority, and standard interrupts have the lowest. The Priority Encoder selects the highest priority pending request for service.

Table 1-1. Interrupt Sources

Interrupt Service	Symbol	Interrupt Source			PTS Service		
		Name	Vector	Priority	Name	Vector	Priority
NMI	NMI	INT15	0203EH	30	—	—	—
EXTINT Pin	EXTINT	INT14	0203CH	14	PTS14	0205CH	29
Peripheral Interrupt	PI	INT13	0203AH	13	PTS13*	0205AH	28
Capture/Compare5**	CAPCOMP5	INT12	02038H	12	PTS12	02058H	27
Compare4**	COMP4	INT11	02036H	11	PTS11	02056H	26
Capture/Compare4**	CAPCOMP4	INT10	02034H	10	PTS10	02054H	25
Compare3	COMP3	INT09	02032H	09	PTS09	02052H	24
Capture/Compare3	CAPCOMP3	INT08	02030H	08	PTS08	02050H	23
Unimplemented Opcode	—	—	02012H	—	—	—	—
Software TRAP Instruction	—	—	02010H	—	—	—	—
Compare2	COMP2	INT07	0200EH	07	PTS07	0204EH	22
Capture/Compare2	CAPCOMP2	INT06	0200CH	06	PTS06	0204CH	21
Compare1	COMP1	INT05	0200AH	05	PTS05	0204AH	20
Capture/Compare1	CAPCOMP1	INT04	02008H	04	PTS04	02048H	19
Compare0	COMP0	INT03	02006H	03	PTS93	02046H	18
Capture/Compare0	CAPCOMP0	INT02	02004H	02	PTS02	02044H	17
A/D Conversion Complete	AD_DONE	INT01	02002H	01	PTS01	02042H	16
Timer Overflow	TOVF	INT00	02000H	00	PTS00*	02040H	15

*This is a shared interrupt from the PI_PEND register. Be aware that the PTS cannot determine the source of the PI interrupt.

**These interrupts only exist on the 8XC196MD. They are reserved on the 8XC196MC.

1.3.1 MODIFYING INTERRUPT PRIORITIES

The software can modify the default priorities of maskable interrupts by controlling the interrupt mask registers (INT__MASK, INT__MASK1, and PI__MASK). For example, you can specify which interrupts, if any, can interrupt an interrupt service routine. The following code shows one way to prevent all interrupts, except EXTINT (priority 14), from interrupting an EPA COMP3 interrupt service routine (priority 9).

```
COMP3_ISR:
    PUSHA                                ; Save PSW, INT_MASK, INT_MASK1, & WSR
                                        ;(this disables all interrupts)
    LDB INT_MASK1, #01000000B           ;Enable EXTINT only
    EI                                    ;Enable interrupt servicing
                                        ;
                                        ; Service the COMP3 interrupt
                                        ;
    POPA                                  ;Restore PSW, INT_MASK, INT_MASK1, &
                                        ;WSR registers

    RET
CSEG AT 02032H                          ;fill in interrupt table
    DCW COMP3_ISR                        ;
    END
```

Note that location 02032H in the interrupt vector table must be loaded with the value of the label COMP3_ISR before the interrupt request occurs and that the COMP3 interrupt must be enabled for this routine to execute.

This routine, like all interrupt service routines, is handled in the following manner:

1. After the hardware detects and prioritizes an interrupt request, it generates and executes an interrupt call. This pushes the program counter onto the stack and then loads it with the contents of the vector corresponding to the highest priority, pending, unmasked interrupt. The hardware will not allow another interrupt call until after the first instruction of the interrupt service routine is executed.
2. The PUSHA instruction, which is now guaranteed to execute, saves the contents of the PSW, INT__MASK, INT__MASK1, and Window Select Register (WSR) onto the stack and then clears the PSW, INT__MASK, and INT__MASK1. In addition to the arithmetic flags, the PSW contains the global interrupt enable bit (I) and the PTS enable bit (PSE). By clearing the PSW and the interrupt mask registers, PUSHA effectively masks all maskable interrupts, disables standard interrupt servicing, and disables the PTS. Because PUSHA is a protected instruction, it also inhibits interrupt calls until after the next instruction executes.
3. The LDB INT__MASK1 instruction enables those interrupts that you choose to allow to interrupt the service routine. In this example, only EXTINT can interrupt the COMP3 interrupt service routine. By enabling or disabling interrupts, the software establishes its own interrupt servicing priorities.
4. The EI instruction re-enables interrupt processing and inhibits interrupt calls until after the next instruction executes.
5. The actual interrupt service routine executes within the priority structure established by the software.
6. At the end of the service routine, the POPA instruction restores the original contents of the PSW, INT__MASK, INT__MASK1, and WSR registers; any changes made to these registers during the interrupt service routine are overwritten. Because interrupt calls cannot occur immediately following a POPA instruction, the last instruction (RET) will execute before another interrupt call can occur.

Notice that the “preamble” and exit code for this routine does not save or restore register RAM. The interrupt service routine is assumed to allocate its own private set of registers from the lower Register File. The general-purpose Register RAM in the lower Register File makes this quite practical. In addition, the RAM in the upper Register File is available via *windowing*.

1.4 End-of-PTS Interrupts

An end-of-PTS interrupt is a “standard” interrupt, but it is generated by a bit being set in the PTSSRV register. The Interrupt Controller processes it with an interrupt service routine that is stored in the memory location pointed to by the standard interrupt vector. When the end-of-PTS interrupt vectors to the interrupt service routine, hardware clears the PTSSRV bit.

It is important to note that the end-of-PTS interrupt does not affect the conventional interrupt pending bit. The end-of-PTS interrupt is forced by the PTS microcode setting the corresponding PTSSRV bit. Thus, a conventional interrupt and an end-of-PTS interrupt can occur simultaneously, and both will be serviced.

The end-of-PTS interrupt routine typically services the PTSCB and may also process data that was collected from multiple PTS cycles. Before exiting, the interrupt service routine must set the PTSSEL bit to re-enable PTS interrupt servicing for that interrupt. Figure 1-1 shows the program flow for processing of all interrupts.

1.5 Special Interrupts

The device supports three special interrupts: Unimplemented Opcode, Software Trap, and NMI. These interrupts are not affected by the interrupt enable bit (I) in the PSW (PSW.1), and they cannot be masked. All of these interrupts are serviced by the Interrupt Controller; they cannot be assigned to the PTS. Of these three, only NMI goes through the Transition Detector and Priority Encoder. The other two special interrupts go directly to the Interrupt Controller for servicing. Be aware that these interrupts are often assigned to special functions in Intel development tools.

1.5.1 UNIMPLEMENTED OPCODE

If the CPU attempts to execute an unimplemented opcode, an indirect vector through location 02012H occurs. This prevents random software execution during hardware and software failures. The interrupt vector should contain the starting address of an error routine that will not further corrupt an already erroneous situation. The Unimplemented Opcode interrupt prevents other interrupts from being acknowledged until after the next instruction is executed.

1.5.2 SOFTWARE TRAP

The TRAP instruction (opcode 0F7H) causes an interrupt call that is vectored through location 02010H. The TRAP instruction provides a single-instruction interrupt that is useful when debugging software or generating software interrupts. The TRAP instruction prevents other interrupts from being acknowledged until after the next instruction is executed.

1.5.3 NMI

The external NMI pin generates a Nonmaskable Interrupt for implementation of critical interrupt routines. NMI has the highest priority of all the prioritized interrupts. It is passed directly from the Transition Detector to the Priority Encoder, and it vectors indirectly through location 0203EH.

A Transition Detector samples NMI during Phase 1 (CLKOUT low) and latches the interrupt when a low-to-high transition occurs. The interrupt input must be held high for at least one state time to guarantee recognition. Because the interrupts are edge-triggered, only one interrupt is generated if an input is held high.

NOTE:

The interrupt detection logic can generate an interrupt if a momentary negative glitch occurs while the NMI pin is held high. For this reason, NMI should normally be held low when not active.

For design symmetry with the INT_PEND1 register, an NMI mask bit exists in the INT_MASK1 register. However, the mask bit has no function; NMI is enabled for both NMI_MASK set and NMI_MASK cleared. To ensure compatibility with future products, always write zero to the NMI mask bit.

1.6 EPA Interrupts

Interrupts from the EPA to the core are handled in two ways. Event interrupts from all EPA modules except COMP5 are mapped directly to the core interrupt unit as separate sources. These interrupts can be serviced either by a software interrupt routine or by the PTS. The PI interrupt is a shared interrupt which includes the Compare5 (COMP5) and the Waveform Generator (WG) interrupts. Since the PTS cannot tell which interrupt source caused the PI interrupt, it should not be used if more than one of the shared interrupts is unmasked.

1.6.1 USING THE PTS TO SERVICE THE EPA

The PTS can be configured to service the individual EPA interrupts. For example, you could create four PTS channels, one for each Capture module, 0–3. Set up each channel with a fixed source address (the CAPCOMx__TIME register), an auto-incrementing destination address (somewhere in RAM), and a transfer count (*n* transfers). Select PTS service for each interrupt, and configure an EPA module for capture. Each time an event occurs, the time is stored in CAPCOMx__TIME and a PTS interrupt is generated. The PTS transfers the time value from CAPCOMx__TIME to a RAM location and decrements the transfer count. After *n* interrupts (PTS transfers) occur on the channel, the transfer count equals zero, and a conventional software interrupt routine is initiated. This software interrupt service routine could reinitialize the PTS channels and perhaps process the captured data arrays.

You can use a similar method to support multiple sequential output events programmed for a single pin. After one output event executes, the resulting interrupt request causes the PTS to transfer the next event time from RAM into CAPCOMx__TIME. Since the PTS can take only a single action, it cannot change the control register. Therefore, this method can be used only when the same action is to be reinitiated at different times or when the action is a pin toggle. (See “Configuring the PTS” Section 1.8 for a complete discussion of the PTS modes.)

1.7 Programming the Interrupts

Table 1-2 lists the programmable registers that affect the performance and function of the Interrupt Controller and PTS.

Table 1-2. Interrupt and PTS Control and Status Registers

Register Mnemonic	Register Name	Description
PI_MASK	Peripheral Interrupt Mask Register	This register enables/disables the 4 shared interrupts COMP5, WG, TF2, TF1
PI_PEND	Peripheral Interrupt Pending Register	The bits in this register are set by hardware to indicate that an interrupt is pending.
INT_MASK INT_MASK1	Interrupt Mask Registers	These registers enable/disable each maskable interrupt (that is, each interrupt except Unimplemented Opcode, Software Trap, and NMI.)
INT_PEND INT_PEND1	Interrupt Pending Registers	The bits in this register are set by hardware to indicate that an interrupt is pending.
PSW	Program Status Word	This register contains one bit that globally enables or disables servicing of all maskable interrupts and another that enables or disables the PTS.
PTSSEL	PTS Select Register	This register selects either a PTS cycle or a standard interrupt service routine for each of the fifteen maskable interrupt requests.
PTSSRV	PTS Service Register	The bits in this register are set by microcode to request an end-of-PTS interrupt.

1.7.1 SELECTING EITHER PTS OR STANDARD INTERRUPT SERVICE

The PTS Select register (PTSSSEL) selects either a PTS cycle or a standard software interrupt service routine for each of the maskable interrupt requests (see Figure 1-4). Setting a bit selects a PTS cycle; clearing a bit selects a standard interrupt service routine. Note that both the PTSSSEL and PTSSRV registers share the same bit names and locations within their registers.

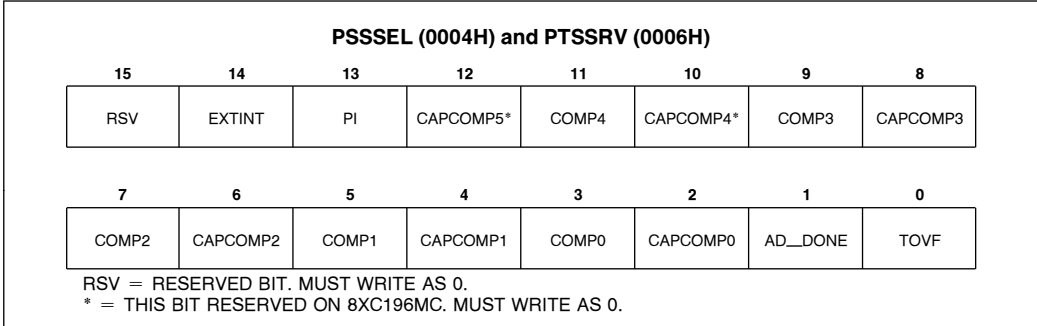


Figure 1-4. PTSSSEL Register

1.7.2 ENABLING PTS INTERRUPTS

When you assign an interrupt to the PTS, you must set up a PTS control block (PTSCB) for each interrupt source. Once the control block is set up, you must enable both the PTS and the individual interrupt. The PTS enable (PSE) bit in the Program Status Word (PSW.2) globally enables or disables the PTS. The EPTS instruction sets the bit, which enables the PTS, and the DPPTS instruction clears the bit, which disables the PTS. The bits in INT__MASK and INT__MASK1 individually enable or disable the interrupts (see Figure 1-5). These bits must be set to service the interrupt with the PTS.

The PTS does not require that global interrupt servicing be enabled. However, you must enable global interrupt servicing to handle the end-of-PTS interrupt. The global interrupt enable (I) bit in the Program Status Word (PSW.1) globally enables or disables the servicing of all maskable interrupts. The EI instruction sets the bit, which enables standard interrupt servicing, and the DI instruction clears the bit, which disables standard interrupt servicing.

NOTE:

PTS cycles will occur after a DI instruction, if the appropriate INT__MASK and PTSSSEL bits are set. However, the end-of-PTS interrupt will not occur.

1.7.3 ENABLING STANDARD INTERRUPTS

When you assign an interrupt to a standard software service routine, you must enable both the servicing of the interrupt and the individual interrupt. The global interrupt enable (I) bit in the Program Status Word (PSW.1) globally enables or disables the servicing of all maskable interrupts. The EI instruction sets the bit, which enables interrupt servicing. The DI instruction clears the bit, which disables interrupt servicing. The bits in INT__MASK and INT__MASK1 individually enable or disable the interrupts (see Figure 1-5). Interrupts that occur while interrupt servicing is globally disabled (PSW.1 cleared) are held in the interrupt pending registers.

1.7.4 INTERRUPT MASK REGISTERS

The interrupt mask registers, INT_MASK and INT_MASK1 (Figure 1-5), enable or disable (mask) individual interrupts. With the exception of the Nonmaskable Interrupt (NMI) bit (INT_MASK1.7), setting a bit enables the corresponding interrupt source; clearing a bit disables the source. A device reset or the PUSHA instruction clears the interrupt mask registers (disabling interrupts). The PUSHF instruction clears INT_MASK but does not clear INT_MASK1.

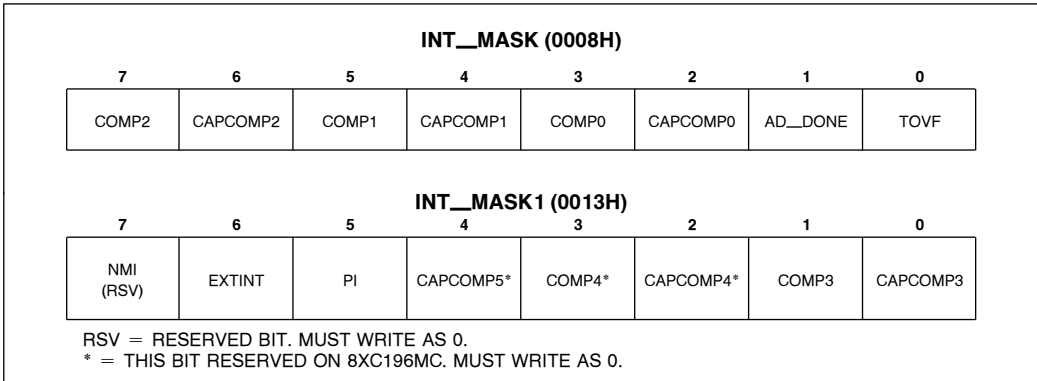


Figure 1-5. INT_MASK and INT_MASK1 Registers

1.7.5 INTERRUPT PENDING REGISTERS

When the Transition Detector detects an interrupt, it sets the corresponding bit in the INT_PEND or INT_PEND1 register. This bit is set even if the individual interrupt is disabled (masked). The pending bit is cleared when the program vectors to the interrupt service routine. INT_PEND and INT_PEND1 can be read, to determine which interrupts are pending. They can also be modified (written), either to clear pending interrupts or to generate interrupts under software control.

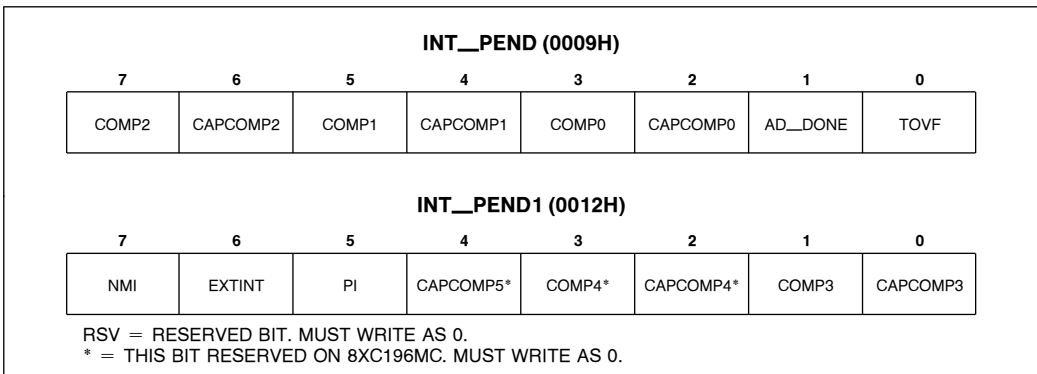


Figure 1-6. INT_PEND and INT_PEND1 Registers

Care should be taken in writing code that modifies these registers. For example, an instruction sequence that clears a pending bit could result in an interrupt being acknowledged after the sequence begins but before the bit is actually cleared. In this case a five-state-time *partial* interrupt cycle occurs. That is, the interrupt process begins, but never jumps to the interrupt service routine. This time delay can be avoided by making the code inseparable, in the sense that an interrupt will not be acknowledged while the code is executing. The easiest way to do this is to use the logical instructions in the two- or three-operand format, as in the example:

```

ANDB INT_PEND,#01111111B      ; Clears the COMP2 interrupt
ORB  INT_PEND,#10000000B      ; Sets the COMP2 interrupt
    
```

The device does not acknowledge interrupts during execution of these “read-modify-write” instructions.

1.7.6 THE PERIPHERAL INTERRUPT (PI)

The Peripheral Interrupt (PI) is a “shared” interrupt, and signals that an interrupt has been generated by either the Compare5 (COMP5) module or the Waveform Generator (WG). When any of these sources generate an interrupt, the corresponding bit in the PI_PEND register is set. If the bit is unmasked in the PI_MASK register, the PI interrupt pending bit in INT_PEND1 is set. Figure 1-7 illustrates the relationship between these registers.

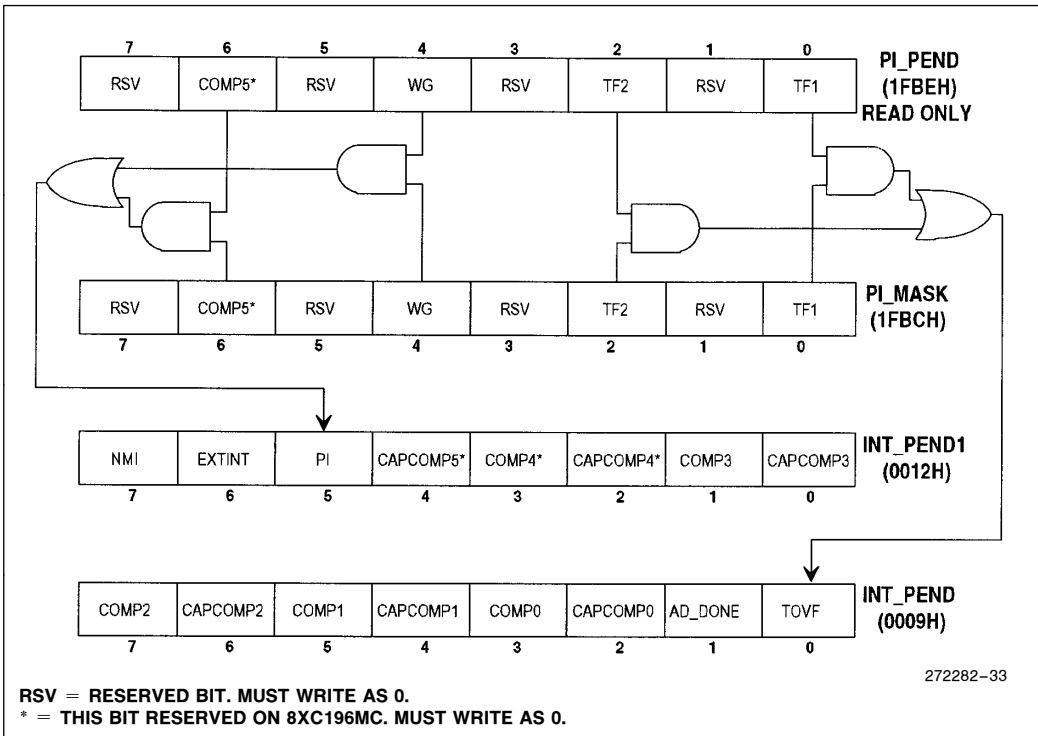


Figure 1-7. PI Interrupt Sharing

The user interrupt routine can read the PI_PEND register to determine what the source of the interrupt was. Note that reading PI_PEND clears all bits. Therefore, the value of the register must be stored in a shadow register if more than one bit needs to be checked. Also note that the PI_PEND bits cannot be set by writing to the PI_PEND register. This register is read only, writes will have no effect.

It is not necessary to read (clear) PI_PEND to re-enable the PI interrupt function. The signals that set the PI_PEND bits will (if unmasked) always cause the PI bit to be set in INT_PEND1.

1.7.6.1 WG Interrupt

The WG interrupt is generated by the Waveform Generator reload compare function, discussed in Section 6. This interrupt is typically used to calculate and/or reload the output and compare registers in the WFG. If the WFG registers are all that need updating, the PTS can service this interrupt with a block move of data to these registers.

1.7.6.2 COMP5 Interrupt

The Compare5 interrupt is also shared through the PI. This interrupt is generated by the EPA in response to an EPA compare event. See the user’s manual for information on using this function.

1.7.7 THE EXTINT INTERRUPT

The EXTINT interrupt is generated by the WFG protection circuitry, described in Section 6.10. Two bits in the WG_PROTECT register (Figure 6-14), Interrupt Type (IT) and Enable Sampling (ES) control the type of external event which will cause EXTINT. EXTINT can be caused by a transition (rising or falling edge), or by a constant level (high or low). Table 1-3 shows the combinations for these bits. The EXTINT interrupt sets the EXTINT bit in the INT_PEND1 register, and if unmasked in the INT_MASK1 register, takes the vector located at 203CH.

Table 1-3. EXTINT Mode Selection

ES	IT	EXTINT Input Characteristics
0	0	Falling Edge Triggered
0	1	Rising Edge Triggered
1	0	Low Level Triggered
1	1	High Level Triggered

The transition modes are selected by clearing the ES bit. To be a valid transition, the signal must remain asserted for a minimum period of $2 T_{osc}$ ($T_{osc} = 2/F_{XTAL}$). The IT bit controls whether a rising edge (IT = 1) or falling edge (IT = 0) causes the interrupt.

The level modes are selected by setting the ES bit. To be a valid level, the signal must remain asserted for a minimum period of $24 T_{osc}$. When the signal is asserted, sample circuitry monitors the input level 3 times during a $24 T_{osc}$ period. The signal must be asserted for each of the samples before it is recognized as valid. If the signal is valid, the EXTINT interrupt is generated. The IT bit controls whether a high level (IT = 1) or low level (IT = 0) input signal causes the interrupt.

The level mode is useful in noisy environments, where a noise spike might cause an unintended interrupt. Note that the same signal which generates the EXTINT also generates the output disable signal, discussed in Section 6.10.

1.7.8 THE TOVF INTERRUPT

The Timer Overflow Interrupt (TOVF) is a “shared” interrupt, and signals that an interrupt has been generated by either Timer 1 (TF1) or Timer 2 overflow/underflow (TF2). When these sources generate an interrupt, the corresponding bit in the PI_PEND register is set. If the bit is unmasked in the PI_MASK register, the TOVF interrupt pending bit in INT_PEND is set. Figure 1-7 illustrates the relationship between these registers.

The user interrupt routine can read the PI_PEND register to determine what the source of the interrupt was. Note that reading PI_PEND clears all bits. Therefore, the value of the register must be stored in a shadow register if more than one bit needs to be checked. Also note that the PI_PEND bits cannot be set by writing to the PI_PEND register. This register is read only, writes will have no effect.

It is not necessary to read (clear) PI_PEND to re-enable the TOVF interrupt function. The signals that set the PI_PEND bits will (if unmasked) always cause the PI bit to be set in INT_PEND1.

1.8 Configuring the PTS

Each PTS interrupt requires a block of data called the PTS Control Block (PTSCB). The PTSCB determines the PTS mode, the number of PTS cycles, and the address of the source and destination of data transfers. You must set up the PTSCB for each interrupt source **before** enabling the corresponding PTS interrupts.

1.8.1 SETTING UP PTS CONTROL BLOCKS

Each PTS control block (PTSCB) requires eight data bytes in register RAM. The address of the first (lowest) byte is stored in the PTS vector table in special-purpose memory (see Chapter 4). Figure 1-8 shows the PTSCB for each PTS mode. Unused PTSCB bytes can be used as RAM. The SIO mode is unique in that it links two PTSCB's together (SIO #1 and SIO #2) in order to perform its function. The PTSVEC in SIO #1 points to the base address of SIO #2.

NOTE:

The PTSCB must be located in register RAM, in page 00H. The location of the first byte of the PTSCB must be aligned on a quad-word boundary (an address evenly divisible by 8).

Single Transfer	Block Transfer	A/D Scan Mode	SIO #1	SIO #2
Unused	Unused	Unused	PTSVEC1 (HI)	Unused
Unused	PTSBLOCK	Unused	PTSVEC1 (LO)	SAMPTIME
PTSDST (HI)	PTSDST (HI)	PTSPTR2 (HI)	BAUD (HI)	DATA (HI)
PTSDST (LO)	PTSDST (LO)	PTSPTR2 (LO)	BAUD (LO)	DATA (LO)
PTSSRC (HI)	PTSSRC (HI)	PTSPTR1 (HI)	EPAREG (HI)	PTSCON1
PTSSRC (LO)	PTSSRC (LO)	PTSPTR1 (LO)	EPAREG (LO)	PORTMASK
PTSCON	PTSCON	PTSCON	PTSCON	PORTREG (HI)
PTSCOUNT	PTSCOUNT	PTSCOUNT	PTSCOUNT	PORTREG (LO)

Figure 1-8. PTS Control Blocks

1.8.1.1 PTSCOUNT Register

In Single Transfer, Block Transfer, and A/D Scan modes, the first location of each PTSCB is the PTSCOUNT register. PTSCOUNT defines the number of PTS cycles to be executed consecutively without software intervention. Since PTSCOUNT is an 8-bit value, the maximum number of cycles is 255. PTSCOUNT is decremented at the end of each PTS cycle. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.

1.8.1.2 PTSCON Register

The second location of each PTSCB (except SIO #2) is the PTSCON register. The upper three bits of the PTSCON register determine the PTS mode (Table 1-4).

Table 1-4. PTS Mode Select (PTSCON Bits 5–7)

Desired PTS Mode	Bit 7 (M2)	Bit 6 (M1)	Bit 5 (M0)
Single Transfer	1	0	0
Block Transfer	0	0	0
A/D Scan	1	1	0
Serial Transmit Mode	0	1	1
Serial Receive Mode	0	0	1

The PTS Mode determines the configuration of the remaining bits. PTSCON has one configuration for Single and Block Transfer modes and A/D Scan mode (Figure 1-9), and one for the serial I/O modes (Figure 1-11).

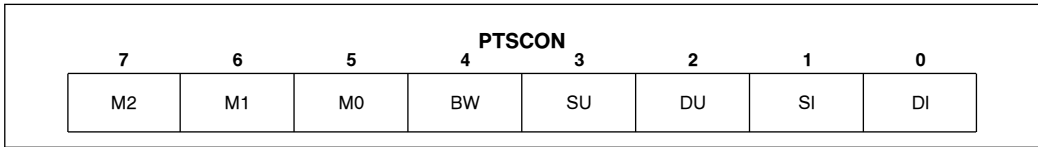


Figure 1-9. PTSCON Register

Table 1-5. PTSCON Bits 0–4 for Single and Block Transfer Modes

Bit Number	Bit Mnemonic	Bit Name	Description
0	DI	PTSDST Auto-Increment	Setting this bit causes the PTS destination register to increment at the end of each PTS cycle. (In Single Transfer mode, the DI and DU bits must be equal.)
1	SI	PTSSRC Auto-Increment	Setting this bit causes the PTS source register to increment at the end of each PTS cycle. (In Single Transfer mode, the SI and SU bits must be equal.)
2	DU	Update PTSDST	Setting this bit causes the PTSDST register to retain its final value at the end of a PTS cycle. Clearing it causes the register to revert to the value that existed at the beginning of the PTS cycle. (In Single Transfer mode, the DI and DU bits must be equal.)
3	SU	Update PTSSRC	Setting this bit causes the PTSSRC register to retain its final value at the end of a PTS cycle. Clearing it causes the register to revert to the value that existed at the beginning of the PTS cycle. (In Single Transfer mode, the SI and SU bits must be equal.)
4	BW	Byte/Word Transfer	Setting this bit specifies a byte transfer. Clearing it specifies a word transfer.

Table 1-6. PTSCON Bits 0–4 for A/D Scan Mode

Bit Number	Bit Mnemonic	Bit Name	Description
0	—	—	Always one.
1	—	—	Always one.
2	—	—	Always zero.
3	SU	Update PTSPTR1	Setting this bit causes the PTSPTR1 register to retain its final value at the end of a PTS cycle. Clearing it causes the register to revert to the value that existed at the beginning of the PTS cycle.
4	—	—	Always zero.

1.8.2 SINGLE TRANSFER MODE

Single Transfer mode is typically used with the EPA, in capture mode, to move the captured time value from the SFR location to internal RAM for further processing.

In the Single Transfer mode, each PTS cycle transfers a single byte or word (selected by the BW bit in PTSCON) from one memory location to another. The PTSCOUNT register specifies the number of transfers (each transfer is one PTS cycle). The PTS moves the byte or word from the location pointed to by the source register (PTSSRC) to the location pointed to by the destination register (PTSDST).

PTSSRC and PTSDST may point to any memory location; however, they must point to an even address if word transfers are selected. Setting the auto-increment and update bits causes the PTS to increment the source (if SI and SU are set) and/or destination (if DI and DU are set) address at the end of each PTS cycle. The address increments by one if byte transfers are selected or by two if word transfers are selected. In Single Transfer mode, each pair of auto-increment and update bits (SI/SU, DI/DU) must both be either set or cleared. Programming either pair to unequal values selects an invalid mode. The source and destination can be incremented and updated independently of one another. (The SI and SU pair must be equal, and the DI and DU pair must be equal. However, the two pairs, SI/SU and DI/DU, need not be equal.)

1.8.2.1 Single Transfer Mode Example

The PTSCB in Table 1.7 defines nine PTS cycles. Each cycle moves a single word from location 20H to an external memory location. The PTS transfers the first word to location 6000H. Then it increments and updates the destination address and decrements the PTSCOUNT register; it does not increment the source address. When the second cycle begins, the PTS moves a second word from location 20H to location 6002H. When PTSCOUNT equals zero, the PTS will have filled locations 6000H-600FH, and an end-of-PTS interrupt is generated.

Table 1-7. Single Transfer Mode PTSCB

Unused
Unused
PTSDST (HI) = 60H
PTSDST (LO) = 00H
PTSSRC (HI) = 00H
PTSSRC (LO) = 20H
PTSCON = 85H (Mode = 100, DI & DU = 1 BW = 0)
PTSCOUNT = 09H

1.8.3 BLOCK TRANSFER MODE

In Block Transfer mode, the PTS moves a block of bytes or words (selected by the BW bit in PTSCON) from one memory location to another. The PTSBLOCK register specifies the number (from 1–32) of bytes or words in each block. The PTS moves the block of bytes or words from the location pointed to by the source register (PTSSRC) to the location pointed to by the destination register (PTSDST).

PTSSRC and PTSDST may point to any memory location; however, they must point to an even address if word transfers are selected. Setting the auto-increment bits in the PTSCON register causes the PTS to increment the source (SI set) and/or destination (DI set) address at the end of each PTS transfer. If the update bit is also set, the incremented address is saved in the PTSSRC (SU set) or PTSDST (DU set) register after each PTS cycle. Setting both the increment and update bits causes the source and/or destination address to be incremented after each cycle. The registers increment by one if byte transfers are selected or by two if word transfers are selected. The increment and update features may be selected independently (unlike in Single Transfer Mode).

In this mode, it is important to differentiate between a *PTS transfer* and a *PTS cycle*. A *PTS transfer* is the movement of a single byte or word from the source to the destination. A *PTS cycle* consists of the transfer of an entire block of bytes or words. Because a *PTS cycle* is uninterruptable, the Block Transfer mode can create long interrupt latency. The worst-case latency could be as high as 500 states. This worst-case latency assumes a block transfer of 32 words from one external memory location to another using an 8-bit bus with no wait states.

1.8.3.1 Block Transfer Mode Example

The PTSCB in Table 1.8 defines three PTS cycles that will each transfer the bytes in memory locations 20H–24H to one of the following blocks: 6000H–6004H, 6005H–6009H, or 600AH–600EH. Each PTS cycle transfers a block of five bytes. The source and destination are incremented after each transfer, but only the destination is updated after each cycle. The first byte of each cycle is always read from location 20H.

Table 1-8. Block Transfer Mode PTSCB

Unused
PTSBLOCK = 05H
PTSDST (HI) = 60H
PTSDST (LO) = 00H
PTSSRC (HI) = 00H
PTSSRC (LO) = 20H
PTSCON = 17H (Mode = 000; DI, SI DU, BW = 1; SU = 0)
PTSCOUNT = 03H

1.8.4 A/D SCAN MODE

In the A/D Scan mode, the PTS causes the A/D converter to perform multiple conversions on one or more channels and then stores the results. To use the A/D Scan mode, you must first set up a command/data table in memory (Table 1-9). The command/data table contains A/D commands that are interleaved with blank memory locations. The PTS stores the conversion results in these blank locations.

To initiate A/D Scan mode, enable the A/D Conversion Complete interrupt and assign it to the PTS, then have software start the first conversion. When the A/D finishes the first conversion and generates an A/D Conversion Complete interrupt, the PTS cycle is initiated.

During each PTS cycle, the PTS stores the results from the previous conversion and then executes the next conversion command. Since the conversion results are not stored until the next PTS cycle, the last command location should contain all zeros to prevent a final conversion from starting. Typically, the A/D commands are loaded into the table from an external ROM. Only the amount of available memory limits the table size; it can reside in internal or external RAM.

Table 1-9. A/D Scan Mode Command/Data Table

XXX + 0AH	A/D Result 2	
XXX + 8H	Unused	A/D Command 3
XXX + 6H	A/D Result 1	
XXX + 4H	Unused	A/D command 2
XXX + 2H	A/D Result 0*	
XXX	Unused	A/D command 1

*Result of the A/D conversion that initiates the PTS cycle.

In A/D Scan mode, the PTSCOUNT specifies the total number of A/D conversion cycles. The PTSPTR1 register points to the table of conversion commands and results. Setting the UPDT bit in the PTSCON register (PTSCON.3) causes the PTSPTR1 register to retain its final value at the end of the PTS cycle. Clearing it causes the register to revert to the value that existed at the beginning of the PTS cycle. PTSPTR2 points to the AD_RESULT register.

1.8.4.1 PTS Cycles in A/D Scan Mode

Software must start the first A/D conversion. The A/D Conversion Complete interrupt initiates the PTS cycle. The following actions occur after the PTS cycle begins:

1. The PTS reads the first command, stores it in a temporary location, and increments the PTSPTR1 register twice. PTSPTR1 now points to the first blank location in the command/data table (address XXX + 2).
2. The PTS reads the AD_RESULT register, stores the results of the first conversion into location XXX + 2 in the command/data table, and increments the PTSPTR1 register twice. PTSPTR1 now points to XXX + 4.
3. The PTS loads the command from the temporary location into the AD_COMMAND register. This starts the next A/D conversion cycle.
4. If UPDT (PTSCON.3) is clear, the PTSPTR1 register is reinitialized to its original value. The next cycle will use the same command and overwrite previous data. If UPDT is set, the PTS saves the new contents of PTSPTR1 and it points to the next command.
5. PTSCOUNT is decremented and the CPU returns to regular program execution. When PTSCOUNT reaches zero, hardware clears the corresponding PTSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.

When the conversion started by the PTS cycle completes and the A/D generates the A/D Conversion Complete interrupt, a new PTS cycle begins. Steps 1–5 repeat.

Because the lower six bits of the AD_RESULT register contain status information, the end-of-PTS interrupt service routine could shift the results data to the right six times to leave only the conversion results in the memory locations.

1.8.4.2 A/D Scan Mode Example 1

The command/data table shown in Table 1.10 sets up a series of A/D conversions, beginning with channel 7 and ending with channel 4. Each table entry is a word (two bytes). Table 1-11 shows the corresponding PTSCB.

Software starts a conversion on Channel 7. Upon completion of the conversion, the A/D Conversion Complete interrupt initiates the first PTS cycle. Step 1 stores the Channel 6 command in a temporary location and increments PTSPTR1 to 3002H. Step 2 stores the result of the Channel 7 conversion in location 102H and increments PTSPTR1 to 3004H. Step 3 loads the Channel 6 command from the temporary location into the AD_COMMAND register to start the next conversion. Step 4 updates PTSPTR1 (PTSPTR1 points to 3004H) and step 5 decrements PTSCOUNT to 7. The next cycle begins by storing the Channel 5 command in the temporary location. During the last cycle (PTSCOUNT = 1), the dummy command is loaded into the AD_COMMAND register and no conversion is performed. PTSCOUNT is decremented to zero and the end-of-PTS interrupt is requested.

Table 1-10. Command/Data Table (Example 1)

Address	Contents
300EH	AD_RESULT for ACH4
300CH	0000H (Dummy Command)
300AH	AD_RESULT for ACH5
3008H	AD_COMMAND for ACH4
3006H	AD_RESULT for ACH6
3004H	AD_COMMAND for ACH5
3002H	AD_RESULT for ACH7
3000H	AD_COMMAND for ACH6

Table 1-11. A/D Scan Mode PTSCB (Example 1)

Unused
Unused
PTSPTR2 (HI) = 1FH
PTSPTR2 (LO) = AAH
PTS_PTR1 (HI) = 30H
PTS_PTR1 (LO) = 00H
PTSCON = CBH (Mode = 110, UPDT = 1)
PTSCOUNT = 04H

1.8.4.3 A/D Scan Mode Example 2

Table 1-13 sets up a series of ten PTS cycles, each of which reads a single A/D channel and stores the result in a single location (3002H). The UPDT bit (PTSCON.3) is cleared so that original contents of PTSPTR1 are restored after the cycle. The command/data table is shown in Table 1-12.

Table 1-12. Command/Data Table (Example 2)

Address	Contents
3002H	AD__RESULT for ACHx
3000H	AD__COMMAND for ACHx

Table 1-13. A/D Scan Mode PTSCB (Example 2)

Unused
Unused
PTSPTR2 (HI) = 1FH
PTSPTR2 (LO) = AAH
PTSPTR1 (HI) = 30H
PTSPTR1 (LO) = 00H
PTSCON = C3H (Mode = 110, UPDT = 0)
PTSCOUNT = 0AH

Software starts a conversion on Channel *x*. When the conversion is finished and the A/D Conversion Complete interrupt is generated, the first PTS cycle begins. The PTS stores the value of the AD__RESULT register in location 3002H and then copies the conversion command from location 3000H to the AD__COMMAND register. The CPU can process or move the conversion results data from the table before the next conversion completes and a new PTS cycle begins. When the next cycle begins, PTSPTR1 again points to 3000H. The value of the AD__RESULT register is written to location 3002H and the command at location 3000H is re-executed.

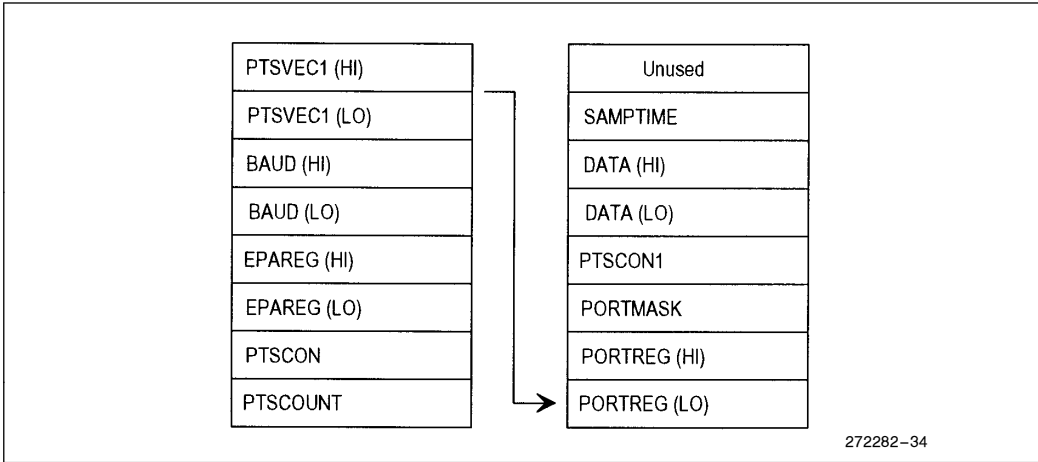
1.8.5 SERIAL I/O MODES

Since the 8XC196MC/MD has no hardware UART, the serial I/O function is implemented using special PTS modes. There are four basic serial modes of operation; several options are available within each of these modes.

The SIO mode require 2 PTSCB's to completely configure all possible options. These blocks do not need to be contiguous, but must be in register RAM and located on quad-word boundaries.

The serial modes are very versatile. The baud rate is established by an EPA channel, and up to 16 bits per character (including parity and stop bits) may be transmitted/received. In the synchronous modes, the shift clock can either be input to or output by the 8XC196MC/MD. The data and clock may be assigned to most I/O pins in port 2 or port 7. Ports 3, 4 and 5 do not support the serial I/O function.

Additional information on the PTS SIO modes can be found in the *8XC196MC User's Manual*. Software examples are given in this Ap-Note, Chapters 2–5.



272282-34

Figure 1-10. PTS SIO Control Blocks

1.8.5.1 PTSCOUNT Register

PTSCOUNT defines the number of PTS cycles to be executed consecutively without software intervention. Since PTSCOUNT is an 8-bit value, the maximum number of cycles is 256 (with setting of 0). In the SIO modes, PTSCOUNT determines how many bits will be transmitted or received. The different SIO modes have different requirements for PTSCOUNT; refer to Sections 1.8.6 and 1.8.7 for detailed information.

Like in all other PTS modes, PTSCOUNT is decremented at the end of each PTS cycle. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.

1.8.5.2 PTSCON Register

The PTSCON register controls the type of serial mode that is implemented. The upper three bits determine if the transmit or receive function is being generated. The other bits are described below in Table 1-14.

	7	6	5	4	3	2	1	0
	M2	M1	M0	SA	RSV	RSV	SA	MAJ

M2, M1, M0 FUNCTION
 0 0 1 RECEIVE MODE
 0 1 1 TRANSMIT MODE

RSV = RESERVED BITS. MUST WRITE AS 0.

Figure 1-11. PTSCON Register in SIO Modes

Table 1-14. PTSCON Bits 0–4 for SIO Modes

Bit Number	Bit Mnemonic	Bit Name	Description
0	MAJ	Majority Sample	Setting this bit enables majority sampling. This is valid in asynchronous receive mode only. Otherwise this bit must be zero.
1	SA*	Sync/Asynch	Setting this bit enables the synchronous modes; clearing it enables asynchronous modes. Both SA bits must have the same value.
2	—	—	Always zero.
3	—	—	Always zero.
4	SA*	Sync/Asynch	Setting this bit enables the synchronous modes; clearing it enables asynchronous modes. Both SA bits must have the same value.

*Note that there are two SA bits. They must always be set to the same value.

1.8.5.3 EPAREG Register

The EPAREG holds the 16-bit address of the CAPCOMP_x_TIME or CAP_x_TIME register that is being used for the SIO operation.

1.8.5.4 BAUD Register

The BAUD register holds a 16-bit value that is used to determine the baud rates for the SIO. The value is calculated as follows:

Asynchronous modes:

$$\text{BAUD_VALUE} = \frac{F_{\text{XTAL}}}{4 \times \text{BAUDRATE} \times \text{EPA_PRESCALE}}$$

where

- BAUD_VALUE = 16-bit integer loaded into BAUD register
- F_{XTAL} = XTAL1 pin input frequency, Hz
- BAUDRATE = baudrate in bits/seconds
- EPA_PRESCALE = amount of EPA timer prescaling, 1–64

Synchronous modes:

$$\text{BAUD_VALUE} = \frac{F_{\text{XTAL}}}{8 \times \text{BAUDRATE} \times \text{EPA_PRESCALE}}$$

where

- BAUD_VALUE = 16-bit integer loaded into BAUD register
- F_{XTAL} = XTAL1 pin input frequency, Hz
- BAUDRATE = baudrate in bits/seconds
- EPA_PRESCALE = amount of EPA timer prescaling, 1–64

1.8.5.5 PTSVEC1 Register

This is a 16-bit pointer to the base address of the SIO #2 PTSCB.

1.8.5.6 PORTREG Register

This is the 16-bit address of port that contains the TXD pin (Px_REG) or the RXD pin (Px_PIN).

1.8.5.7 PORTMASK Register

This register defines which pin of the port will be used for the TXD or RXD pin. The pin that is selected for this function must = 1; all other bits must = 0.

1.8.5.8 PTSCON1 Register

This control register takes two different forms depending on whether asynchronous or synchronous mode is selected.

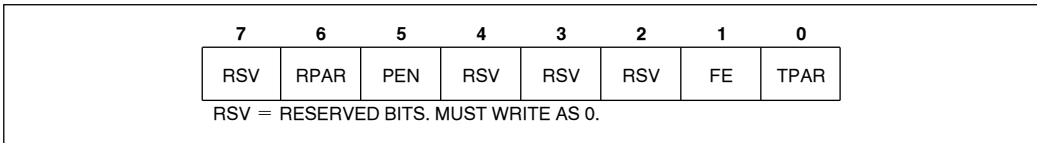
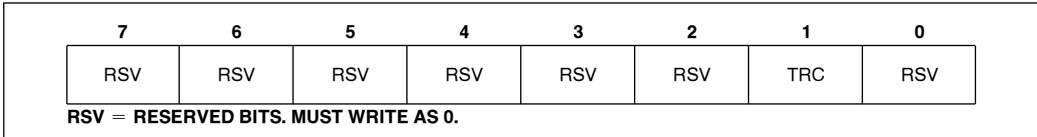


Figure 1-12. PTSCON1 in Asynchronous Mode

Table 1-15. PTSCON Bits 0–7 for Asynchronous SIO Mode

Bit Number	Bit Mnemonic	Bit Name	Description
0	TPAR	Transmit Parity Control	Set this bit for even parity; clear for odd parity.
1	FE	Framing Error Flag	This bit is set if the stop bit received was not a one, otherwise it is cleared. This bit must be reset at the start of every reception.
2	—	—	Always zero.
3	—	—	Always zero.
4	—	—	Always zero.
5	PEN	Parity Enable Bit	Setting this bit enables Parity; clearing it disable parity.
6	RPAR	Receive Parity Control/Status	Before a reception, initialize to 0 for even and 1 for odd parity. If at the end of a reception the bit is 1, a parity error has occurred. This bit must be initialized prior to each reception.
7	—	—	Always zero.


Figure 1-13. PTSCON1 in Synchronous Mode
Table 1-16. PTSCON Bits 0–7 for Synchronous SIO Mode

Bit Number	Bit Mnemonic	Bit Name	Description
0	—	—	Always zero.
1	TRC	Transmit/ Receive Control	Set this bit to receive or transmit data on first and every other (odd numbered) PTS cycles. Clear bit to receive or transmit data on second and every other (even numbered) PTS cycles. This bit must be initialized at the start of every transmission or reception.
2	—	—	Always zero.
3	—	—	Always zero.
4	—	—	Always zero.
5	—	—	Always zero.
6	—	—	Always zero.
7	—	—	Always zero.

1.8.5.9 DATA Register

This 16-bit register holds the data to be transmitted or that has been received. In receive mode, data is shifted into the most significant bit (bit 15), and shifts right with each successive bit received. The received bits will be left justified, with the last bit received in the MSB position.

In transmit mode, the register must be loaded right justified. The bits will shift out to the right, the LSB being sent first.

1.8.5.10 SAMPTIME Register

This 8-bit register controls the time between samples when majority sampling mode is used during asynchronous receive only. If majority sampling is enabled, this register must contain a value between 1 and 31. Use the following formula to calculate time between samples:

$$\text{SAMPTIME_VALUE} = \frac{T_{\text{sam}} \times F_{\text{XTAL}}}{2} - 9$$

where

T_{sam} = time between samples, μs

F_{XTAL} = XTAL1 pin input frequency, MHz

SAMPTIME_VALUE = integer to load into SAMPTIME register. Must be 1–31

1.8.6 PTS ASYNCHRONOUS SIO RECEIVE MODE

To set up the PTS asynchronous SIO receive mode we need a EPA module to set up the timing interval between bits.

During a receive operation, a CAPCOMP module is used, as we need to capture the leading edge of the “start” bit. After the start bit has been captured, the EPA is switched to the compare mode, and time intervals are selected such that the bit input value is sampled in the middle of each bit period. If majority sampling is selected, three consecutive samples are taken, and the majority value is shifted into the DATA register.

After all samples are collected, the end-of-PTS interrupt is generated, and the data read out. The PTSCB's must then be reinitialized.

Figure 1-14 illustrates the sampling process during Asynchronous SIO mode. Chapter 5 contains a software example of this mode.

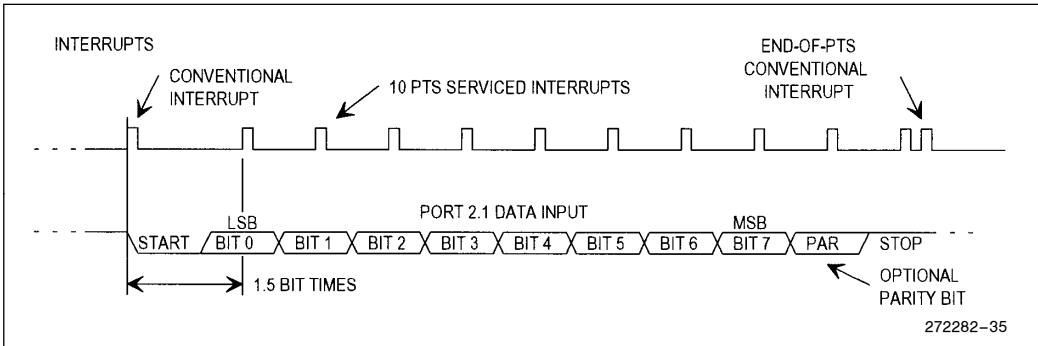


Figure 1-14. Asynchronous SIO Receive Mode Timing

1.8.6.1 PTS Asynchronous SIO Transmit Mode

To set up the PTS asynchronous SIO transmit mode we need an EPA module to set up the timing interval between bits.

A shift clock is generated internally using an EPA compare module. Each time the EPA interrupt occurs, the PTS shifts a bit out of the data register and onto the I/O pin. After the last bit is shifted out, the end-of-PTS interrupt is used to reinitialize the PTSCB's.

Figure 1-15 shows the timing when using the SIO transmit mode. Chapter 4 contains a software example of this mode.

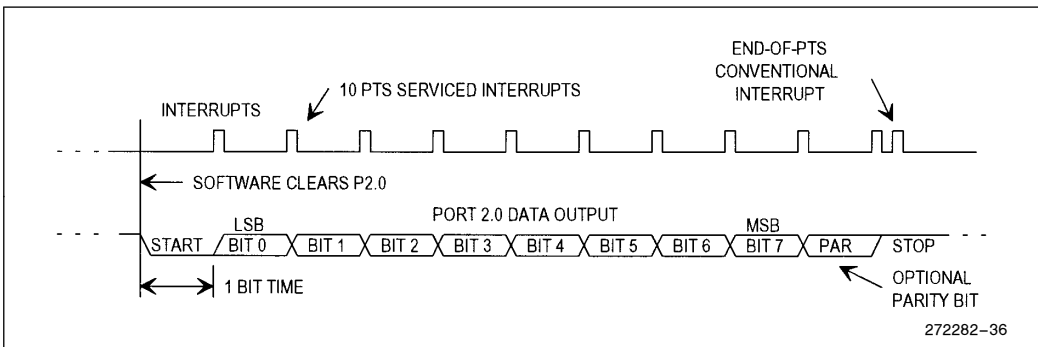


Figure 1-15. Asynchronous SIO Transmit Mode Timing

1.8.6.2 PTS Synchronous SIO Receive Mode

To set up the PTS synchronous SIO receive mode we need an EPA module to set up the timing interval between bits. This same module can either generate a shift clock output, or input an external shift clock.

During receive operation, the EPA channel captures (or generates) the shift clock. Every other edge is used to shift the value of the RXD pin into the DATA register. Either even or odd edges can accomplish this shift, depending on system requirements.

After all samples are collected, the end-of-PTS interrupt is generated, and the data read out. The PTSCB's must then be reinitialized.

Figure 1-16 illustrates the sampling process during Synchronous SIO mode. Chapter 3 contains a software example of this mode.

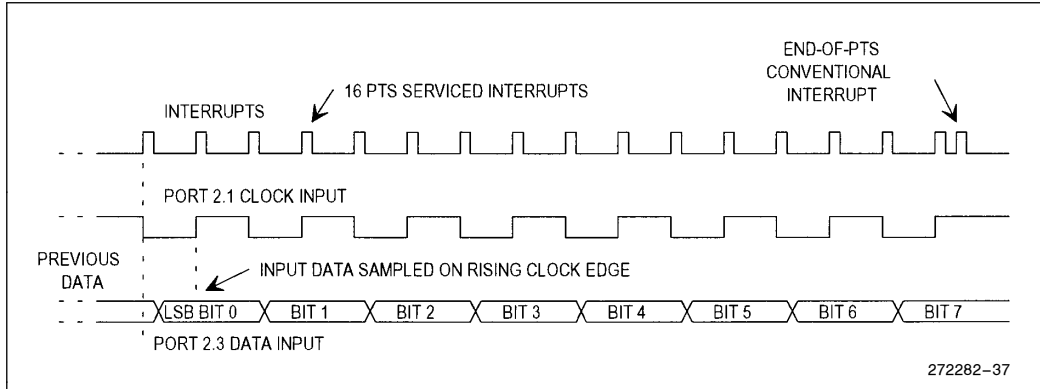


Figure 1-16. Synchronous SIO Receive Mode Timing

1.8.7 PTS SYNCHRONOUS SIO TRANSMIT MODE

To set up the PTS synchronous SIO transmit mode we need an EPA module to set up the timing interval between bits. This same module can either generate a shift clock output, or input an external shift clock.

During transmit operation, the EPA channel captures (or generates) the shift clock. Every other edge is used to shift the value of the TXD pin out of the DATA register. Either even or odd edges can accomplish this shift, depending on system requirements.

After all bits have been transmitted, the end-of-PTS interrupt is generated. The PTSCB's must then be reinitialized.

Figure 1-17 illustrates the sampling process during Synchronous SIO mode. Chapter 2 contains a software example of this mode.

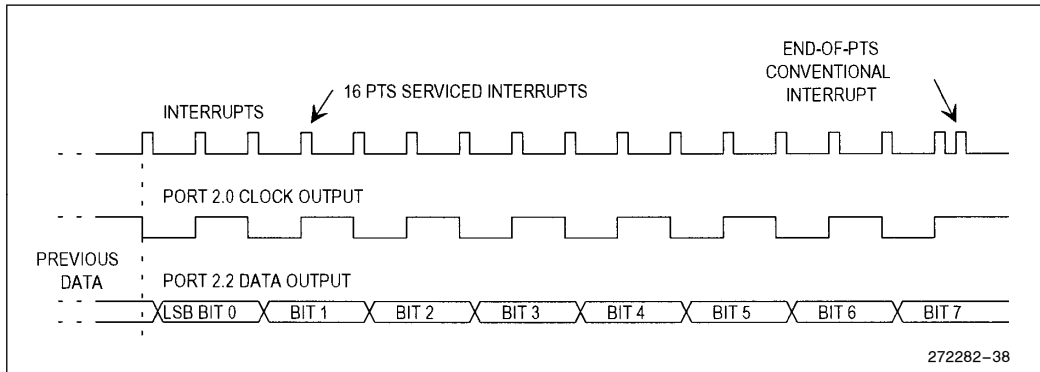


Figure 1-17. Synchronous SIO Transmit Mode Timing

2.0 SYNCHRONOUS SERIAL DATA TRANSMISSION PROGRAM SAMPLE

2.1 Introduction

The synchronous serial data transmit function on the 8XC196MC/MD is performed in software, using the PTS SSIO mode with an EPA channel. In this program example the EPA CAPCOMP0 channel is set-up to generate a shift clock output (SCK) on P2.0. Each time a timer match is made between CAPCOMP0 and TIMER1, P2.0 is toggled and an interrupt is generated. On every other interrupt, the PTS outputs the next bit of data on the output pin, in this case P2.2. The time interval between interrupts establishes the baud rate. To transmit 8 data bits, a total of 16 PTS interrupt and one conventional (end-of-PTS interrupt) cycles occur. Figure 1-17 illustrates the timing of this mode of operation.

2.1.1 END-OF PTS INTERRUPT

The final interrupt (when PTSCOUNT = 0) is called the end-of PTS interrupt. This interrupt will take the conventional interrupt vector to CAPCOMP0_INT, and the PTS control block is serviced here. In this example, PTSCOUNT is reloaded with 16, PTSCON1 is reloaded (this is required), and DATA0_W0_L is loaded with the next data byte. The EPA CAPCOMP0 channel is reloaded, and this “primes the pump” to start the next data byte transmission. A total of 16 bytes are transmitted.

2.2 Detailed Program Description

2.2.1 CONSTANT DECLARATIONS (Lines 1–46)

This section of the code defines the location of the PTS control block (PTSCB) registers, first as accessed through 64-byte window 4 (WSR = 24H) and then in their absolute locations (0100H–010EH). By accessing the control block through the window, loading and servicing of the PTSCB is much faster, and requires less code space. Note that all PTSCB's must be located on a quad-word boundary (divisible by 8).

Lines 39–41 define data storage for a transmit character buffer, a loop counter, and a flag register.

2.2.2 INTERRUPT VECTORS AND CCB (Lines 47–66)

Lines 47–49 fill the interrupt vector table location 2004H with the address of the conventional CAPCOMP0 interrupt service routine. This routine is known as the “end-of-PTS” interrupt routine.

Lines 55–58 define the chip configuration bytes, CCB and CCB1. These need to be configured for the particular system that this program is run on.

Lines 64–65 fill the PTS interrupt vector table location 2044H (EPA CAPCOMP0 PTS interrupt) with the base address of the PTSCB.

2.2.3 MAIN PROGRAM (Lines 67–135)

Lines 71–82 define the program starting location (2080H), set up the stack at 0200H, and disable and clear out all pending interrupts.

Lines 84–93 initialize I/O port 2 (used for the clock and TXD outputs) and set up timer1 for the time base. This is done through the windowed address of the special Function Registers (SFRs). In line 89, the port pins are initialized to 1, so the initial output value of SCK (P2.2) and TXD (P2.0) will be high. Because the CAPCOMP module (which generates SCK) is used in the “toggle” mode, the initial value of P2.2 is critical to the polarity of the clock signal.

Lines 96–105 initialize most of the PTSCB. Line 106 enables the interrupt mask bit for the CAPCOMP0 module.

Line 111 defines the number of bytes to transmit and line 112 clears the transfer done flag, indicating that the transmit operation is not yet complete.

Line 113 fills in the number of bits to be transmitted in each word (in this case 8 bits). Note that the value loaded into `PTSCOUNT0_W0` is twice the number of bits to be transmitted. This number also is the number of PTS cycles that will occur for each word transmitted.

Line 114 sets up the synchronization mode for the output data. The data can be made to change on EVEN numbered PTS cycles (`PTSCON10_W0 = 0`), or on ODD numbered PTS cycles (`PTSCON10_W0 = 2`).

Line 115 gets the first data byte from the transmit buffer `TRANSMIT` and loads it into the `PTSCB DATA0_W0_L` register. At this point, the entire `PTSCB` has been initialized, and is ready for action!

Line 117 unmask the PTS interrupt for `CAPCOMP0`.

Line 119 sets up the `CAPCOMP0` module in the compare/toggle output mode by loading `70H` to the `CAPCOMP0_CON_W0` register.

Line 120 starts the operation of the `CAPCOMP0` module by writing the time of the first interrupt to `CAPCOMP0_TIME_W0`.

Note that to do this, the current `TIMER1` value is read `80H` is added to it, and the result is stored into `CAPCOMP0_TIME_W0`. The value of `80H` determines how long before the first PTS interrupt occurs. The user must finish any initialization before this happens.

Lines 122–123 enable the PTS and conventional interrupts.

Lines 129–132 is a “do nothing” loop that monitors the `TXDDONE` flag and waits for the transmit operation to complete sending 16 bytes (as determined by `T_COUNT`). When the operation is complete, the NOPs are executed, and the user needs to place the next segment of code here.

2.2.4 END OF PTS INTERRUPT ROUTINE

The end-of PTS interrupt routine is entered immediately after the final PTS interrupt. It performs housekeeping activities as detailed below.

Lines 141–144 `PUSH` the CPU status and interrupt masks onto the stack, disables `CAPCOMP0` and clears any pending interrupt it may have generated. Disabling `CAPCOMP0` is an important point, as the final PTS cycle loads the next `SCK` toggle time into `CAPCOMP0`. We do not want this toggle to occur, or else the clock polarity will change due to the ODD number of toggles, and/or erroneous data may be output.

Line 145–146 check if 16 bytes have been transmitted yet. If yes, the program returns; otherwise, execution continues.

Lines 149–155 “refresh” the `PTSCB` registers that have changed. The next data byte is loaded into `DATA0_W0_L`, `PTSCOUNT0_W0` and `PTSCON10_W0` are reloaded, and the PTS service is re-enabled by setting the `PTSSEL` bit for `CAPCOMP0`.

Lines 156–158 restart the `CAPCOMP0` module by loading `CAPCOMP0_CON_W0` and `CAPCOMP0_TIME_W0`. Note that the time loaded into `CAPCOMP0_TIME_W0` determines how long before the next data transmission occurs.

Finally, lines 160–162 return program execution to the main program flow.

2.3 Top 5 Issues of Using the PTS/SSIO Transmit Mode

1. The `SCK` clock output must be from a `CAPCOMP` or `COMP` module output pin.
2. The port pins used for `SCK` and `TXD` must be initialized to the system-required logic level prior to starting any data transmissions.
3. The first PTS cycle must be manually started before the transfer will begin (see line 120). Once started, the bits will shift out until the end-of-PTS interrupt occurs, or the PTS is disabled.

4. The end-of-PTS interrupt routine must disable the CAPCOMP0 module used for SCK and clear the corresponding interrupt pending bit (see lines 143–144).
5. The end-of-PTS interrupt routine must reload the DATA0, PTSCOUNT0 and PTSCON10 registers in the PTSCB. The CAPCOMP1 PTSSEL bit must be set and CAPCOMP0_CON and CAPCOMP0_TIME must be reloaded before additional data can be transmitted.

2.4 Program Example

```

1  ;*****
2  ; PTSCB address in vertical window 24H
3  ;*****
4  ;
5  PTSCOUNT0_WO EQU          0C0H:BYTE
6  PTSCONO_WO   EQU          0C1H:BYTE
7  EPAREGO_WO   EQU          0C2H:WORD
8  BAUDCONSTO_WO EQU        0C4H:WORD
9  PTSVEC10_WO  EQU          0C6H:WORD
10 PORTREGO_WO  EQU          0C8H:WORD
11 PORTMASKO_WO EQU         0CAH:BYTE
12 PTSCON10_WO  EQU          0CBH:BYTE
13 DATA0_WO    EQU          0CCH:WORD
14 SAMPTIMEO_WO EQU         0CEH:WORD
15 ;
16 DATA0_WO_L  EQU          0CCH:BYTE
17 T1CONTROL_WO EQU         038H:BYTE
18 ;*****
19 ; absolute address of PTSCB
20 ;*****
21 ;
22 PTSCOUNT0    EQU          100H:BYTE
23 PTSCONO      EQU          101H:BYTE
24 EPAREGO      EQU          102H:WORD
25 BAUDCONSTO   EQU         104H:WORD
26 PTSVEC10     EQU          106H:WORD
27 PORTREGO     EQU          108H:WORD
28 PORTMASKO    EQU          10AH:BYTE
29 PTSCON10     EQU          10BH:BYTE
30 DATA0       EQU          10CH:WORD
31 SAMPTIMEO    EQU          10EH:WORD
32 T1CONTROL     EQU         1F78H:WORD
33 ;*****
34 ; USER DEFINED REGISTERS
35 ;*****
36 ;
37 RSEG  AT          LAH
38 ;
39 TRANSMIT:  DSB          17      ;transfer data area
40 T_COUNT:   DSW          1       ;transfer data index counter
41 TXDDONE:   DSB          1       ;transfer done flag
42 ;
43 ;*****
44 ; INTERRUPT VECTOR TABLE
45 ;*****
46 ;
47 CSEG  AT          2004H      ;capture/compare module 0 interrupt
                                   ;location

```

2.4 Program Example (Continued)

```

48
49             DCW    CAPCOMP0_INT    ;INT 02
50 ;
51 ;*****
52 ; CHIP CONFIGURATION BYTES
53 ;*****
54 ;
55 CSEG    AT                2018H
56
57             DCW    20CFH            ;CCB
58             DCW    20DCH            ;CCB1
59 ;
60 ;*****
61 ; PTS VECTOR TABLE
62 ;*****
63 ;
64 CSEG    AT                2044H            ;capture/compare module 0 PTSCB
                                           ;location
                                           ;PTSCB vector address
65             DCW    100H
66 ;
67 ;*****
68 ; MAIN ROUTINE
69 ;*****
70 ;
71 CSEG    AT                2080H
72
73 MAIN_START:
74         DI                ;disable interrupt
75         LD    SP,#0200H    ;set-up stack pointer
76 ;
77 ;Clear interrupt mask register
78 ;
79         CLRB    INT_MASK    ;reset interrupt mask register
80         CLRB    INT_MASK1
81         LDB    WSR,#3EH    ;map 64 bytes to 1F80H-1FBFH
82         CLRB    PI_MASK_WO ;reset peripheral interrupt mask reg.
83 ;
84 ;Initialize port & timer
85 ;
86         LDB    WSR,#3FH    ;map 64 bytes to 1FC0H-1FFFH
87         LDB    P2_MODE_WO,RO ;P2.0-P2.7=LSIO
88         LDB    P2_DIR_WO,#00H ;P2.0-P2.7=OUTPUT
89         LDB    P2_REG_WO,#OFFH ;P2.0-P2.7=high
90         LDB    P2_MODE_WO,#01H ;P2.0=EPA capture compare SCK send
91 ;
92         LDB    WSR,#3DH    ;map 64 bytes to 1F40H-1F7FH
93         LDB    T1CONTROL_WO,#0COH ;timer 1 enable, up count, clock
94                                     ;internal, pre-scale=div 1
    
```

2.4 Program Example (Continued)

```

95 ;
96 ;Initialize PTSCB Mode
97 ;
98     LDB     WSR,#24H           ;map 64 bytes to 100H-13FH
99     LDB     PTSCONO_WO,#72H   ;SSIO transfer
100    LD      EPAREGO_WO,#1F42H ;EPA capture/compare0 register address
101    LD      BAUDCONST0_WO,#0D0H ;set baud rate
102                                ; 9600 baud
103    LD      PTSVEC10_WO,#108H  ;pointer to PTSCB1
104    LD      PORTREGO_WO,#1FD4H ;Port 2 contains TXD pin
105    LDB     PORTMASKO_WO,#04H  ;P2.2=TXD
106    ORB     INT_MASK,#04H      ;enable interrupt on capture/compare
107                                ;module 0
108 ;
109 ;Set transfer data
110 ;
111     LD      T_COUNT,#16        ;set transfer data count
112     CLRB   TXDDONE            ;clear transfer done flag
113     LDB     PTSCOUNT0_WO,#10H  ;10H = (# of bits)*2
114     LDB     PTSCON10_WO,#02H   ;PTSCON1 - transfer on 1st PTS cycle
115     LDB     DATA0_WO_L,TRANSMIT[T_COUNT]
116                                ;set transfer data
117     ORB     PTSEL,#04H         ;enable PTS on capture/compare module 0
118     LDB     WSR,#3DH           ;map 64 bytes to 1F40H-1F7FH
119     LDB     CAPCOMPO_CON_WO,#70H ;Compare - interrupt and toggle
120     ADD     CAPCOMPO_TIME_WO,TIMER1_WO,#80H
121                                ;set first PTS & delay time
122     EPTS                                ;enable PTS
123     EI                                ;enable interrupt
124 ;
125 ;*****
126 ;Polling routine
127 ;*****
128 ;
129 LOOP:
130     JBC     TXDDONE,0,LOOP      ;check transfer done flag
131     NOP                                ;dummy command
132     NOP                                ;dummy command
133 ;     -                                ;further user code here
134 ;     -
135 ;

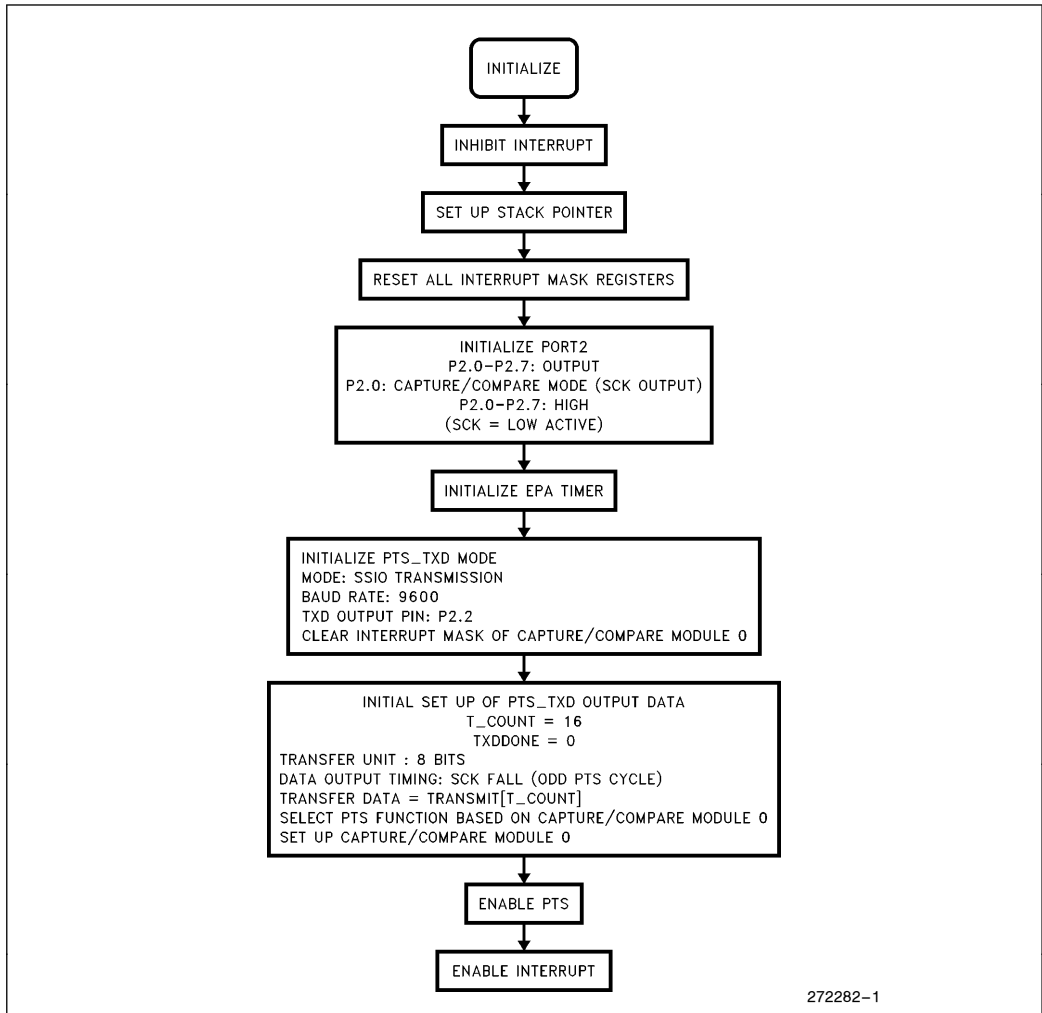
```


2.4 Program Example (Continued)

```

136 ;*****
137 ;end of PTS interrupt service routine
138 ;*****
139 ;
140 CAPCOMPO_INT:                end-of-PTS interrupt routine
141     PUSHA                    ;save PSW,INT_MASK,INT_MASK1,WSR
142     LDB    WSR,#3DH           ;map 64 bytes to 1F40H-1F7FH
143     LDB    CAPCOMPO_CON_WO,#00H ;disable capture/compare module 0
144     ANDB  INT_PEND,#0FBH     ;clear false pending interrupt
145     DJNZW T_COUNT,CAPCOMPO_SKIP ;decrement data counter & check end of
                                ;transfer
146     LDB    TXDDONE,#01H      ;set done flag
147     BR     CAPCOMPO_RET      ;exit, entire buffer sent
148 CAPCOMPO_SKIP:              ;prepare to xmit next word
149     LDB    WSR,#24H           ;map 64 bytes to 100H-13FH
150     LDB    DATAO_WO_L,TRANSMIT[T_COUNT]
151                                ;set next transfer data
152     LDB    PTSCOUNT0_WO,#10H  ;(# of bits)*2
153     LDB    PTSCON10_WO,#02H  ;PTSCON1 - transfer on 1st PTS cycle
154                                ;data to be transmitted
155     ORB    PTSEL,#04H        ;enable PTS on capture/compare module 0
156     LDB    WSR,#3DH           ;map 64 bytes to 1F40H-1F7FH
157     LDB    CAPCOMPO_CON_WO,#70H ;Compare - interrupt and toggle
158     ADD    CAPCOMPO_TIME_WO,EPA_TIMER1_WO,#80H
159                                ;set first PTS & delay time
160 CAPCOMPO_RET:
161     POPA                    ;load PSW,INT_MASK,INT_MASK1,WSR
162     RET                      ;return to main loop from
163                                ;interrupt service routine
164 END

```



272282-1

Figure 2-1. Flow Chart—SSIO Transmit Initialization

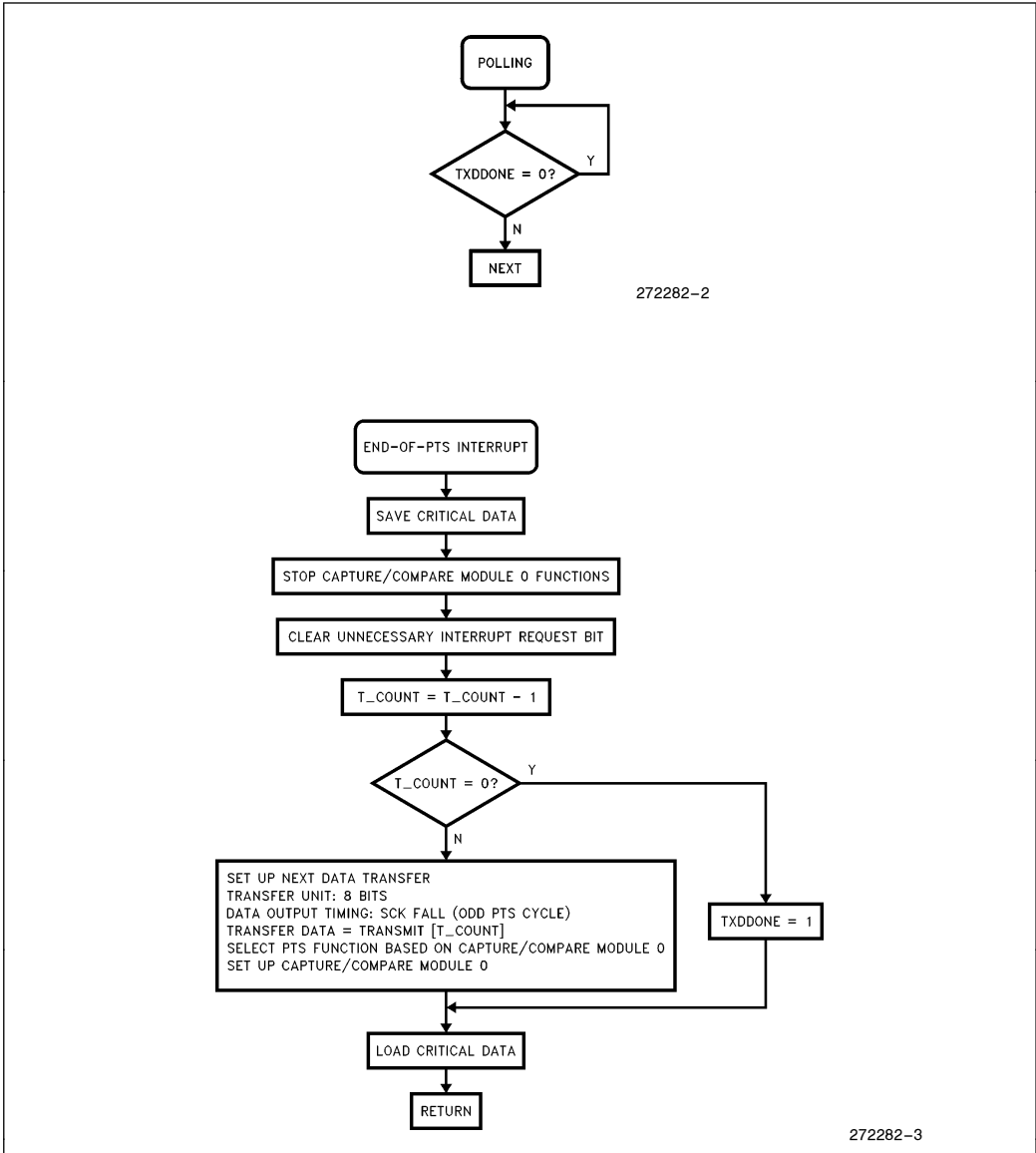


Figure 2-2. Flow Chart—SSIO Transmit Interrupt Routine

3.0 SYNCHRONOUS SERIAL DATA RECEPTION PROGRAM EXAMPLE

3.1 Introduction

The synchronous serial data receive function on the 8XC196MC/MD is performed in software using the PTS SSIO mode with an EPA channel. In this program example the EPA CAPCOMP1 channel is used in the capture mode to

receive a shift clock input on P2.1. Each time P2.1 is toggled, CAPCOMP1 generates an interrupt. The interrupts are processed by the PTS, and on every other interrupt the PTS cycle reads the value on the input pin, in this case P2.3. The values are shifted into the DATA1_W0_H register PTS control block. The time interval between interrupts establishes the baud rate. To receive 8 data bits, a total of 16 PTS cycles and one conventional (end-of-PTS interrupt) interrupt occur.

Because this example uses an external clock input, TIMER1 and the BAUDCONST1 register are not used. Note that by changing the CAPCOMP1 module to compare mode, it would be possible for the synchronous clock to be generated by the 8XC196MC and output to the external device. In this case the baud rate will be determined by TIMER1 and BAUDCONST1.

Also note that the program flow is very similar to the PTS SSIO transmit mode.

3.1.1 END-OF PTS INTERRUPT

The final interrupt (when PTSCOUNT = 0) is called the end-of PTS interrupt. This interrupt will take the conventional interrupt vector to CAPCOMP1_INT, and the PTS control block is serviced here. In this example, DATA1_W0_H (which contains the incoming character) is stored in the RECEIVE buffer, PTSCOUNT is reloaded with 16, PTSCON1 is reloaded (this is required), and the input buffer DATA1_W0 is cleared. The EPA CAPCOMP1 channel is reloaded, and this “primes the pump” to start the next data byte reception. A total of 16 bytes are received.

3.2 Detailed Program Description

3.2.1 CONSTANT DECLARATIONS (Lines 1–42)

This section of the code defines the location of the PTS control block (PTSCB) registers, first as accessed through 64-byte window 4 (WSR = 24H) and then in their absolute locations (0110H–011EH). By accessing the control block through the window, loading and servicing of the PTSCB is much faster, and requires less code space. Note that all PTSCB's must be located on a quad-word boundary (divisible by 8).

Lines 37–41 define data storage for a receive character buffer, a loop counter, and a flag register.

3.2.2 INTERRUPT VECTORS AND CCB (Lines 43–66)

Lines 47–49 fill the interrupt vector table location 2008H with the address of the conventional CAPCOMP1 interrupt service routine. This routine is known as the “end-of-PTS” interrupt routine.

Lines 55–58 define the chip configuration bytes, CCB and CCB1. These need to be configured for the particular system that this program is run on.

Lines 64–65 fill the PTS interrupt vector table location 2048H (EPA CAPCOMP1 PTS interrupt) with the base address of the PTSCB.

3.2.3 MAIN PROGRAM (Lines 67–135)

Lines 71–82 define the program starting location (2080H), set up the stack at 0200H, and disable and clear out all pending interrupts.

Lines 84–93 initialize I/O port 2 (used for the clock and RXD inputs) and set up timer1 for the time base. This is done through the windowed address of the special Function Registers (SFRs). In line 90, the unused port pins are initialized to 1. Because the CAPCOMP1 module (which inputs SCK) is used in the “toggle” mode, the initial value input to P2.1 is critical to the polarity of the clock signal.

Lines 97–105 initialize most of the PTSCB. Line 106 enables the interrupt mask bit for the CAPCOMP1 module. Line 111 defines the number of bytes to receive and line 112 clears the transfer done flag, indicating that the receive operation is not yet complete.

Line 113 fills in the number of bits to be received for each word (in this case 8 bits). Note that the value loaded into PTSCOUNT1_W0 is twice the number of bits to be received. This number also is the number of PTS cycles that will occur for each word transmitted.

Line 114 sets up the synchronization mode for the input data. The data is sampled on EVEN numbered PTS cycles (PTSCON11_W0 = 0).

Line 115 clears the receive register DATA1_W0 in the PTSCB. At this point, the entire PTSCB has been initialized, and is ready for action!

Line 117 unmask the PTS interrupt for CAPCOMP1.

Line 118 sets up the CAPCOMP1 module in the capture both edges mode by loading 30H to the CAPCOMP1_CON_W0 register.

Lines 119–120 enable the PTS and conventional interrupts.

Lines 126–129 are a “do nothing” loop that monitors the RXDDONE flag and waits for the transmit operation to complete sending 16 bytes. When the operation is complete, the NOPS are executed, and the user needs to place the next segment of code here.

3.2.4 END-OF-PTS INTERRUPT ROUTINE

The end-of-PTS interrupt routine is entered immediately after the final PTS interrupt. It performs housekeeping activities as detailed below.

Lines 137–144 PUSH the CPU status and interrupt masks onto the stack, disables CAPCOMP1 and clears any pending interrupt it may have generated.

Line 145–146 check If 16 bytes have been received yet. If yes, the program returns; otherwise, execution continues.

Lines 149–153 “refresh” the PTSCB registers that have changed. DATA1_W0 is cleared, PTSCOUNT1_W0 and PTSCON11_W0 are reloaded, and the PTS service is re-enabled by setting the PTSSEL bit for CAPCOMP1.

Line 154 readies the CAPCOMP1 module to capture the next edge by loading CAPCOMP1_CON_W0.

Finally, lines 155–157 return program execution to the main program flow.

3.3 Top 5 Issues of Using the PTS/SSIO Receive Mode

1. SCK can be either input to or output from the 8XC196MC/MD. If SCK is an output, either a CAPCOMP or COMP module can be used. If SCK is an input, a CAPCOMP module must be used.
2. The port pin used for SCK (if an output) must be initialized to the system-required logic level prior to starting any data transmissions.
3. The data reception will begin as soon as the clock pulses begin. If the 8XC196MC/MD is generating SCK, the first PTS cycle must be manually started before the transfer will begin. Once started, the bits will shift in until the end-of-PTS interrupt occurs, or the PTS is disabled.
4. The end-of-PTS interrupt routine must disable the CAPCOMP module used for SCK and clear the corresponding interrupt pending bit (see lines 140–141)
5. The end-of-PTS interrupt routine must clear the DATA register, and reinitialize PTSCOUNT1 and PTSCON11 registers in the PTSCB. The CAPCOMP1 PTSSEL bit must be set and the CAPCOMP1_CON must be reloaded before additional data can be received. Additionally, if the 8XC196MC/MD is generating (outputting) SCK, CAPCOMP1_TIME would need to be reloaded.

3.4 Program Example

```

1  ;*****
2  ; PTSCB address in vertical window 24H
3  ;*****
4  ;
5  PTSCOUNT1_WO EQU          0D0H:BYTE
6  PTSCON1_WO   EQU          0D1H:BYTE
7  EPAREG1_WO   EQU          0D2H:WORD
8  BAUDCONST1_WO EQU        0D4H:WORD
9  PTSVEC11_WO  EQU          0D6H:WORD
10 PORTREG1_WO  EQU          0D8H:WORD
11 PORTMASK1_WO EQU         0DAH:BYTE
12 PTSCON11_WO  EQU          0DBH:BYTE
13 DATA1_WO    EQU          0DCH:WORD
14 SAMPTIME1_WO EQU         0DEH:WORD
15 ;
16 DATA1_WO_H  EQU          0DDH:BYTE
17 ;
18 ;*****
19 ; absolute address of PTSCB
20 ;*****
21 ;
22 PTSCOUNT1    EQU          110H:BYTE
23 PTSCON1     EQU          111H:BYTE
24 EPAREG1     EQU          112H:WORD
25 BAUDCONST1  EQU          114H:WORD
26 PTSVEC11    EQU          116H:WORD
27 PORTREG1    EQU          118H:WORD
28 PORTMASK1   EQU          11AH:BYTE
29 PTSCON11    EQU          11BH:BYTE
30 DATA1      EQU          11CH:WORD
31 SAMPTIME1   EQU          11EH:WORD
32 ;
33 ;*****
34 ; USER DEFINED REGISTERS
35 ;*****
36 ;
37 RSEG        AT           1AH
38 ;
39 RECEIVE:     DSB          17    ;receive data buffer
40 R_COUNT:     DSW          1     ;receive character counter
41 RXDDONE:     DSB          1     ;transfer done flag
42 ;
43 ;*****
44 ; INTERRUPT VECTOR TABLE
45 ;*****
46 ;
47 CSEG        AT           2008H   ;capture/compare module 1 INT. location
48
49             DCW          CAPCOMPL_INT ;INT 04
50 ;
51 ;*****
52 ; CHIP CONFIGURATION BYTES
53 ;*****
54 ;

```

3.4 Program Example (Continued)

```

55  CSEG  AT                2018H
56
57          DCW  20CFH      ;CCB
58          DCW  20DCH      ;CCB1
59  ;
60  ;*****
61  ; PTS VECTOR TABLE
62  ;*****
63  ;
64  CSEG  AT                2048H      ;capture/compare module 1 PTSCB
                                   ;location
                                   ;PTSCB vector address
65          DCW  110H
66  ;
67  ;*****
68  ; MAIN ROUTINE
69  ;*****
70  ;
71  CSEG  AT                2080H
72
73  MAIN_START:
74          DI                ;disable interrupt
75          LD  SP,#0200H     ;set-up stack pointer
76  ;
77  ;Clear interrupt mask register
78  ;
79          CLRB INT_MASK     ;reset interrupt mask register
80          CLRB INT_MASK1
81          LDB WSR,#3EH      ;map 64 bytes to 1F80H-1FBFH
82          CLRB PI_MASK_WO   ;reset peripheral interrupt mask reg.
83  ;
84  ;Initialize port & timer
85  ;
86          LDB WSR,#3FH      ;map 64 bytes to 1FC0H-1FFFH
87          LDB P2_MODE_WO,RO  ;P2.0-P2.7=LSIO
88          LDB P2_DIR_WO,#0AH ;OUTPUTP=2.0,P2.2,P2.4-P2.7
89                                   ;INPUT=P2.1,P2.3
90          LDB P2_REG_WO,#0FFH ;P2.0-P2.7=High
91          LDB P2_MODE_WO,#02H ;P2.1=EPA capture compare SCK receive
92  ;
93          LDB WSR,#3DH      ;map 64 bytes to 1F40H-1F7FH
94  ;
95  ;
96  ;
97  ;Initialize RXD Mode
98  ;
99          LDB WSR,#24H      ;map 64 bytes to 0100H-013FH
100         LDB PTSCON1_WO,#32H ;SSIO receive
101         LD  EPAREG1_WO,#1F46H ;CAPCOMP1_TIME address
102         LD  BAUDCONST1_WO,#00DOH ;set baud rate
103         LD  PTSVEC11_WO,#118H ;pointer to PTSCB1
104         LD  PORTREG1_WO,#1FD6H ;Port 2 has RXD pin
105         LDB PORTMASK1_WO,#08H ;P2.1=SCK, P2.3=RXD
106         ORB INT_MASK,#10H   ;enable interrupt on capture/compare
107                                   ;module 1
108  ;
    
```

3.4 Program Example (Continued)

```

109 ;Set receive mode
110 ;
111 LD R_COUNT,#16 ;set receive data count
112 CLRB RXDDONE ;clear done flag
113 LDB PTSCOUNT1_WO,#10H ;(# of bits)*2
114 LDB PTSCON11_WO,#00H ;PTSCON1 - receive on 2nd PTS cycle
115 CLR DATA1_WO ;clear receive data buffer
116 ORB PTSSEL,#10H ;enable PTS on capture/compare mod. 1
117 LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
118 LDB CAPCOMPL_CON_WO,#30H ;capture on both edge
119 EPTS ;enable PTS
120 EI ;enable interrupt
121 ;
122 ;*****
123 ;Polling routine
124 ;*****
125 ;
126 LOOP:
127 JBC RXDDONE,0,LOOP ;check receive done flag
128 NOP ;dummy command
129 NOP ;dummy command
130 ; - ;further user code here
131 ; -
132 ;
133 ;*****
134 ;end of PTS interrupt service routine
135 ;*****
136 ;
137 CAPCOMPL_INT: end-of-PTS interrupt routine
138 PUSHA ;save PSW,INT_MASK,INT_MASK1,WSR
139 LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
140 LDB CAPCOMPL_CON_WO,#00H ;disable capture/compare module 1
141 ANDB INT_PEND,#0EFH ;clear false pending interrupt
142 LDB WSR,#24H ;map 64 bytes to 0100H-013FH
143 STB DATA1_WO_H,RECEIVE[R_COUNT]
144 ;save received data
145 DJNZW R_COUNT,CAPCOMPL_SKIP ;decrement data counter & check end of
;receive
146 LDB RXDDONE,#01H ;set done flag
147 BR CAPCOMPL_RET ;finish data receive
148 CAPCOMPL_SKIP:
149 LDB PTSCOUNT1_WO,#10H ;(# of bits)*2
150 LDB PTSCON11_WO,#00H ;PTSCON1 - receive on 2nd PTS cycle
151 CLR DATA1_WO ;clear receive data buffer
152 ORB PTSSEL,#10H ;enable PTS on capture/compare module 1
153 LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
154 LDB CAPCOMPL_CON_WO,#30H ;capture on both edge
155 CAPCOMPL_RET:
156 POPA ;load PSW,INT_MASK,INT_MASK1,WSR
157 RET ;return to main loop
158
159 END

```

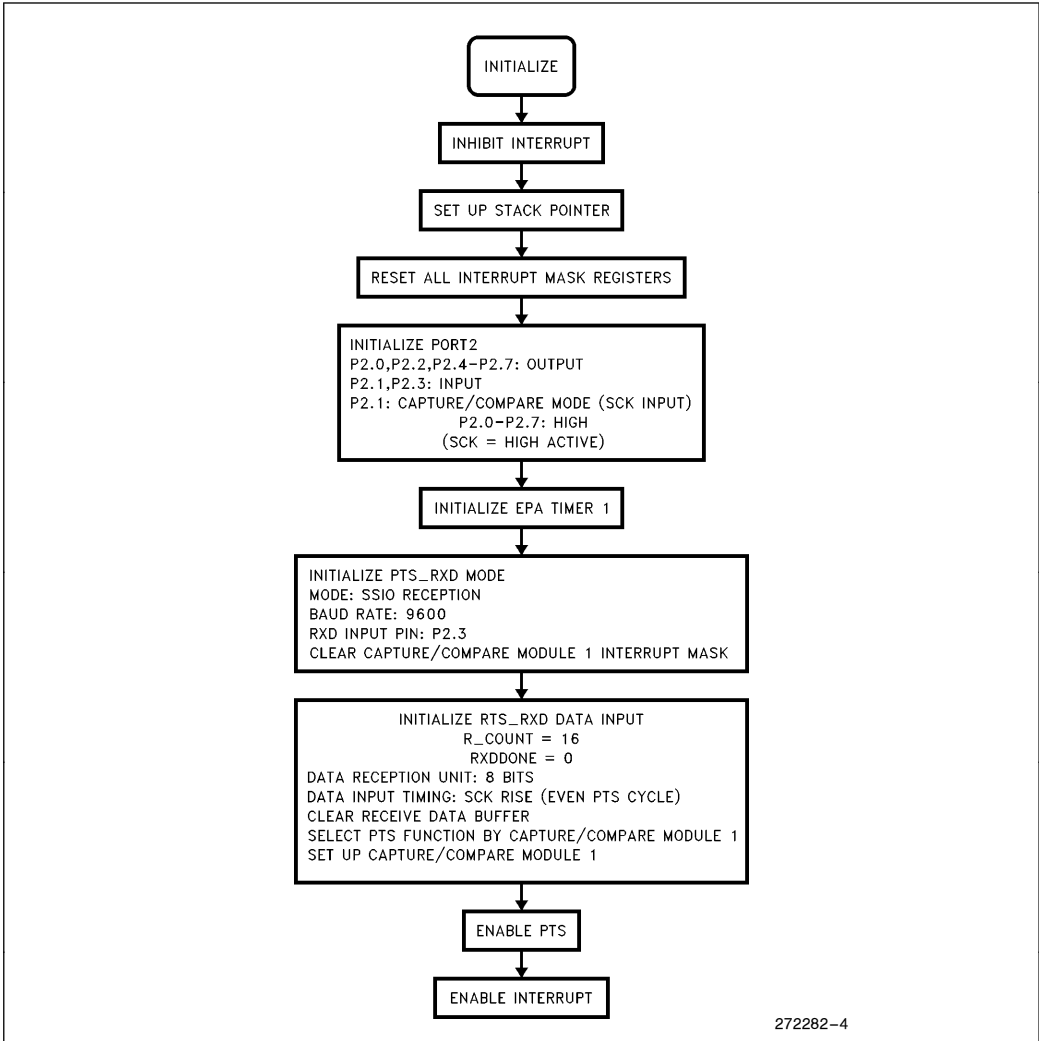



Figure 3-1. Flow Chart—SSIO Receive Initialization

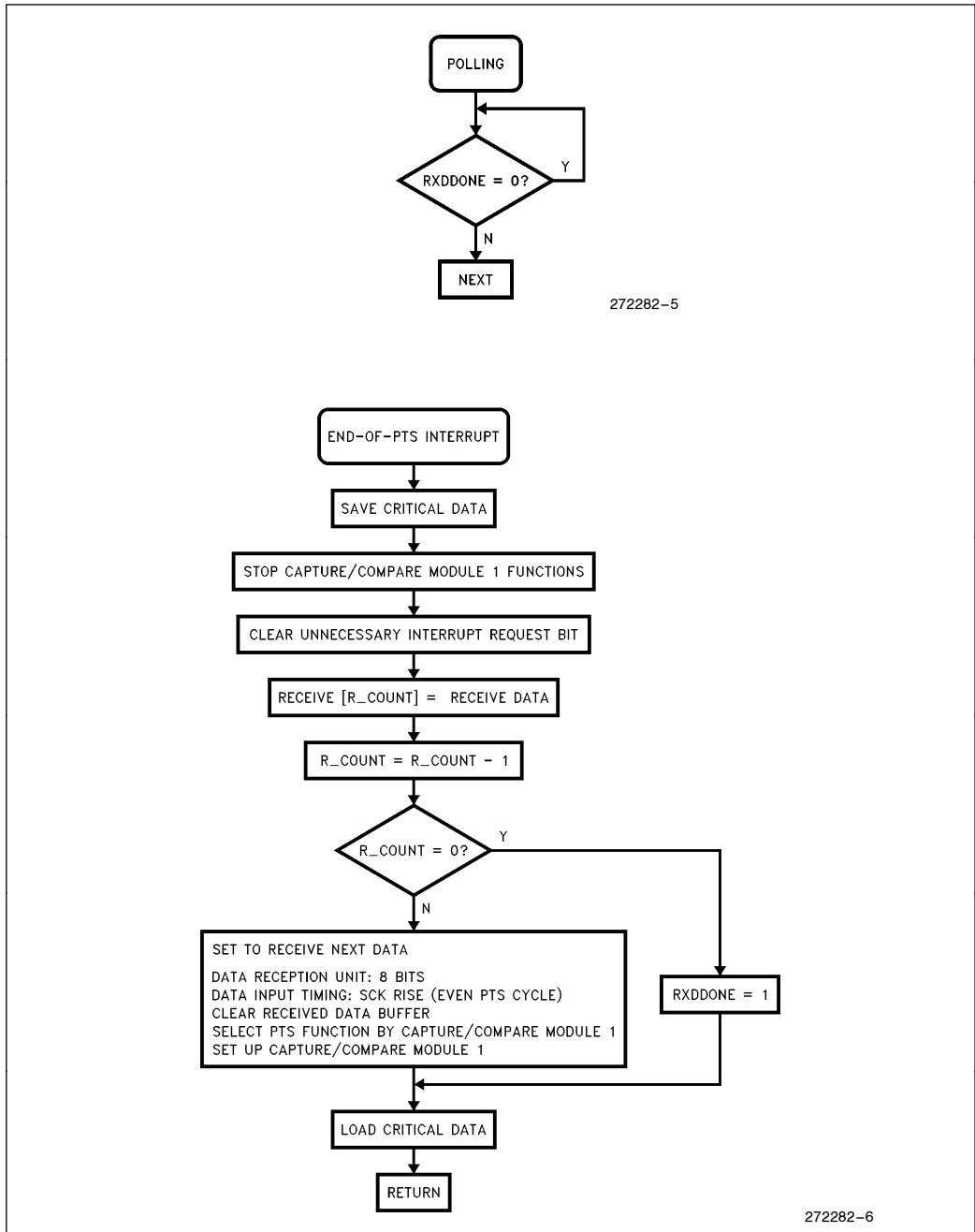


Figure 3-2. Flow Chart—SSIO Receive Interrupt Routine

4.0 ASYNCHRONOUS SERIAL DATA TRANSMISSION PROGRAM EXAMPLE

4.1 Introduction

The asynchronous serial data transmit function on the 8XC196MC/MD is performed in software, using the PTS SIO mode with an EPA channel. In this program example, the EPA CAPCOMP0 channel is set-up to generate the time base for outputting serial data, thus determining the baud rate.

Transmissions are started by clearing the output pin which generates the “start” bit (0). The CAPCOMP0 module is loaded with the time at which the first data bit should be driven onto the port. This time must correspond to 1 “bit time” for the baud rate being used.

Each time a timer match is made between CAPCOMP0 and TIMER0, an interrupt is generated. The PTS outputs the next bit of data on the output pin, in this case P2.0. The asynchronous transmit mode automatically transmits up to 15 data bits, followed by an optional parity bit, and terminated by a “stop” bit (1). If the parity bit is sent, a maximum of 14 data bits may be sent (data + parity + stop = 16 maximum). To transmit 8 data bits with parity, a total of 10 PTS interrupt and one conventional (end-of-PTS interrupt) cycles occur.

Note that data to be transmitted is right-justified in the PTSCB DATA0 register, and will be shifted out least significant (rightmost) bit first.

The number of data bits that will be transmitted is determined by the number of PTS cycles, PTSCOUNT. PTSCOUNT must equal data bits + parity bit + stop bit.

4.1.1 END-OF-PTS INTERRUPT

The final interrupt (when PTSCOUNT = 0) is called the end-of-PTS interrupt. This interrupt occurs immediately after the stop bit is output, and takes the conventional interrupt vector to CAPCOMP0__INT, where the PTS control block is serviced. In this example, DATA0__W0__L is loaded with the next data byte, PTSCOUNT is reloaded with 10, and PTSCON10 is reloaded (this is required). Next, P2.0 is cleared creating the “start” bit for the next data word to be transmitted. The EPA CAPCOMP0 channel is initialized, and CAPCOMP0__TIME is written to, establishing the time at which the first data bit (BIT0, LSB) of the next word will be driven out. A total of 16 bytes are transmitted.

4.2 Detailed Program Description

4.2.1 CONSTANT DECLARATIONS (Lines 1–42)

This section of the code defines the location of the PTS control block (PTSCB) registers, first as accessed through 64-byte window 4 (WSR = 24H) and then in their absolute locations (0100H–010EH). By accessing the control block through the window, loading and servicing of the PTSCB is much faster, and requires less code space. Note that all PTSCB's must be located on a quad-word boundary (divisible by 8).

Lines 37–41 define data storage for a transmit character buffer, a loop counter, and a flag register.

4.2.2 INTERRUPT VECTORS AND CCB (Lines 43–66)

Lines 47–49 fill the interrupt vector table location 2004H with the address of the conventional CAPCOMP0 interrupt service routine. This routine is known as the “end-of-PTS” interrupt routine.

Lines 55–58 define the chip configuration bytes, CCB and CCBI. These need to be configured for the particular system that this program is run on.

Lines 64–65 fill the PTS interrupt vector table location 2044H (EPA CAPCOMP0 PTS interrupt) with the base address of the PTSCB.

4.2.3 MAIN PROGRAM (Lines 67–135)

Lines 71–82 define the program starting location (2080H), set up the stack at 0200H, and disable and clear out all pending interrupts.

Lines 84–92 initialize I/O port 2 (used for the TXD output) and set up timer1 for the time base. This is done through the windowed address of the special Function Registers (SFRs). In line 89, the port pins are initialized to 1, so the initial output value of TXD (P2.0) will be high.

Lines 94–102 initialize most of the PTSCB. Line 103 enables the interrupt mask bit for the CAPCOMP0 module.

Line 108 defines the number of bytes to transmit and line 109 clears the transfer done flag, indicating that the transmit operation is not yet complete.

Line 110 fills in the number of bits to be transmitted in each word (in this case 8 bits). This number also is the number of PTS interrupts that will occur for each word transmitted.

Line 112 sets up the SIO mode for odd parity.

Line 113 gets the first data byte from the transmit buffer TRANSMIT and loads it into the PTSCB DATA0_W0_L register. At this point, the entire PTSCB has been initialized, and is ready for action!

Line 115 unmask the PTS interrupt for CAPCOMP0.

Line 117 begins transmission of the “start” bit by clearing P2.0.

Line 119 sets up the CAPCOMP0 module in the compare mode by loading 40H to the CAPCOMP0_CON_W0 register.

Line 120 starts the operation of the CAPCOMP0 module by writing the time of the first interrupt to CAPCOMP0_TIME_W0. Note that to do this, the current TIMER1 value is read and 1A0H is added to it, and the result is stored into CAPCOMP0_TIME_W0. The value 1A0H determines how long before the first PTS interrupt occurs, which transmits the first bit. 1A0H equals 1 bit time at 9600 baud. The program must finish any initialization before this happens.

Lines 122–123 enable the PTS and conventional interrupts.

Lines 129–132 is a “do nothing” loop that monitors the TXDDONE flag and waits for the transmit operation to complete sending 16 bytes. When the operation is complete, the NOPS are executed, and the user needs to place the next segment of code here.

4.2.4 END-OF-PTS INTERRUPT ROUTINE

The end-of-PTS interrupt routine is entered immediately after the final PTS interrupt. It performs housekeeping activities as detailed below.

Lines 140–146 PUSH the CPU status and interrupt masks onto the stack, disables CAPCOMP0 and clears any pending interrupt it may have generated. Disabling CAPCOMP0 is an important point, as an additional PTS cycle would shift out an additional data bit before it is wanted.

Line 147 checks if 16 bytes have been transmitted yet. If yes, the program sets the TXDDONE flag and returns; otherwise, execution continues.

Lines 150–156 “refresh” the PTSCB registers that have changed. The next data byte is loaded into DATA0_W0_L, PTSCOUNT0_W0 and PTSCON10_W0 are reloaded.

Lines 157–160 issue the “start” bit for the next word by clearing P2.0, and enable the PTS service by setting the PTSSEL bit for CAPCOMP0.

Lines 161–164 restart the CAPCOMP0 module by loading CAPCOMP_CON_W0 and CAPCOMP0_TIME_W0. Note that the time loaded into CAPCOMP0_TIME_W0 determines how long before the next data bit is output, and must equal 1 bit time.

Finally, lines 165–167 return program execution to the main program flow.

4.3 Top 5 Issues of Using the PTS/SIO Transmit Mode

1. The port pins used for TXD must be initialized to one prior to starting any data transmissions.
2. A zero is written to the TXD port pin to begin transmission of the “start” bit.
3. The first PTS cycle must be manually started immediately after the “start” bit has been written (see line 119). Once started, the bits will shift out until the end-of-PTS interrupt occurs, or the PTS is disabled.
4. The end-of-PTS interrupt routine must disable the CAPCOMP module used for TXD and clear the corresponding interrupt pending bit (see lines 143–144).
5. The end-of-PTS interrupt routine must reload the DATA0, PTSCOUNT0 and PTSCON10 registers in the PTSCB. The port pin used for TXD must be cleared, thus generating the “start” bit. The CAPCOMP1 PTSSEL bit must be set and CAPCOMP0_CON and CAPCOMP0_TIME must be reloaded.

4.4 Program Example

```

1  ;*****
2  ; PTSCB address in vertical window 24H
3  ;*****
4  ;
5  PTSCOUNT0_WO EQU          0C0H:BYTE
6  PTSCONO_WO   EQU          0C1H:BYTE
7  EPAREGO_WO   EQU          0C2H:WORD
8  BAUDCONSTO_WO EQU        0C4H:WORD
9  PTSVECIO_WO  EQU          0C6H:WORD
10 PORTREGO_WO  EQU          0C8H:WORD
11 PORTMASKO_WO EQU         0CAH:BYTE
12 PTSCON10_WO EQU          0CBH:BYTE
13 DATAO_WO    EQU          0CCH:WORD
14 SAMPTIMEO_WO EQU         0CEH:WORD
15 ;
16 DATAO_WO_L  EQU          0CCH:BYTE
17 ;
18 ;*****
19 ; absolute address of PTSCB
20 ;*****
21 ;
22 PTSCOUNT0    EQU          100H:BYTE
23 PTSCONO      EQU          101H:BYTE
24 EPAREGO      EQU          102H:WORD
25 BAUDCONSTO   EQU          104H:WORD
26 PTSVECIO     EQU          106H:WORD
27 PORTREGO     EQU          108H:WORD
28 PORTMASKO    EQU          10AH:BYTE
29 PTSCON10     EQU          10BH:BYTE
30 DATAO       EQU          10CH:WORD
31 SAMPTIMEO    EQU          10EH:WORD
32 ;
33 ;*****
34 ; USER DEFINED REGISTERS
35 ;*****
36 ;
37 RSEG   AT          1AH
38 ;
39 TRANSMIT:   DSB          17      ;transfer data area
40 T_COUNT:    DSW          1       ;transfer data index counter
41 TXDDONE:    DSB          1       ;transfer done flag
42 ;

```

4.4 Program Example (Continued)

```

43 ;*****
44 ; INTERRUPT VECTOR TABLE
45 ;*****
46 ;
47 CSEG AT 2004H ;capture/compare module 0 interrupt
;location
48
49 DCW CAPCOMP0_INT ;INT 02
50 ;
51 ;*****
52 ; CHIP CONFIGURATION BYTES
53 ;*****
54 ;
55 CSEG AT 2018H
56
57 DCW 20CFH ;CCB
58 DCW 20DCH ;CCB1
59 ;
60 ;*****
61 ; PTS VECTOR TABLE
62 ;*****
63 ;
64 CSEG AT 2044H ;capture/compare module 0 PTSCB
;location
65 DCW 100H ;PTSCB vector address
66 ;
67 ;*****
68 ; MAIN ROUTINE
69 ;*****
70 ;
71 CSEG AT 2080H
72
73 MAIN_START:
74 DI ;disable interrupt
75 LD SP,#0200H ;set-up stack pointer
76 ;
77 ;Clear interrupt mask register
78 ;
79 CLRB INT_MASK ;reset interrupt mask register
80 CLRB INT_MASK1
81 LDB WSR,#3EH ;map 64 bytes to 1F80H-1FBFH
82 CLRB PI_MASK_WO ;reset peripheral interrupt mask reg.
83 ;
84 ;Initialize port & timer
85 ;
86 LDB WSR,#3FH ;map 64 bytes to 1FC0H-1FFFH
87 LDB P2_MODE_WO,RO ;P2.0-P2.7=LSIO
88 LDB P2_DIR_WO,#00H ;P2.0-P2.7=OUTPUT
89 LDB P2_REG_WO,#0FFH ;P2.0-P2.7=high
90 LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
91 LDB T1CONTROL_WO,#0COH ;timer 1 enable, up count, clock
92 ;internal, pre-scale=div 1
93 ;
    
```

4.4 Program Example (Continued)

```

94 ;Initialize TXD Mode
95 ;
96     LDB     WSR,#24H           ;map 64 bytes to 100H-13FH
97     LDB     PTSCONO_WO,#60H   ;SIO transfer
98     LD      EPAREGO_WO,#1F42H ;EPA capture/compare0 register address
99     LD      BAUDCONSTO_WO,#01A0H ;set baud rate 9600 @ 16 MHz
100    LD      PTSVEC10_WO,#108H  ;pointer to PTSCB1
101    LD      PORTREGO_WO,#1FD4H ;Port 2 contains TXD pin
102    LDB     PORTMASKO_WO,#01H  ;P2.0=TXD pin
103    ORB     INT_MASK,#04H      ;enable interrupt on capture/compare
104                                     ;module 0
105 ;
106 ;Set transfer data
107 ;
108     LD      T_COUNT,#16        ;set transfer data count
109     CLRFB   TXDDONE            ;clear transfer done flag
110     LDB     PTSCOUNT0_WO,#0AH  ;# of bits(including parity/stop bits
111                                     ;and excluding start bit)
112     LDB     PTSCON10_WO,#21H   ;PTSCON1 - odd parity
113     LDB     DATA0_WO_L,TRANSMIT[T_COUNT]
114                                     ;set transfer data
115     ORB     PTSSEL,#04H        ;enable PTS on capture/compare module 0
116     LDB     WSR,#3FH           ;map 64 bytes to 1FCOH-1FFFH
117     ANDB    P2_REG_WO,#0FEH    ;clear P2.0 (start bit)
118     LDB     WSR,#3DH           ;map 64 bytes to 1F40H-1F7FH
119     LDB     CAPCOMPO_CON_WO,#40H ;Compare - interrupt only
120     ADD     CAPCOMPO_TIME_WO,TIMER1_WO,#01A0H
121                                     ;start bit width
122     EPTS                    ;enable PTS
123     EI                        ;enable interrupt
124 ;
125 ;*****
126 ;Polling routine
127 ;*****
128 ;
129 LOOP:
130     JBC     TXDDONE,0,LOOP      ;check transfer done flag
131     NOP                    ;dummy command
132     NOP                    ;dummy command
133 ; -
134 ; -
135 ;

```


4.4 Program Example (Continued)

```

136 ;*****
137 ;end of PTS interrupt service routine
138 ;*****
139 ;
140 CAPCOMPO_INT: ;end-of-PTS interrupt routine
141     PUSHA ;save PSW,INT_MASK,INT_MASK1,WSR
142 CAPCOMPO_LOOP:
143     JBC INT_PEND,2,CAPCOMPO_LOOP ;wait interrupt for last PTS cycle
144     LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
145     LDB CAPCOMPO_CON_WO,#00H ;disable capture/compare module0
146     ANDB INT_PEND,#0FBH ;clear false pending interrupt
147     DJNZW T_COUNT,CAPCOMPO_SKIP ;check end of transfer
148     LDB TXDDONE,#01H ;set done flag
149     BR CAPCOMPO_RET ;finish transfer
150 CAPCOMPO_SKIP:
151     LDB WSR,#24H ;map 64 bytes to 100H-13FH
152     LDB DATA0_WO_L,TRANSMIT[T_COUNT]
153 ;set next transfer data
154     LDB PTSCOUNT0_WO,#0AH ;# of bits(including parity/stop
155 ;bits and excluding start bit)
156     LDB PTSCON10_WO,#21H ;PTSCON1 - odd parity
157     LDB WSR,#3FH ;map 64 bytes to 1FC0H-1FFFH
158     ORB P2_REG_WO,#01H ;set P2.0 (prepare start bit)
159     ANDB P2_REG_WO,#0FEH ;clear P2.0 (start bit)
160     ORB PTSEL,#04H ;enable PTS on capture/compare
module 0
161     LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
162     LDB CAPCOMPO_CON_WO,#40H ;Compare - interrupt only
163     ADD CAPCOMPO_TIME_WO,TIMER1_WO,#01A0H
164 ;start bit width
165 CAPCOMPO_RET:
166     POPA ;load PSW,INT_MASK,INT_MASK1,WSR
167     RET ;return to main loop
168
169 END

```

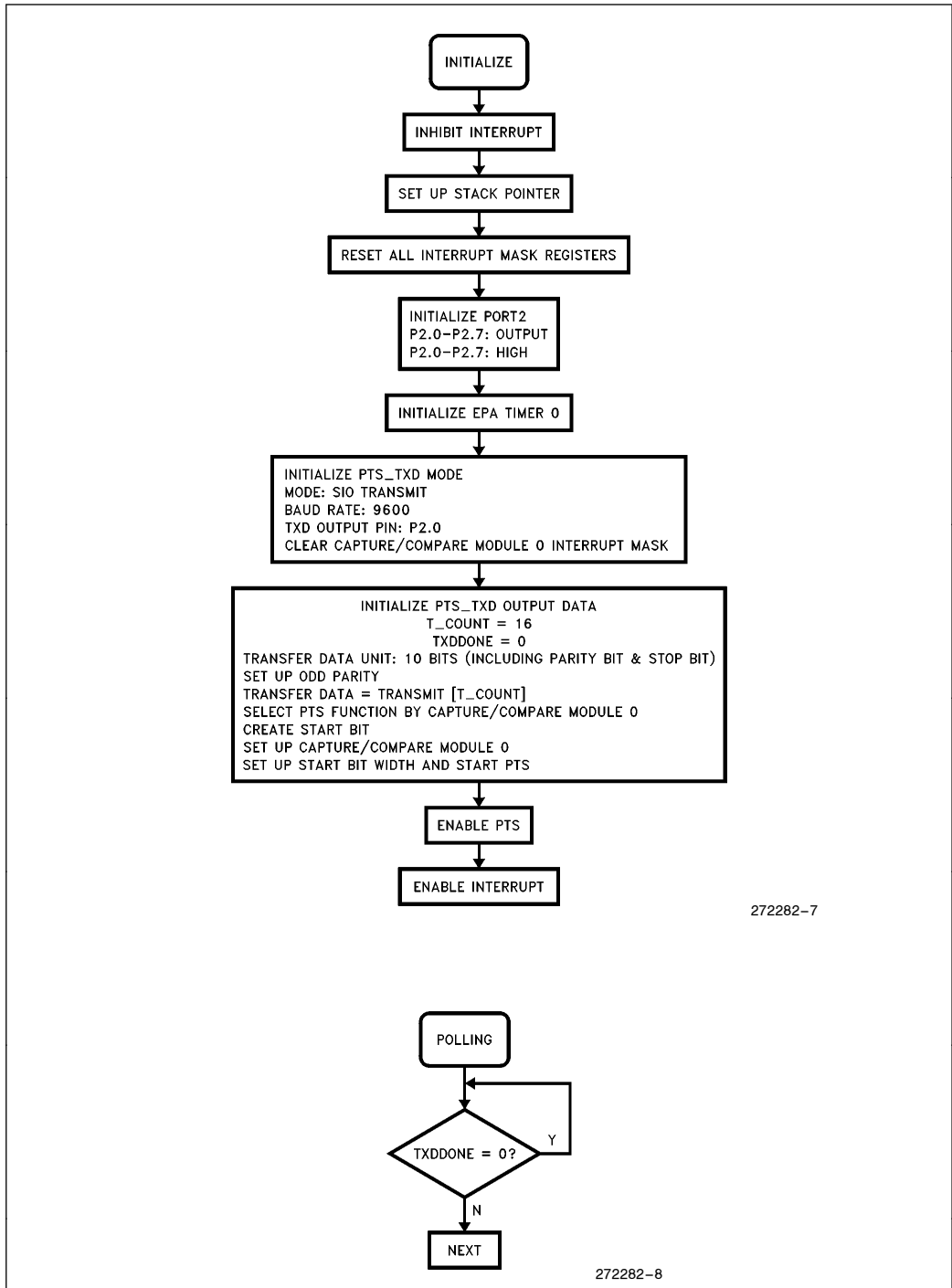


Figure 4-1. Flow Chart—ASIO Transmit Initialization

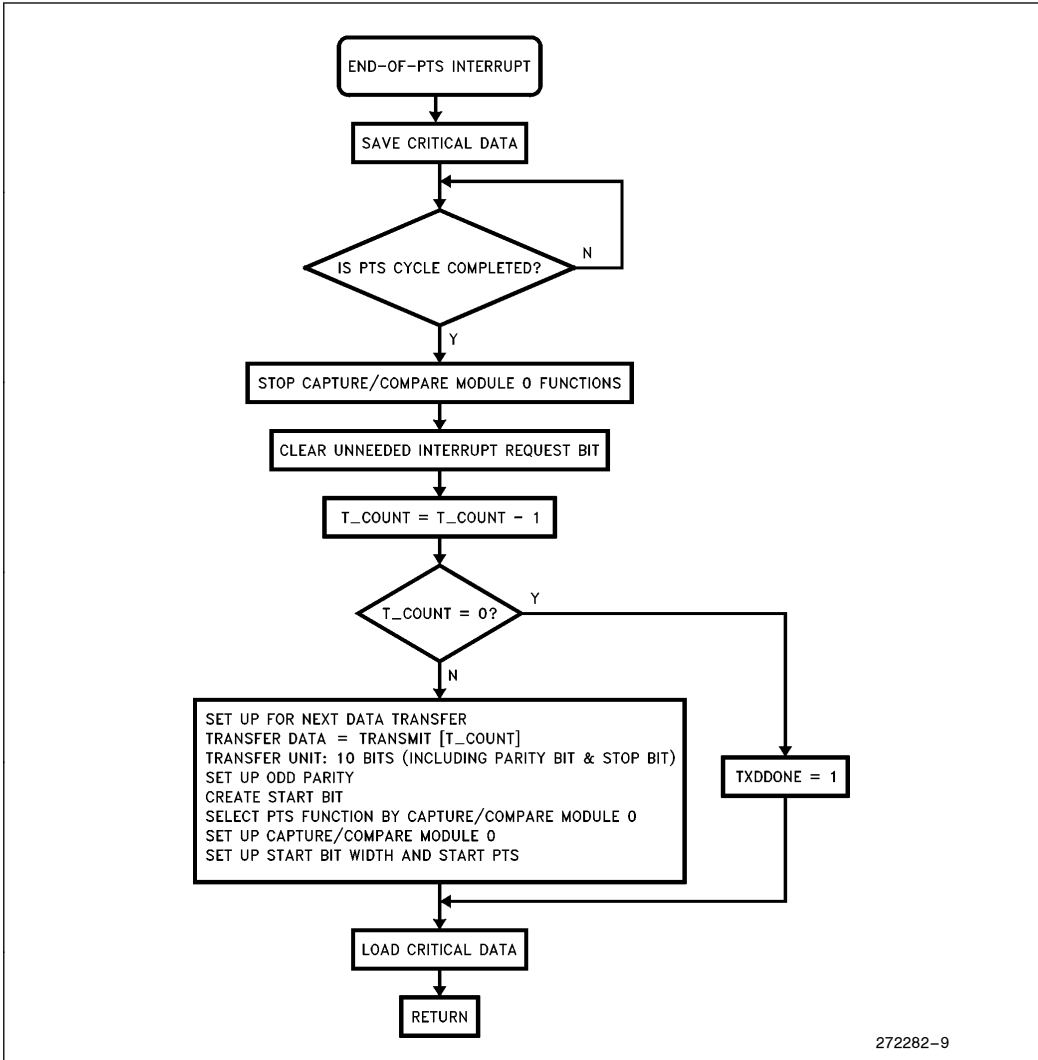


Figure 4-2. Flow Chart—ASIO Transmit Interrupt Routine

272282-9

5.0 ASYNCHRONOUS SERIAL DATA RECEPTION PROGRAM SAMPLE

5.1 Introduction

The asynchronous serial data receive function on the 8XC196MC/MD is performed in software using the PTS SIO mode with an EPA channel. In this program example the EPA CAPCOMP1 channel is initially used in the capture falling edge mode to receive the data “start” bit input on P2.1. This generates a conventional interrupt (the same as the “end of PTS” interrupt) which starts the asynchronous receive process.

This initial interrupt changes the CAPCOMP1 module to the compare mode, and sets the time of the next compare to 1.5 bit times and enables the PTS. Thus, at exactly 1.5 bit times from the beginning of the start bit, the first PTS cycle will sample the input data on P2.1 and shift it into the DATA1_W0 register.

If majority sample mode is selected (as in this example), an additional sample is taken. If the two samples are different, the data is sampled one more time to determine which polarity is correct. The time between samples is controlled by the value of the SAMPTIME register in the PTSCB. Note that for each additional sample taken, the execution time for the PTS cycle increases substantially. This is described in the 8XC196MC user manual section 12.3.5.

Each PTS cycle samples the input data on P2.1 and shifts the value into the DATA1_W0 register. The time interval between cycles establishes the baud rate. To receive 8 data bits with parity, a total of 10 PTS cycles and two conventional (end-of-PTS interrupt) interrupts occur.

5.1.1 END-OF-PTS INTERRUPT

The end-of-PTS interrupt services both the initial interrupt (as mentioned earlier) and the final interrupt (when PTSCOUNT = 0). This interrupt takes the conventional interrupt vector to CAPCOMP1_INT, where it is determined if this is the initial or final interrupt. This is done by reading CAPCOMP1_CON to see if the module is in the capture or compare mode. If CAPCOMP1_CON.6 = 0, the capture mode is active indicating that this is the initial “start” bit interrupt. In this case, the CAPCOMP1 module is switched to the compare mode, and CAPCOMP1_TIME is loaded with the time to sample the first data bit (1.5 bit times). The PTS is enabled, and the routine returns to a loop waiting for the rest of the data bits to be received.

If CAPCOMP1_CON.6 = 1 the compare mode is selected and this is the final interrupt. In this example, PTSCON11 is checked to see if any errors occurred, and DATA1_W0_H (which contains the incoming character) is stored in the RECEIVE buffer. PTSCOUNT1 is reloaded with 10, PTSCON11 is reloaded (this is required), and the input buffer DATA1_W0 is cleared. The EPA CAPCOMP1_CON channel is set to the capture falling edge mode, thus readying P2.1 to wait for the next start bit. A total of 16 bytes are received.

5.2 Detailed Program Description

5.2.1 CONSTANT DECLARATIONS (Lines 1–42)

This section of the code defines the location of the PTS control block (PTSCB) registers, first as accessed through 64-byte window 4 (WSR = 24H) and then in their absolute locations (0110H–011EH). By accessing the control block through the window, loading and servicing of the PTSCB is much faster, and requires less code space. Note that all PTSCB’s must be located on a quad-word boundary (divisible by 8).

Lines 37–41 define data storage for a receive character buffer, a character counter, and a flag register.

5.2.2 INTERRUPT VECTORS AND CCB (Lines 43–66)

Lines 47–49 fill the interrupt vector table location 2008H with the address of the conventional CAPCOMP1 interrupt service routine. This routine is known as the “end-of-PTS” interrupt routine.

Lines 55–58 define the chip configuration bytes, CCB and CCBI. These need to be configured for the particular system that this program is run on.

Lines 64–65 fill the PTS interrupt vector table location 2048H (EPA CAPCOMP1 PTS interrupt) with the base address of the PTSCB.

5.2.3 MAIN PROGRAM (Lines 67–135)

Lines 71–82 define the program starting location (2080H), set up the stack at 0200H, and disable and clear out all pending interrupts.

Lines 84–93 initialize I/O port 2 (used for the RXD input) and set up timer1 for the time base. This is done through the windowed address of the special Function Registers (SFRs) . In line 90, the unused port pins are initialized to 1. Line 92 initializes the timer1 module.

Lines 97–105 initialize most of the PTSCB. Line 106 enables the interrupt mask bit for the CAPCOMP1 module. Line 111 defines the number of bytes to receive and line 112 clears the transfer done flag, indicating that the receive operation is not yet complete.

Lines 113–116 set up the SIO asynchronous receive mode parameters, establishing the number of bits per word and the parity.

Line 116 clears the receive register DATA1__W0 in the PTSCB. At this point, the entire PTSCB has been initialized, and is ready for action!

Line 118 sets up the CAPCOMP1 module in the capture falling edge mode by loading 10H to the CAPCOMP1__CON__W0 register.

Lines 119–120 enable the PTS and conventional interrupts.

It is important to note that the PTSSEL bit for CAPCOMP1 has not yet been set. This is because the first interrupt must be processed by a conventional ISR.

Lines 126–129 are a “do nothing” loop that monitors the RXDDONE flag and waits for the transmit operation to complete sending 16 bytes. When the operation is complete, the NOPs are executed, and the user needs to place the next segment of code here.

5.2.4 END-OF-PTS INTERRUPT ROUTINE (Lines 133–176)

As mentioned earlier, the end-of-PTS interrupt services both the initial interrupt (“start” bit) and the final interrupt (when PTSCOUNT = 0).

Line 138 PUSHes the CPU status and interrupt masks onto the stack. Line 140 checks the value of the CAPCOMP1__CON register to determine if this is the “start” bit interrupt or the “end-of-PTS” interrupt. If CAPCOMP1__CON.6 = 0 the program flow is switched to RXD__SETUP, line 153, to process the “start” bit.

If CAPCOMP1__CON.6 = 1 the program flow continues and processes the “end-of-PTS” interrupt as follows:

Line 142 disables CAPCOMP1 and clears any pending interrupt it may have generated.

Lines 145–147 check for PTSCON11__W0 register to determine if a framing or parity error occurred. If there is an error, the program jumps to the RXD__ERROR routine at line 162, sets an error code in the RXDDONE flag register, and returns without performing any further housekeeping activity. This is just a “hook” for user written code. Note that if this occurs, additional software needs to reinitialize the receive routine as required by the system design.

Lines 147–152 store the received data in the RECEIVE buffer and check if 16 bytes have been received yet. If yes, the program returns; otherwise, execution continues at CAPCONP1__SKIP (line 165).

On lines 162–171 the CAPCOMP1__SKIP routine “refreshes” the PTSCB registers that have changed. PTSCOUNT1__W0 and PTSCON11__W0 are reloaded, and DATA1__W0 is cleared. The CAPCOMP1 module is returned to the capture falling edge mode to wait for another “start” bit. Note that the PTSSEL bit for CAPCOMP1 in NOT set at this time, as the next interrupt needs to be a conventional ISR.

Finally, lines 172–175 return program execution to the main program flow.

5.3 Top 5 Issues of Using the PTS/SIO Receive Mode

1. The CAPCOMP module is used in the capture mode to trap the start bit, and the compare mode to receive the data bits.
2. The capture of the “start” bit must cause a conventional interrupt, not a PTS cycle.
3. The end-of-PTS routine must determine if the “start” bit interrupt or the “real” end-of-PTS interrupt has caused the routine to be entered. This is done by reading the CAPCOMP__CON register to determine whether the capture or compare mode is active.

4. The end-of-PTS interrupt routine must disable the CAPCOMP module and clear the corresponding interrupt pending bit (see lines 142–143).
5. The time loaded into CAPCOMP_TIME after the “start” bit interrupt must equal 1.5 bit times so that each following bit is sampled in the center of its period.

5.4 Program Example

```

1  ;*****
2  ; PTSCB address in vertical window 24H
3  ;*****
4  ;
5  PTSCOUNT1_WO EQU      ODOH:BYTE
6  PTSCON1_WO   EQU      OD1H:BYTE
7  EPAREG1_WO   EQU      OD2H:WORD
8  BAUDCONST1_WO EQU     OD4H:WORD
9  PTSVEC11_WO  EQU      OD6H:WORD
10 PORTREG1_WO  EQU      OD8H:WORD
11 PORTMASK1_WO EQU      ODAH:BYTE
12 PTSCON11_WO  EQU      ODBH:BYTE
13 DATA1_WO    EQU      ODCH:WORD
14 SAMPTIME1_WO EQU      ODEH:WORD
15 ;
16 DATA1_WO_H  EQU      ODDH:BYTE
17 ;
18 ;*****
19 ; absolute address of PTSCB
20 ;*****
21 ;
22 PTSCOUNT1    EQU      110H:BYTE
23 PTSCON1      EQU      111H:BYTE
24 EPAREG1      EQU      112H:WORD
25 BAUDCONST1   EQU      114H:WORD
26 PTSVEC11     EQU      116H:WORD
27 PORTREG1     EQU      118H:WORD
28 PORTMASK1    EQU      11AH:BYTE
29 PTSCON11     EQU      11BH:BYTE
30 DATA1       EQU      11CH:WORD
31 SAMPTIME1    EQU      11EH:WORD
32 ;
33 ;*****
34 ; USER DEFINED REGISTERS
35 ;*****
36 ;
37 RSEG          AT          LAH
38 ;
39 RECEIVE:      DSB          17      ;receive data buffer
40 R_COUNT:      DSW          1       ;receive data index counter
41 RXDDONE:      DSB          1       ;transfer done flag
42 ;

```

5.4 Program Example (Continued)

```

43 ;*****
44 ; INTERRUPT VECTOR TABLE
45 ;*****
46 ;
47 CSEG AT 2008H ;capture/compare module 1 INT. location
48
49 DCW CAPCOMPL_INT ;INT 04
50 ;
51 ;*****
52 ; CHIP CONFIGURATION BYTES
53 ;*****
54 ;
55 CSEG AT 2018H
56
57 DCW 20CFH ;CCB
58 DCW 20DCH ;CCB1
59 ;
60 ;*****
61 ; PTS VECTOR TABLE
62 ;*****
63 ;
64 CSEG AT 2048H ;capture/compare module 1 PTSCB
        location
65 DCW 110H ;PTSCB vector address
66 ;
67 ;*****
68 ; MAIN ROUTINE
69 ;*****
70 ;
71 CSEG AT 2080H
72
73 MAIN_START:
74 DI ;disable interrupt
75 LD SP,#0200H ;set-up stack pointer
76 ;
77 ;Clear interrupt mask register
78 ;
79 CLRB INT_MASK ;reset interrupt mask register
80 CLRB INT_MASK1
81 LDB WSR,#3EH ;map 64 bytes to 1F80H-1FBFH
82 CLRB PI_MASK_WO ;reset peripheral interrupt mask reg.
83 ;
84 ;Initialize port & timer
85 ;
86 LDB WSR,#3FH ;map 64 bytes to 1FC0H-1FFFH
87 LDB P2_MODE_WO,RO ;P2.0-P2.7=LSIO
88 LDB P2_DIR_WO,#02H ;OUTPUTP=2.0,P2.2-P2.7,
89 ;INPUT=P2.1
90 LDB P2_REG_WO,#0FFH ;P2.0-P2.7=high
91 LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
92 LDB T1CONTROL_WO,#0COH ;timer 1 enable, up count, clock
93 ;internal, pre-scale=div 1
94 ;
    
```

5.4 Program Example (Continued)

```

95 ;Initialize RXD Mode
96 ;
97     LDB     WSR,#24H           ;map 64 bytes to 0100H-013FH
98     LDB     PTSCON1_WO,#21H   ;SIO receive
99     LDB     SAMPTIME1_WO,#01H ;sample time for majority sample
100    LD      EPAREG1_WO,#1F46H ;CAPCOMP1_TIME Address
101    ;
102    LD      BAUDCONST1_WO,#01A0H ;set baud rate 9600 baud @ 16 MHz
103    LD      PTSVEC11_WO,#118H  ;pointer to PTSCB1
104    LD      PORTREG1_WO,#1FD6H ;Port 2 has RXD pin
105    LDB     PORTMASK1_WO,#02H  ;P2.1 = RXD
106    ORB     INT_MASK,#10H     ;enable interrupt on capture/
107    ;                           ;compare module 1
108    ;
109    ;Set receive mode
110    ;
111    LD      R_COUNT,#16        ;set receive data count
112    CLRB    RXDDONE           ;clear done flag
113    LDB     PTSCOUNT1_WO,#0AH  ;# of bits(including parity/stop bits
114    ;                           ;and excluding start bit)
115    LDB     PTSCON1_WO,#60H    ;PTSCON1 - odd parity
116    CLR     DATA1_WO         ;clear receive data buffer
117    LDB     WSR,#3DH          ;map 64 bytes to 1F40H-1F7FH
118    LDB     CAPCOMP1_CON_WO,#10H ;capture - negative edge
119    EPTS    ;enable PTS
120    EI      ;enable interrupt
121    ;
122    ;*****
123    ; POLLING ROUTINE
124    ;*****
125    ;
126 LOOP:
127     JBC     RXDDONE,0,LOOP     ;check receive done/error flag
128     NOP
129     NOP
130    ;      -
131    ;      -
132    ;
133    ;*****
134 ;end-of-PTS interrupt service routine
135 ;*****

```


5.4 Program Example (Continued)

```

136 ;
137 CAPCOMPL_INT:
138     PUSHA                                ;save PSW,INT_MASK,INT_MASK1,WSR
139     LDB     WSR,#3DH                     ;map 64bytes to 1F40H-1F7FH
140     JBC     CAPCOMPL_CON_W0,6,RXD_SETUP
141                                           ;start bit or end-of-PTS
                                           ;interrupt?
142     LDB     CAPCOMPL_CON_W0,#00H         ;disable capture/compare module 1
143     ANDB   INT_PEND,#0EFH                ;clear false pending interrupt
144     LDB     WSR,#24H                     ;map 64 bytes to 0100H-013FH
145     JBS     PTSCON11_W0,1,RXD_ERROR      ;framing error?
146     JBS     PTSCON11_W0,6,RXD_ERROR      ;parity error?
147     STB     DATA1_W0_H,RECEIVE[R_COUNT]
148                                           ;save received data
149     DJNZW   R_COUNT,CAPCOMPL_SKIP        ;decrement data counter
150                                           ;& check end of receive
151     LDB     RXDDONE,#01H                 ;set done flag
152     SJMP    CAPCOMPL_RET                 ;finish data receive
153 RXD_SETUP:
154     LDB     WSR,#3FH                     ;map 64 bytes to 1FC0H-1FFFH
155     JBS     P2PIN_W0,1,CAPCOMPL_RET      ;detect false start bit
156     LDB     WSR,#3DH                     ;map 64 bytes to 1F40H-1F7FH
157     LDB     CAPCOMPL_CON_W0,#40H        ;Compare - interrupt only
158     ADD     CAPCOMPL_TIME_W0,#0270H     ;set first PTS cycle,
                                           ;CAPCOMPL_TIME=
                                           ;CAPCOMPL_TIME+(1.5*BAUDCONST)
159     ORB     PTSSEL,#10H                 ;enable PTS on capture/compare
160                                           ;module 1
161     SJMP    CAPCOMPL_RET                 ;exit
162 RXD_ERROR:
163     LDB     RXDDONE,#03H                 ;set error code
164     SJMP    CAPCOMPL_RET                 ;exit
165 CAPCOMPL_SKIP:
166     LDB     PTSCOUNT1_W0,#0AH           ;# of bits(including parity/
                                           ;stop bits
167                                           ;and excluding start bit)
168     LDB     PTSCON11_W0,#60H            ;PTSCON1 - odd parity
169     CLR     DATA1_W0                    ;clear receive data buffer
170     LDB     WSR,#3DH                     ;map 64 bytes to 1F40H-1F7FH
171     LDB     CAPCOMPL_CON_W0,#10H        ;capture - negative edge
172 CAPCOMPL_RET:
173     POPA                                     ;load PSW,INT_MASK,
174                                           ;INT_MASK1,WSR
175     RET                                     ;return to main loop
176 END

```

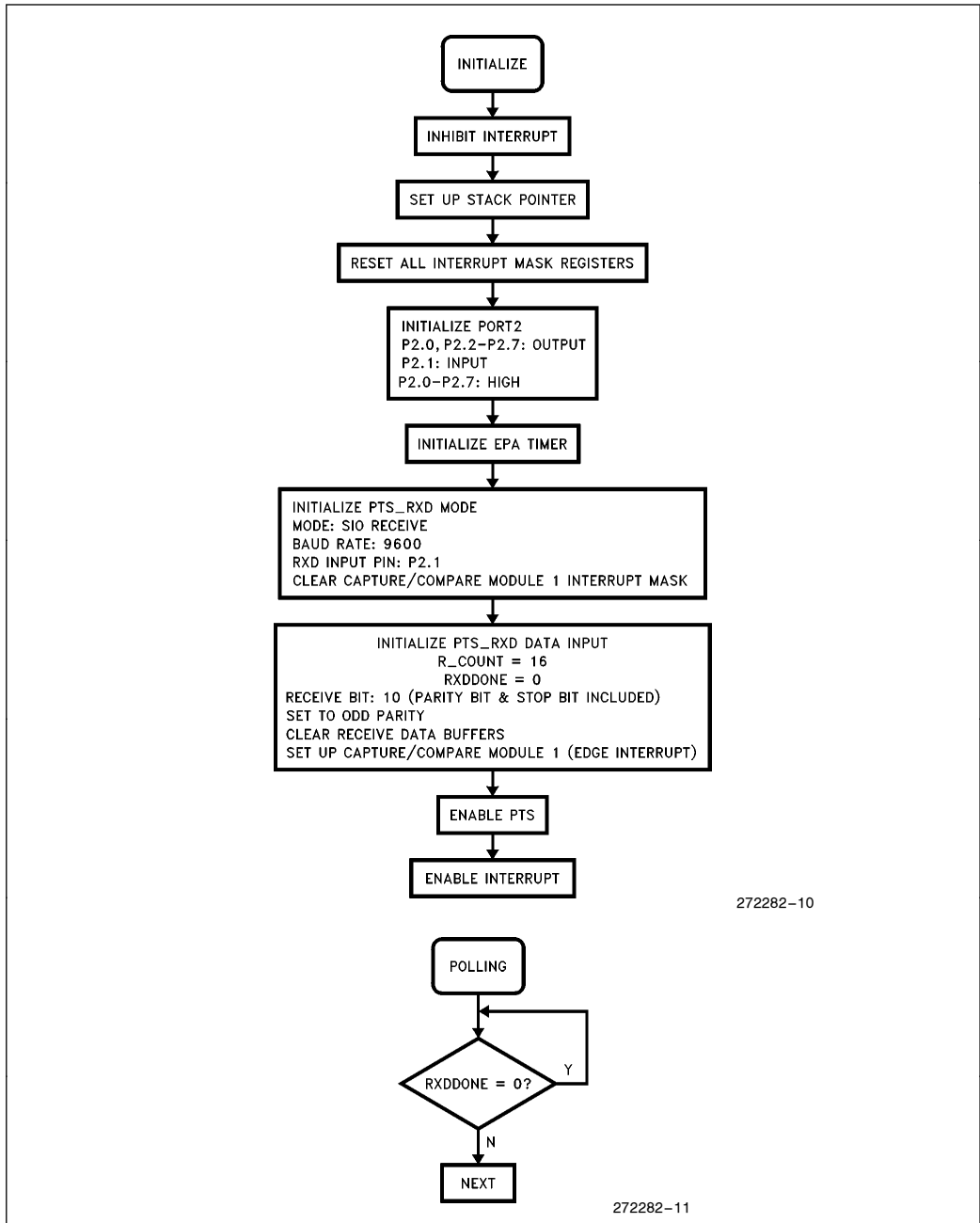


Figure 5-1. Flow Chart—ASIO Receive Initialization

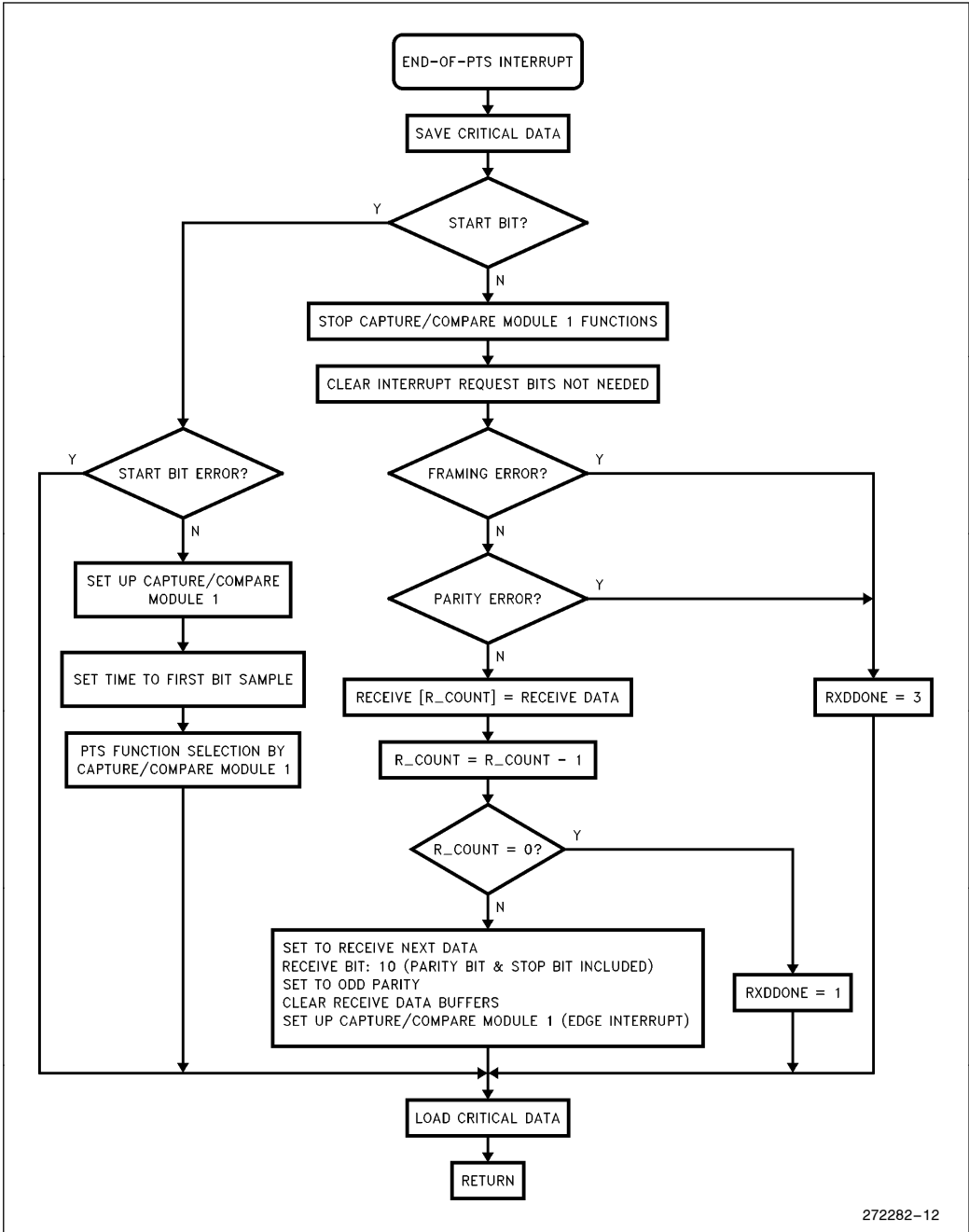


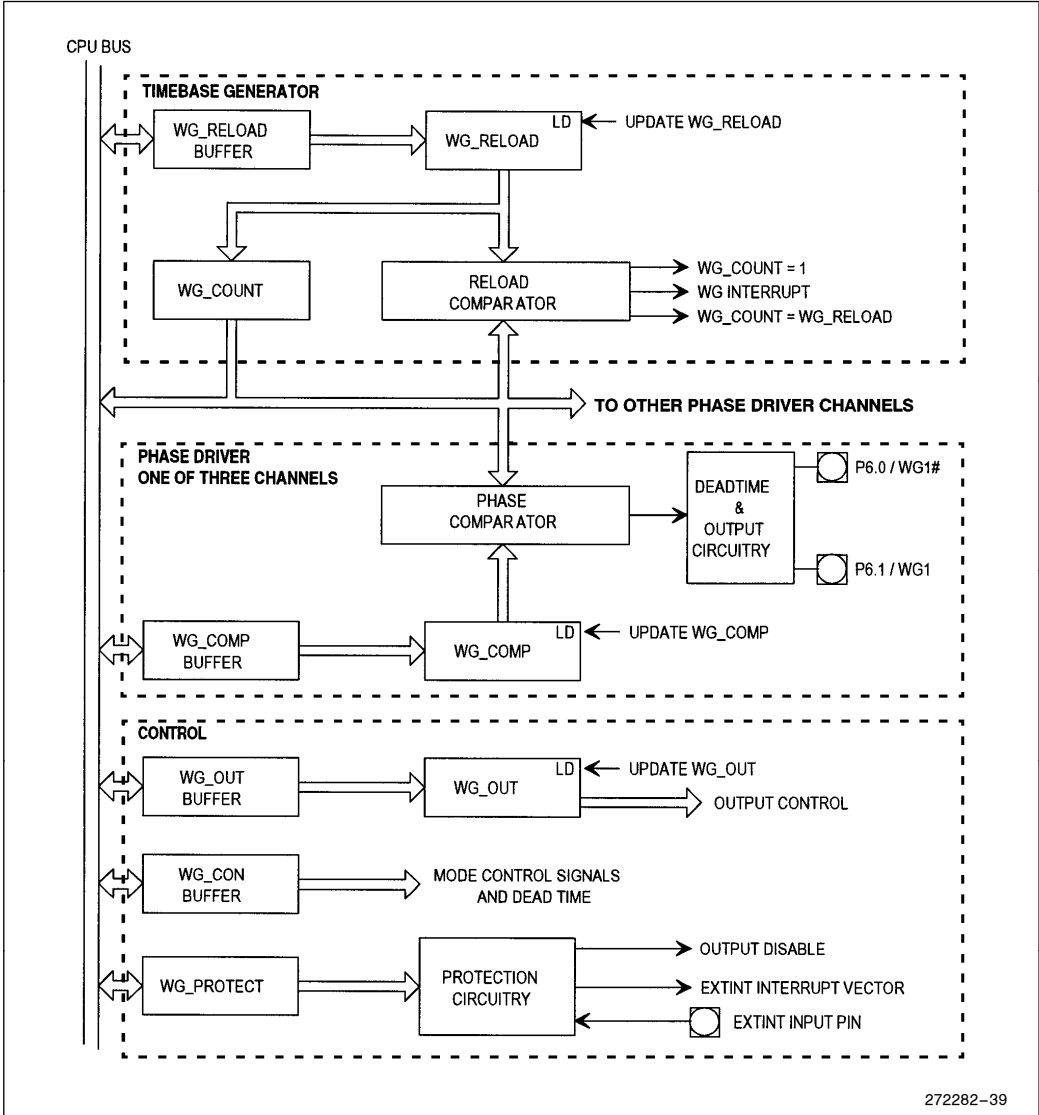
Figure 5-2. Flow Chart—ASIO Receive Interrupt Routine

272282-12

6.0 UNDERSTANDING THE WAVEFORM GENERATOR

6.1 Introduction

One of the unique features of the 8XC196MC/MD is the on-chip waveform generator (WFG). This peripheral greatly simplifies the control software and external hardware used for generating synchronized PWM waveforms. The WFG has three synchronized PWM modules, each with a phase compare register, a dead-time generator and two programmable complimentary outputs. The WFG allows generation of 3 independent complimentary pairs of PWM's. However, the PWM's share a common carrier frequency, dead-time, and mode of operation. Once initialized, the WFG requires CPU intervention only to change PWM duty cycles. A block diagram of the WFG is shown in Figure 6-1.



272282-39

Figure 6-1. Simplified WFG Block Diagram

The WFG is divided into 3 functional areas: the timebase generator, the phase driver channel, and the control section.

The timebase generator establishes the carrier period for the PWMs. This period is determined by the value in the WG_RELOAD register, along with the mode of operation. The timebase generator operates in 4 different modes, allowing either centered or edge aligned PWM generation.

The center aligned PWM modes are desirable when driving 3-phase AC induction motors, as it results in minimum harmonic content of the waveforms. This in turn results in more efficient operation of the power control circuitry and the motor. Additionally, the carrier frequency is effectively doubled compared to edge aligned modes. Center aligned mode operation is described in Section 6.5.

The edge aligned modes are used to generate conventional PWMs. The WFG adds its programmable carrier frequency/resolution, dead-time generation and complimentary output features, reducing external hardware requirements. Edge aligned mode operation is described in Section 6.6.

The phase driver channels determine the duty cycle of the PWMs. There are 3 independent phase driver channels, each with 2 programmable outputs. Each phase driver contains a programmable dead-time generator, which prevents the complimentary outputs from being asserted at the same time. The output circuitry allows using the pins as I/O or PWM, and allows selecting either asserted-high or asserted low output levels. An output disable feature is provided which forces the outputs to their deasserted value in response to an external or software event.

The control section contains registers which determine the modes of operation and other configuration information. A programmable protection circuit monitors the EXTINT input pin, and if a valid event is detected generates an interrupt and disables the WFG outputs.

In the following discussions, output signals are described as being *asserted* or *deasserted*. When asserted, an active-high output will be in the high (or “1”) state, and when deasserted will be low (or “0”). Conversely, when asserted, an active-low output will be low (“0”), and when deasserted will be high (“1”).

6.2 Buffering

Figure 6-1 shows how the counters and registers are buffered. Synchronization of register loading is determined by the mode of operation. Synchronization is necessary to prevent erroneous or non-symmetrical duty cycles from occurring in the middle of a pulse train. Table 6-1 shows the conditions for updating the registers in the various modes.

When the counter WG_COUNT is stopped (by clearing the EC bit in WG_CON or if WG_COUNT = 0), the WG_COMP registers are all loaded 1/2 state time after their respective buffers are written. This allows for initialization of the WFG.

Note that in mode 3, WG_RELOAD can be synchronized with an EPA event. This is a special EPA mode which is enabled by setting the Peripheral Function Enable (PFE) bit in either the CAPCOMP0 or CAPCOMP2 EPA channels.

Table 6-1. Conditions for Register Updates

Mode	PWM Type	Counter Operation	Update WG_RELOAD	Update WG_COUNT	Update WG_COMP	Update WG_OUT*
0	Centered	Up/Down	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD
1	Centered	Up/Down	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD and WG_COUNT = 1	WG_COUNT = WG_RELOAD and WG_COUNT = 1
2	Edge	Up	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD	WG_COUNT = WG_RELOAD
3	Edge	Up	WG_COUNT = WG_RELOAD and EPA event	WG_COUNT = WG_RELOAD and EPA event	WG_COUNT = WG_RELOAD and EPA Event	EPA Event

* If SYNC bit = 1. Otherwise, changes take effect immediately.

The SYNC bit in the WG_OUT register (Figure 6-10) controls loading of the WG_OUT register. The SYNC bit is not buffered, so changes made to it take effect immediately. When SYNC = 1, changes to the WFG output pins are synchronized according to Table 6-1. When SYNC = 0, the output pins are updated immediately. Note that WG_OUT should be initialized with SYNC = 0 to insure that the pins will be in the desired state when the WFG is started.

When the WG_RELOAD, WG_COMP, and WG_OUT registers are read, the returned data is the value in the buffer. The registers themselves cannot be accessed by the CPU.

6.3 The Timebase Generator

The timebase counter, WG_COUNT, is a multi-mode 16-bit counter which is clocked every T_{Osc} (½ F_{XTAL}). This counter may be read by the CPU to determine its value at any time. The counter is controlled by the WG_CON register, Figure 6-4. WG_CON configures the mode of WFG operation using the mode bits (M0 and M1), enables and disables the counter (EC bit), and returns the Counter Status (CS bit). If 0000H is loaded into WG_COUNT (via WG_RELOAD), the counter will stop. When the counter is running, it continuously counts up and/or down (mode dependent) between 0001H and the WG_RELOAD value. Refer to Sections 6.5 and 6.6 for a more complete description of counting modes.

6.3.1 CENTER ALIGNED MODE WG_COUNT OPERATION (MODES 0 AND 1)

When the 8XC196MC is first powered up and during reset, WG_COUNT is reset to 0000H. When the WG_RELOAD buffer is written with the counter stopped, the value is immediately transferred to WG_COUNT. When the counter is started, it immediately begins counting down to 1. When the count = 1, the counter waits one state time, and begins counting back up. When the count reaches the WG_RELOAD value, it is updated from the WG_RELOAD register, and begins the down count again. This process produces a symmetrical ascending and descending count whose period is equal to 2 × WG_RELOAD. This can be visualized as a triangular wave, and is illustrated in Figure 6-2.

The WG_RELOAD register establishes the carrier period for the waveforms that will be generated. Formulas for carrier period are given in Section 6.5.4 and 6.6.4.

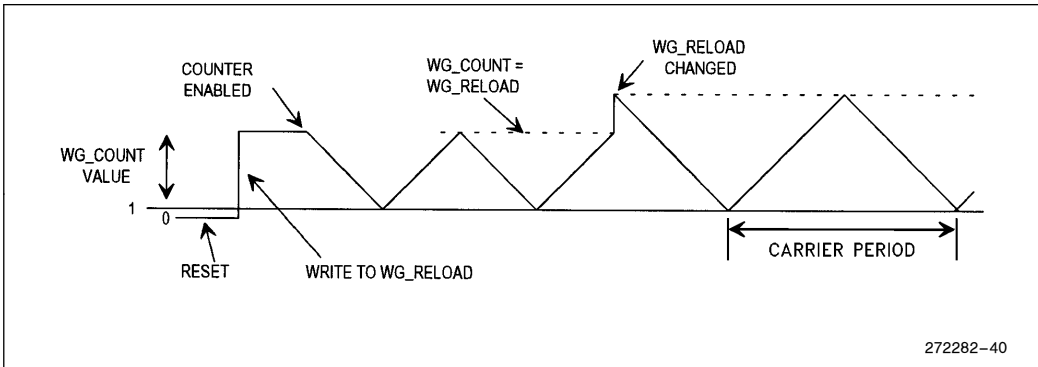


Figure 6-2. Modes 0 and 1 Counter Operation

6.3.2 EDGE ALIGNED MODE WG_COUNT OPERATION (MODES 2 AND 3)

When the 8XC196MC/MD is first powered up and during reset, WG_COUNT is reset to 0000H. When the WG_RELOAD buffer is written with the counter stopped, WG_COUNT is loaded with 0001. When the counter is then started, it counts up until the WG_RELOAD value is reached. At this time, WG_COUNT is reset to 1, and up counting begins again. This process produces a smoothly ascending count whose period equals WG_RELOAD. This can be visualized as a sawtooth wave, illustrated in Figure 6-3. WG_RELOAD is updated when the counter is reset to 1, such that the following cycle will count to the new WG_RELOAD value. In mode 3 only, an EPA event can force an early reset of the counter to 1.

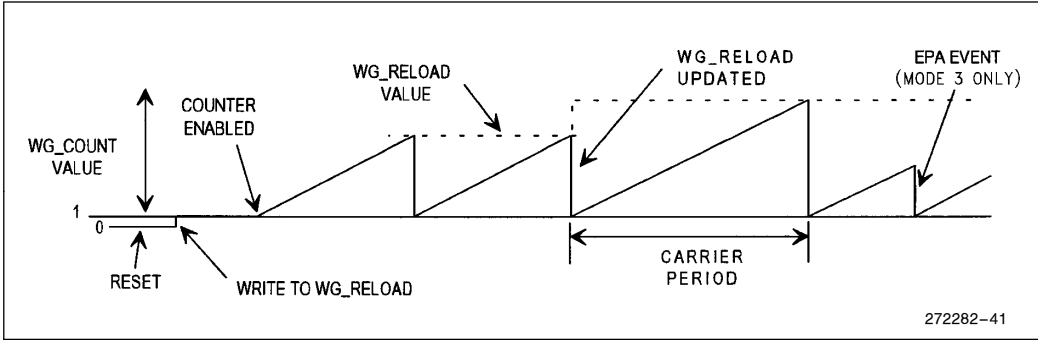


Figure 6-3. Modes 2 and 3 Counter Operation

6.4 The WFG Control Register (WG_CON)

The WFG control register is illustrated in Figure 6-4. The counter mode is selected by writing the mode bits M0 and M1. Counter status is determined by reading the Counter Status (CS) bit. If the counter is currently counting up the CS bit will be set (CS = 1); if down the CS bit will be clear (CS = 0). The counter is enabled by the enable counter (EC) bit, which starts (EC = 1) and stops (EC = 0) counter operation. Additionally, bits 0 to 9 contain the 10-bit dead-time value (DT_VALUE) which determines the output dead-time. See Section 6.8 for further information on dead-time.

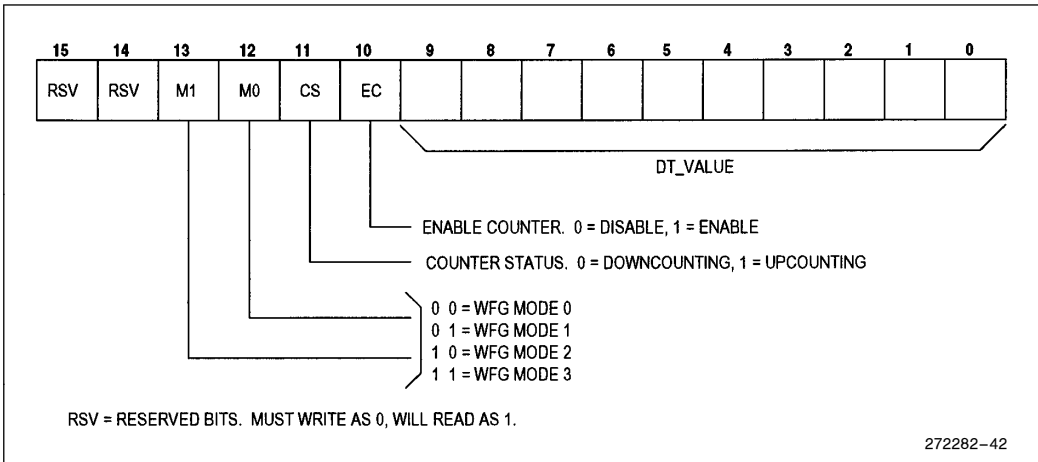


Figure 6-4. WG_CON Register

6.5 Center Aligned PWM Modes (Modes 0 and 1)

6.5.1 POWER-UP INITIALIZATION

When power is first applied to the device and after a reset, all WG registers are reset to 0. The counter is stopped, and all values written to the WFG registers take effect $\frac{1}{2}$ state time later ($\frac{1}{2}$ state time = $1/F_{XTAL}$). In modes 0 and 1, when WG_RELOAD is first written the value will be transferred to WG_COUNT, and down counting starts when the EC bit in WG_CON is set. Figure 6-5 illustrates the PWM outputs generated in modes 0 and 1. Note that when the WG_RELOAD value is changed, both the carrier period and duty cycle change. To change the carrier period without effecting the duty cycle, both must be proportionally changed at the same time immediately after the WG interrupt.

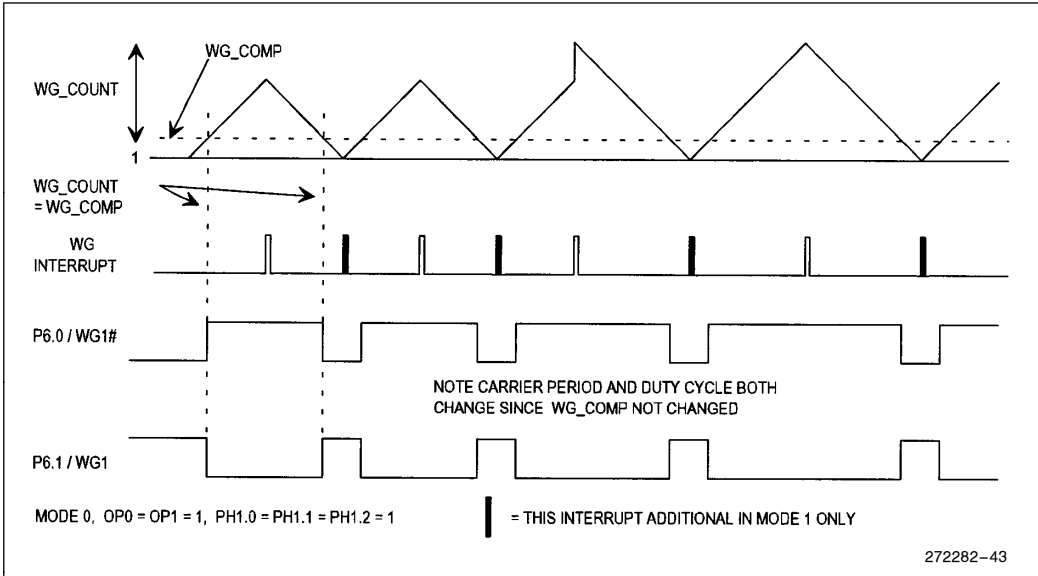


Figure 6-5. WG_COUNT during Center Aligned PWM

6.5.2 WFG OUTPUT OPERATION

When counting up starting from one, the outputs are initially asserted. When WG_COUNT equals WG_COMP the outputs are deasserted. The outputs remain deasserted during up counting and the following down counting until the counter WG_COUNT equals WG_COMP. At this point, the outputs are asserted, the down count continues until 1 is reached, and the process begins over.

This discussion neglects dead-time, discussed in Section 6.8. Note that if WG_RELOAD is changed to a different value, the carrier frequency and duty cycle of the PWM will both change. This is because the outputs remain deasserted for a constant time, while the counter takes longer to cycle.

6.5.3 DIFFERENCES BETWEEN MODE 0 AND MODE 1

The difference between center aligned modes 0 and 1 is in when the WG_COMP and WG_OUT registers are updated (Table 6-1). For mode 0, these registers are updated only when WG_COUNT = WG_RELOAD, at the “peak” of the triangle wave. In mode 1, they are updated twice during the carrier period. First, when WG_COUNT = 1, at the “valley” of the triangle wave and again when WG_COUNT = WG_RELOAD, at the “peak” of the triangle wave.

6.5.4 FORMULAS FOR CARRIER PERIOD AND DUTY CYCLE

The formula for calculating the carrier period in center aligned mode is as follows:

$$T_c = \frac{4 \times \text{WG_RELOAD}}{F_{\text{XTAL}}}$$

where

- WG_RELOAD = 16-bit value
- F_XTAL = Processor clock frequency on XTAL1 pin, MHz
- T_c = carrier period, μs

To calculate the length of time an output is asserted, use the following formula (This neglects dead-time, which is assumed to be minimal):

$$T_{\text{output}} = \frac{4 \times \text{WG_COMP}}{F_{\text{XTAL}}}$$

where

- WG_COMP = 16-bit value, equal to or less than WG_RELOAD
- T_{output} = Total time output is asserted, μs
- F_{XTAL} = Processor clock frequency on XTAL1 pin, MHz

To calculate the duty cycle given WG_RELOAD and WG_COMP values in the center aligned mode, use the following formula (Again, this neglects dead-time):

$$\text{Duty Cycle} = \frac{\text{WG_COMP}}{\text{WG_RELOAD}} \times 100 \text{ percent}$$

6.6 Edge Aligned PWM Modes (Modes 2 and 3)

6.6.1 POWER-UP INITIALIZATION

When power is first applied to the device and after a reset, all WG registers are reset to 0. The counter is stopped, and all values written to the WFG registers take effect 1/2 state time later (1/2 state time = 1/F_{XTAL}). In modes 2 and 3, up counting starts when the EC bit in WG_CON is set. Figure 6-6 illustrates the PWM outputs generated in modes 2 and 3. Note that when the WG_RELOAD value is changed, both the carrier period and duty cycle change. To change the carrier period without effecting the duty cycle, both must be proportionally changed at the same time immediately after the WG interrupt.

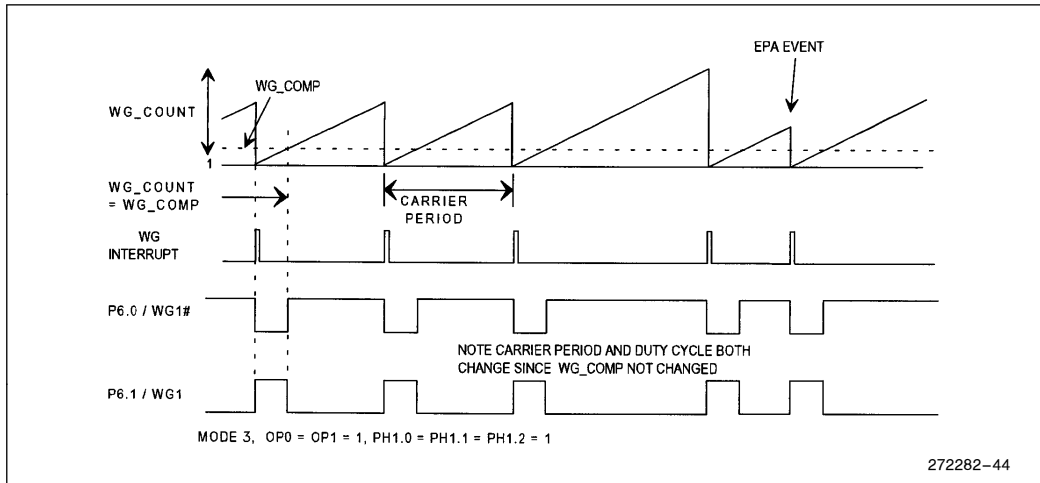


Figure 6-6. WG_COUNT during Edge Aligned PWM

6.6.2 WFG OUTPUT OPERATION

When counting up starting from one, the outputs are initially asserted (see Figure 6-6.). When WG_COUNT equals WG_COMP the outputs are deasserted. The outputs remain deasserted for the rest of the up count. The ramp up continues until WG_COUNT = WG_RELOAD, when the counter is reset to 1 and the outputs are reasserted.

This discussion neglects dead-time, which is discussed in Section 6.7. Note that if WG_RELOAD is changed to a different value, the carrier frequency and duty cycle of the PWM will both change. This is because the outputs remain asserted for a constant time, while the counter takes longer to cycle.

6.6.3 DIFFERENCES BETWEEN MODES 2 AND 3

The difference between edge aligned modes 2 and 3 is when the WG_COMP and WG_OUT registers are updated (Table 6-1). For mode 2, these registers are updated only when WG_COUNT = WG_RELOAD, at the “peak” of the sawtooth wave. In mode 3, they are additionally updated when an EPA “peripheral function” (PFE) event occurs. When the EPA event occurs, the counter will be reset to 1, the new WG_RELOAD value loaded from the buffer, and the up count restarted from 1. The PFE WG reload event can only be generated by the CAPCOMP0 and CAPCOMP2, COMP0 and COMP2 (additionally, in the MD, CAPCOMP3 and COMP3) channels.

6.6.4 FORMULAS FOR CARRIER FREQUENCY AND DUTY CYCLE

The formula for calculating the carrier period in edge aligned mode is as follows:

$$T_c = \frac{2 \times \text{WG_RELOAD}}{F_{\text{XTAL}}}$$

where

- WG_RELOAD = 16-bit value
- F_XTAL = Processor clock frequency on XTAL1 pin, MHz
- T_c = carrier period, μs

To calculate the length of time an output is asserted, use the following formula (this neglects dead-time, which is considered to be minimal):

$$T_{\text{output}} = \frac{2 \times \text{WG_COMP}}{F_{\text{XTAL}}}$$

where

- WG_COMP = 16-bit value, equal to or less than WG_RELOAD
- T_output = Total time output is asserted, μs
- F_XTAL = Processor clock frequency on XTAL1 pin, MHz

To calculate the duty cycle given WG_RELOAD and WG_COMP values in the edge aligned mode, use the following formula (again, this neglects dead-time):

$$\text{Duty Cycle} = \frac{\text{WG_COMP}}{\text{WG_RELOAD}} \times 100 \text{ percent}$$

6.7 Interrupt Generation

There are 2 interrupts associated with the WFG, WG and EXTINT. The WG interrupt is generated by the reload compare function, while the EXTINT interrupt is generated by an external event.

6.7.1 THE WG INTERRUPT

The WG interrupt is a “shared” interrupt with COMP5, which sets the WG pending bit in the PI_PEND register (Figure 6-7). When unmasked in the PI_MASK register, if either of these bits is set, the PI bit in the INT_PEND1 will be set. In turn, if the PI bit is unmasked in the INT_MASK1 register, the interrupt will take the vector located at 203AH. The user interrupt routine can read the PI_PEND register to determine what the source of the interrupt was. Note that reading PI_PEND clears all bits. Therefore, the value of the register must be stored in a shadow register if more than one bit needs to be checked. Also note that the PI_PEND bits cannot be set by writing to the PI_PEND register. This register is read only, writes will have no effect.

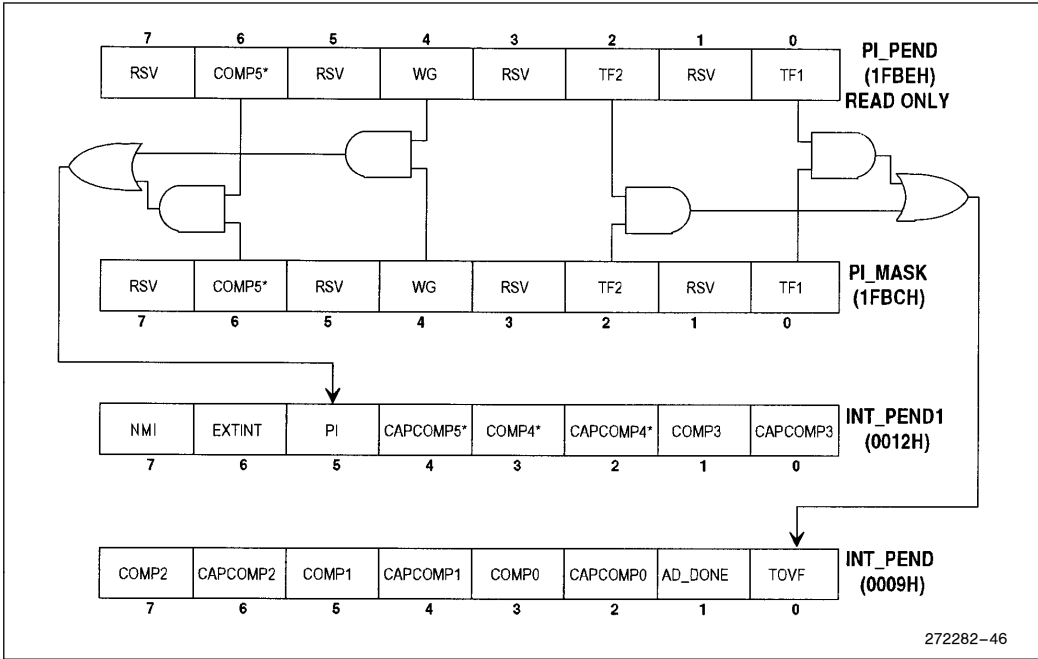


Figure 6-7. PI Interrupt Sharing

6.7.1.1 WG Interrupt Generation

In mode 0 the WG interrupt is generated once during the PWM period, when WG_COUNT = WG_RELOAD. In mode 1, the WG interrupt is generated twice: when WG_COUNT = 1 and also when WG_RELOAD = WG_COUNT. This corresponds to the maximum and minimum values of WG_COUNT (Figure 6-5).

In the edge aligned modes 2 and 3, the WG interrupt is generated once during the PWM period, when the counter is reset to 1 at the end of each PWM period (Figure 6-6).

Note that if WG_RELOAD contains too small a value, it is possible the counter will cycle too quickly for the software interrupt routines to keep up. Each application must be carefully analyzed for this condition.

6.7.2 THE EXTINT INTERRUPT

The EXTINT interrupt is generated by the WFG protection circuitry, which is described in Section 6-10. Two bits in WG_PROTECT (Figure 6-13), Interrupt Type (IT) and Enable Sampling (ES) control the type of external event which will cause EXTINT. EXTINT can be caused by a transition (rising or falling edge), or by a constant level (high or low). Table 6-2 shows the combinations for these bits. The EXTINT interrupt sets the EXTINT bit in the INT_PEND1 register, and if unmasked in the INT_MASKI register, takes the vector located at 203CH.

Table 6-2. EXTINT Mode Selection

ES	IT	EXTINT Input Characteristic
0	0	Falling Edge Triggered
0	1	Rising Edge Triggered
1	0	Low Level Triggered
1	1	High Level Triggered

The transition modes are selected by clearing the ES bit. To be a valid transition, the signal must remain asserted for a minimum period of $2 T_{osc}$ ($T_{osc} = 2/F_{XTAL}$). The IT bit controls whether a rising edge ($IT = 1$) or falling edge ($IT = 0$) causes the interrupt.

The level modes are selected by setting the ES bit. To be a valid level, the signal must remain asserted for a minimum period of $24 T_{osc}$. When the signal is asserted, sample circuitry monitors the input level 3 times during a $24 T_{osc}$ period. The signal must be asserted for each of the samples before it is recognized as valid. If the signal is valid, the EXTINT interrupt is generated. The IT bit controls whether a high level ($IT = 1$) or low level ($IT = 0$) input signal causes the interrupt.

The level mode is useful in noisy environments, where a noise spike might cause an unintended interrupt. Note that the same signal which generates the EXTINT also generates the output disable signal, discussed in Section 6.10.

6.8 Dead-Time Generator

The dead-time generator prevents an output and its compliment from being asserted at the same time. This may be required by the power driver circuitry to prevent complimentary output drivers from being turned on at the same time, resulting in catastrophic failure of the output circuitry! Protection from this condition is built into the 8XC196MC/MD, and is shown in Figure 6-8 (one of 3 channels).

6.8.1 DEAD-TIME GENERATOR OPERATION

The dead-time generator uses 2 internal signals to generate the non-overlapping output waveforms (Figure 6-9). Signal WFG is output from the transition detector. Signal DT from the dead-time counter is asserted (high) only when the counter has a value of 0. When a transition is detected from a phase comparator, the 10-bit dead-time counter is loaded with the 10-bit dead-time value contained in WG_CON, driving DT low. The timer then decrements once every state time ($1 \text{ state time} = 2/F_{XTAL}$) until the count reaches 0. At this point the counter stops and DT is driven high. Signal WFG is ANDed with DT, producing the WG_EVEN signal. Signal WFG# is ANDed with DT, producing the WG_ODD signal. The WG_EVEN and WG_ODD signals are passed to the output circuitry, described in Section 6.9.

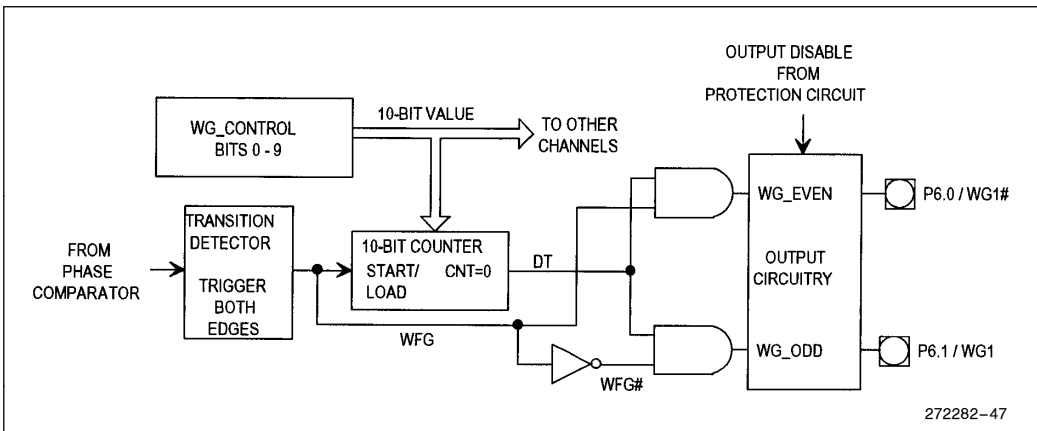


Figure 6-8. Dead-Time Generation

6.8.2 DEAD-TIME CALCULATION

The length of the dead-time is calculated using the following formula:

$$DT_VALUE = \frac{T_{dead} \times F_{XTAL}}{2}$$

Where

- DT_VALUE = the 10-bit value loaded into WG_CON
- T_{dead} = dead-time, in μs
- F_{XTAL} = XTAL1 clock input frequency, MHz

6.8.3 EFFECT OF DEAD-TIME ON PWM DUTY CYCLE

For relatively long pulse widths, short dead-times have little effect on the duty cycle of the waveforms. However, as the pulse width is narrowed and/or the dead-time is lengthened, the duty cycle will be affected and may need to be taken into account (Figure 6-9).

Because there is no hardware limit on minimum PWM pulse width, it is also possible to deassert one of the WFG outputs for the entire PWM period if the total dead-time is longer than the pulse width. For this reason, there should be a software limit check preventing the pulse width from being less than 3 × T_{dead}.

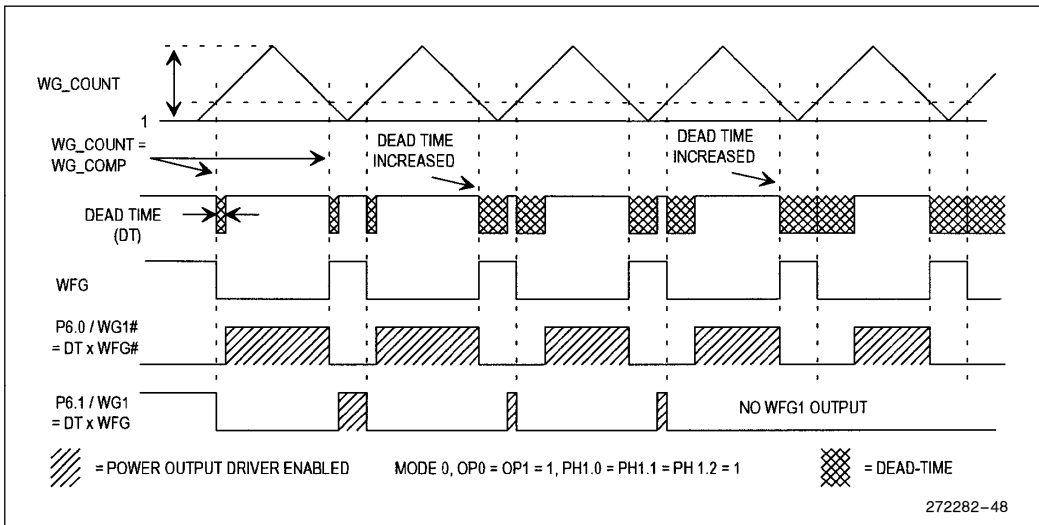


Figure 6-9. Effect of Dead-Time on WFG Duty Cycle

6.9 Output Control Circuitry

The output circuitry is controlled by the WG_OUT register. WG_OUT allows writing 0, 1, as well as assigning WG_EVEN or WG_ODD to the WFG output pins. A SYNC bit determines whether the WFG output pins are updated immediately or only under conditions specified in Table 6-1. Additionally, two bits (PE6, PE7) control port pins P6.6 and P6.7 and select I/O or PWM. A schematic of the output circuitry is given in Figure 6-11.

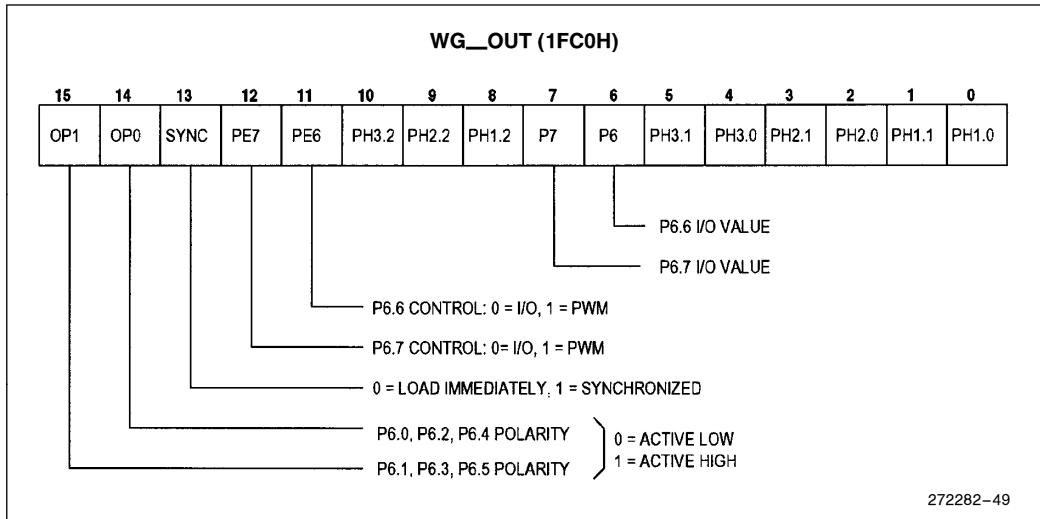


Figure 6-10. WG_OUT Register

6.9.1 PH CONTROL BITS

The PH control bits (Table 6-3) determine the values which are output on the WFG port pins. PHx.2 controls whether the port pins are I/O (PHx.2 = 0) or connected to the WFG (PHx.2 = 1). By writing the appropriate value to the PH bits, the port pins are driven high, low, or connected to the WG_EVEN or WG_ODD signal.

Note that the WG_OUT bits are arranged such that output mode selection is done in the upper byte, while pin value is determined by the lower byte. This facilitates updating the pin values using a byte-write instruction, which is faster than a word-write instruction.

Table 6-3. WG Output Configuration

Output Mode	PHx.2	PHx.1	PHx.0	WGx (1, 3, 5)	WGx# (0, 2, 4)
0	0	0	0	Low	Low
1	0	0	1	Low	High
2	0	1	0	High	Low
3	0	1	1	High	High
4	1	0	0	Low	Low
5	1	0	1	Low	WG_EVEN
6	1	1	0	WG_ODD	Low
7	1	1	1	WG_ODD	WG_EVEN#

Table assumes OP0 = OP1 = 1 (active high outputs). Values are inverted if output assigned to active low by OP bits

6.9.2 ENABLING AND DISABLING THE WFG OUTPUTS

The WFG outputs are enabled and disabled by a combination of the Disable Protection (DP) and Enable Output (EO) bits in WG_PROTECT (Figure 6-13). Note that for the outputs to be enabled, DP must = 0 and EO must = 1. Whenever the WFG outputs are disabled, they are driven to their deasserted state. Table 6-4 summarizes DP and EO bit combinations.

Note that if the EO bit is set in software immediately following the EXTINT event, the outputs will only be disabled for the time between the EXTINT event and the CPU write. This is because the EXTINT event generates a single short pulse which resets the EO bit. The EO bit can be set back to 1 immediately following this by the CPU, even if the EXTINT signal remains asserted. See Section 6-10 for a detailed explanation of the protection circuitry.





Table 6-4. Protection and Output Status

DP	EO	Protection/Output Status
0	0	Protection Enabled, Output Disabled
0	1	Protection Enabled, Output Enabled
1	0	Protection Disabled, Output Enabled
1	1	Protection Disabled, Output Enabled

6.9.3 OUTPUT POLARITY

Table 6-5 is given as an aid to understanding the polarity of the outputs when driven from the WFG. This table assumes that OP0 = OP1 = 1. Signals will be inverted if the respective OP bits are cleared. For the cases illustrated in Table 6-5, the high portion of the waveforms will get longer when dead time is increased. These drawings show a duty cycle of about 15%. Note that the OP bits affect all 3 phase driver channels together.

Table 6-5. Output Polarities, Modes 4–7

Output Mode	WGx (P6.1, P6.3, P6.5) OP1 = 1	WG# (P6.0, P6.2, P6.4) OP0 = 1
4	Low Always	Low Always
5	Low Always	
6		Low Always
7		

6.9.4 OUTPUT SCHEMATIC

Figure 6-11 illustrates the output schematic for one channel of the WFG. Operation of the upper half is as follows: The 3 PH bits select the signal source connected to the 8-input MUX. The output of this MUX is ANDed with the OD# signal generated by the protection circuitry (described in Section 6.10). This signal and its compliment feed a 2-input output polarity select MUX controlled by the OP0 bit. Finally, the output of this MUX feeds the output drivers Q1 and Q2 which are connected to P6.0. Note that if RESET is asserted, Q1 and Q2 will be disabled and the weak pullup Q5 will be turned on. Q5 can source about 10 μ A, and is always turned on asynchronously with RESET. Q5 will remain on until the first write of a “1” to the EO bit in WG_PROTECT.

The lower half of the circuit operates identically to the upper, with the exception that input MUX is connected to different signals, and the polarity select MUX is controlled by the OP1 bit.

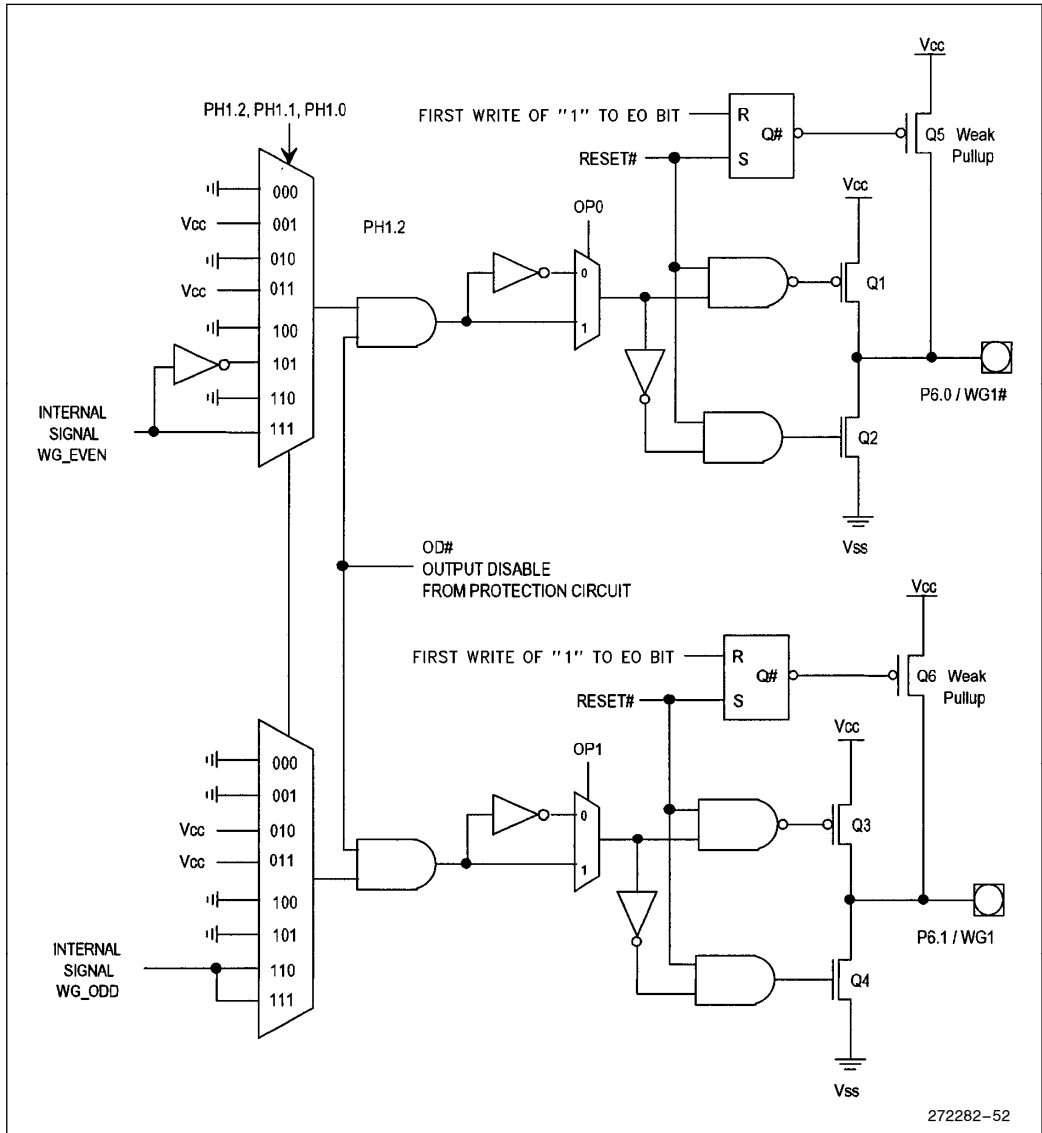


Figure 6-11. WFG Output Schematic

6.10 Protection Circuitry

The protection circuitry allows simultaneously deasserting all WFG outputs under software control or in response to an external event. This same external event will also generate the EXTINT interrupt, allowing software to stage a graceful recovery from an external error condition. Figure 6-12 is a block diagram of the protection circuitry, while Figure 6-13 details the WG_PROTECT register.

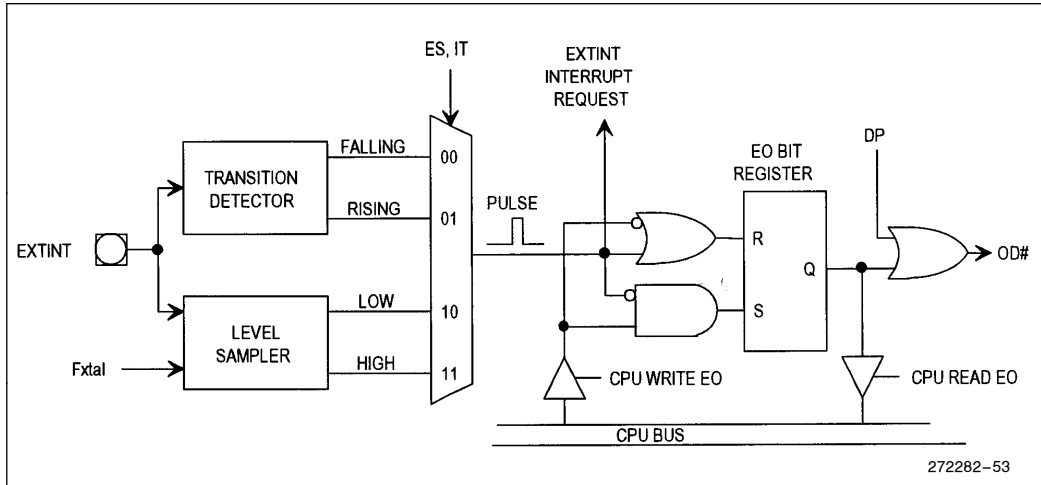


Figure 6-12. Protection Circuitry

6.10.1 PROTECTION CIRCUITRY OPERATION

The EXTINT input pin feeds both a transition (edge) detector and a level sampler. The transition detector can detect both a rising edge or a falling edge. However, to guarantee recognition as a valid transition, the input must remain asserted for at least $2 T_{osc}$ ($T_{osc} = 2/F_{XTAL}$). When a valid transition occurs, a single output pulse is generated.

The level detector can detect either a high or low level signal. To be recognized as a valid level, the signal must remain asserted for a minimum period of $24 T_{osc}$. When the signal is asserted, the level detect circuitry samples the level 3 times during a $24 T_{osc}$ period. When a valid level occurs, a single output pulse is generated.

A 4-input MUX controlled by the Enable Sampling (ES) and Interrupt Type (IT) bits selects which sort of event to recognize. The output pulse generates the EXTINT interrupt request, as well as resets the EO bit register (S-R latch, bit 0 of the WG_PROTECT register).

The output of the EO bit register is ORed with the Disable Protection (DP) bit to produce the Output Disable (OD#) signal which is passed to the output circuitry. When OD# is low, the WFG outputs will be driven to their deasserted state.

Note that if the EO bit is set in software immediately following the EXTINT generated pulse, the effect will be that the outputs are never disabled. For this reason, the EXTINT interrupt routine should be used to write a default output value to the WG_OUT register which will take effect immediately.

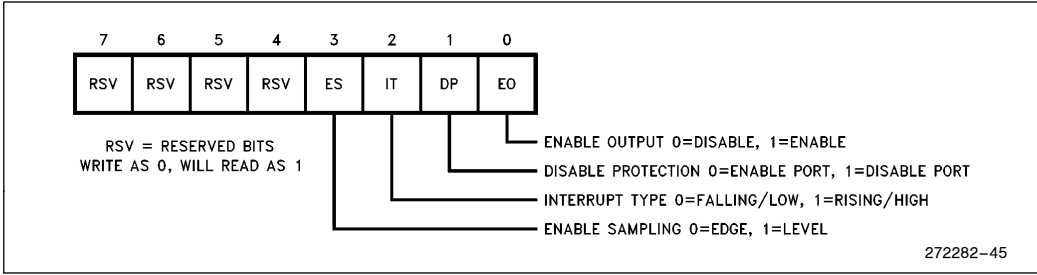


Figure 6-13. The WG_PROTECT Register

6.11 Software Example

The following software example allows the user to test operation of all the WFG registers, and observe how this effects the output waveforms. This is designed to run on a MC “DEMO” board, but is easily modified to operate on the MC “EVAL” board.

All variables are defined as words, and are masked to the appropriate length before being written to the WFG registers. This method is not compact, but is easy to code for and debug!

When running the program under the RISM monitor, any variable can be changed by the following command:

```
WORD.VARIABLE_NAME
```

The result will be immediately apparent on the WFG outputs!

This program can be downloaded from the Intel BBS, and is named “WFGTEST1.A96”.

```

$debug
;Program WFGTEST1.A96 to test WFG peripheral
; Jan 7, 1993
; Gary W. Harris
;
;$nolist
$include (c:\ecm\196mc\mc.inc)
$list
;
; This program allows modifying the WFG input parameters "on the fly"
; on the MC DEMO board.
; this allows easy testing of what is really going on
; within the MC device.
;
; First, set up the variables that I want to control:
;
rseg at 40h
;
mode:          dsw 1          ;mode = 0-3
op0:          dsw 1          ;P6.0,2,4 polarity--0=low, 1=high
    
```

272282-77

6.11 Software Example (Continued)

```

op1:      dsw 1      ;P6.1,3,5 polarity--0=low, 1=high
sync:    dsw 1      ;0=load now, 1 = synchronized
pe7:     dsw 1      ;P6.7 0=i/o, 1=PWM
pe6:     dsw 1      ;P6.6 0=i/o, 1=PWM
p7:      dsw 1      ;P6.7 I/O value
p6:      dsw 1      ;P6.7 I/O value
ph1:     dsw 1      ;P6.0,1 config
ph2:     dsw 1      ;P6.2,3 config
ph3:     dsw 1      ;P6.4,5 config
eo:      dsw 1      ;0=disable output, 1=enable output
dp:      dsw 1      ;0=enable protection, 1=disable
it:      dsw 1      ;0=falling edge trig, 1=rising edge
es:      dsw 1      ;0=edge, 1=sample
ec:      dsw 1      ;0=stop cntr, 1=start
dead:    dsw 1      ;10-bit dead time
reload:  dsw 1
comp1:   dsw 1
comp2:   dsw 1
comp3:   dsw 1
temp:    dsw 1
temp1:   dsw 1
temp2:   dsw 1
;
;
;*****
; initialize the values for these variables:
;*****
;
;
cseg at 0e000h                                ;demo board RAM
;
di
dpts
ld      sp,#200h
ld      mode,#0000h        ;mode0
ld      ec,#0001h         ;enable cntr
ld      dead,#0010h       ;1.6 us
ld      op0,#0001h        ;active high
ld      op1,#0001h        ;active high
ld      sync,#0001h       ;synchronized WG_OUT load
ld      pe7,#000000h      ;P6.7 in I/O mode
ld      pe6,#000000h      ;P6.6 in I/O mode
ld      p7,#000000h       ;P6.7 I/O = 0
ld      p6,#000000h       ;P6.6 I/O = 0
ld      ph1,#0007h        ;wfg both outputs
ld      ph2,#0007h        ;wfg both outputs
ld      ph3,#0007h        ;wfg both outputs
ld      eo,#0001h        ;enable outputs
ld      dp,#0001h        ;disable protection
ld      it,#0001h        ;rising/high edge trigger
ld      es,#0001h        ;24-state trigger
ld      reload,#1000h     ;1.024 ms mode0
ld      comp1,#0100h     ;64 μs
ld      comp2,#0200h     ;128 μs
ld      comp3,#0400h     ;256 μs
;
;*****
; now, initialize the WFG
;*****
;

```

6.11 Software Example (Continued)

```

;
; set-up interrupts
;
        ldb    temp,#00010000b    ;
        stb    temp,PI_MASK[r0]  ;unmask WG interrupt
        ldb    temp,#00100000b    ;
        ldb    int_mask1,temp;

;
;now load up WFG registers
;
        call wgout                ;initialize WG_OUT register
        call loadregs             ;initialize reload & compare regs
        call protect              ;initialize protection
        call wgcon                ;initialize WG_CON

;
;enable interrupts & loop here
;
        ei
        sjmp  $

;
;*****
; form WG_OUT value from variable data
;*****
;
wgout:   ld     temp,op1           ;get op1
        and   temp,#0001h        ;mask
        shl   temp,#15           ;move bit to correct loc

        ld     temp1,op0         ;et op0
        and   temp1,#0001h       ;mask
        shl   temp1,#14         ;move bit to correct loc
        or    temp1,temp         ;combine

        ld     temp,sync         ;get sync bit
        and   temp,#0001h       ;mask
        shl   temp,#13         ;move to correct loc
        or    temp1,temp        ;combine

        ld     temp,pe7         ;get pe7 bit
        and   temp,#0001h       ;mask
        shl   temp,#12         ;move to correct loc
        or    temp1,temp        ;combine

        ld     temp,pe6         ;get pe6 bit
        and   temp,#0001h       ;mask
        shl   temp,#11         ;move to correct loc
        or    temp1,temp        ;combine

        ld     temp,ph3         ;get ph3 bits
        and   temp,#0004h       ;mask for ph3.2
        shl   temp,#8h          ;move
        or    temp1,temp        ;combine

        ld     temp,ph2         ;get ph2 bits
        and   temp,#0004h       ;mask for ph2.2
        shl   temp,#7h          ;move
        or    temp1,temp        ;combine

```

272282-55

6.11 Software Example (Continued)

```

        ld    temp,ph1           ;get ph1 bits
        and  temp,#0004h        ;mask for ph1.2
        shl  temp,#6h           ;move
        or   temp1,temp         ;combine

        ld    temp,p7           ;get p7 bit
        and  temp,#0001h        ;mask
        shl  temp,#7           ;move to correct loc
        or   temp1,temp         ;combine

        ld    temp,p6           ;get p6 bit
        and  temp,#0001h        ;mask
        shl  temp,#6           ;move to correct loc
        or   temp1,temp         ;combine

        ld    temp,ph3          ;get ph3 bits again
        and  temp,#0003h        ;mask for ph3.0 & 1
        shl  temp,#4h           ;move
        or   temp1,temp         ;combine1

        ld    temp,ph2          ;get ph2 bits again
        and  temp,#0003h        ;mask for ph2.0 & 1
        shl  temp,#2h           ;move
        or   temp1,temp         ;combine

        ld    temp,ph1          ;get ph1 bits again
        and  temp,#0003h        ;mask for ph3.0 & 1
        or   temp1,temp         ;combine, don't need to move

        st   temp1,WG_OUT[r0]   ;now store it!
        ret

;
;*****
; form control reg word
;*****
;
wgcon:  ld    temp,mode         ;get mode
        and  temp,#0003h        ;mask
        shl  temp,#12           ;shift to correct location

        ld    temp1,ec          ;start/stop bit
        and  temp1,#0001h        ;mask
        shl  temp1,#10          ;shift to correct loc
        or   temp,temp1         ;combine into temp

        ld    temp1,dead        ;get dead time
        and  temp1,#03ffh        ;mask to 10 bits
        or   temp,temp1         ;combine into temp
        st   temp,WG_CON[r0]    ;store WG_CON
        ret

;
;*****
;load the reload and comp registers
;*****
;
loadregs: st   reload,WG_RELOAD[r0]
          st   comp1,WG_COMP1[r0]
          st   comp2,WG_COMP2[r0]
    
```

272282-56

6.11 Software Example (Continued)

```

                st    comp3,WG_COMP3[r0]
                ret
;
;*****
;load protection options
;*****
;
protect:        ld    temp,es                ;get sample control bit
                and   temp,#0001h          ;mask
                shl   temp,#3              ;shift to correct location

                ld    temp1,it              ;interrupt type bit
                and   temp1,#0001h         ;mask
                shl   temp1,#2             ;shift to correct loc
                or    temp,temp1           ;combine into temp

                ld    temp1,dp              ;disable protection bit
                and   temp1,#0001h         ;mask
                shl   temp1,#1             ;shift to correct loc
                or    temp,temp1           ;combine into temp

                ld    temp1,eo              ;enable output bit
                and   temp1,#0001h         ;mask
                or    temp,temp1           ;combine into temp

                stb   temp,WG_PROTECT[r0] ;store byte into WG_PROTECT
                ret
;
;*****
;interrupt routine for WFG
;*****
;
cseg at 0f1a0h                ;demo board PI interrupt

                pusha
                call  wgout                  ;update WG_OUT register
                call  loadregs              ;update reload & compare regs
                call  protect                ;update protection options
                call  wgcon                 ;update WG_CON
                popa
                ret

end

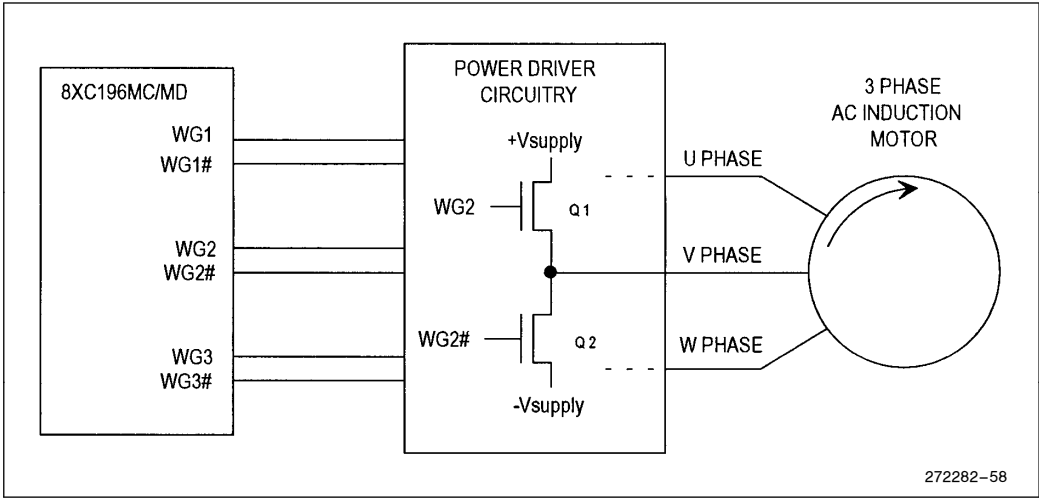
```

272282-57

7.0 3-PHASE INDUCTION MOTOR CONTROL PROGRAM SAMPLE

7.1 Introduction

The following program example shows how the WFG can be used to generate a 3-phase output suitable for driving a 3-phase AC induction motor. Figure 7-1 shows a high-level view of how the 8XC196MC/MD would drive each phase of the motor. Each phase of the motor is driven from a complimentary output power driver, with Q1 driven from the WGx output, and Q2 driven by the WGx# outputs. Note that dead-time is required to prevent Q1 and Q2 from being on at the same time.

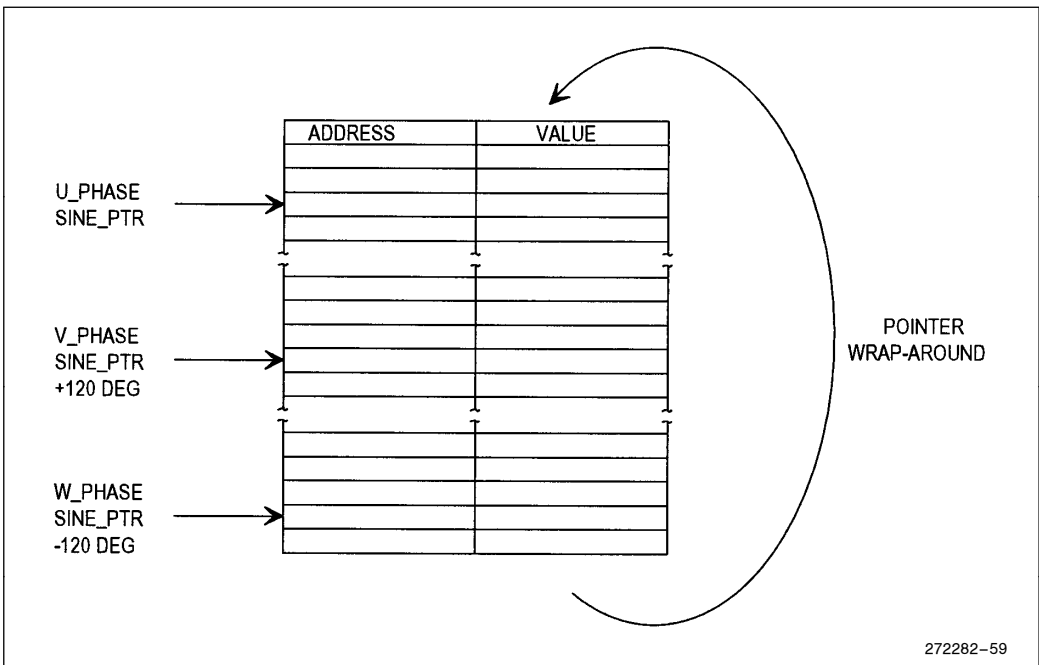


272282-58

Figure 7-1. 3-Phase Induction Motor System

7.1.1 SINE LOOK-UP TABLE

In order to produce the 3 waveforms with a 120° phase difference, a look-up table is used, Figure 7-2. This table contains the normalized values of a 180° segment of the sine function. Each value is a 15-bit integer, where 0 = sin(0) and 32,767 = sin(90). The table is arranged in 0.15° increments (1200 entries) from 0° to 179.85°. Since the sine wave is symmetrical, 0° to 180° is used in the table, although other arrangements are possible.



272282-59

Figure 7-2. Sine Look-Up Table

The U, V and W phase pointers into the table are calculated starting from the SINE_PTR, which increments from 0 to 4800 on word boundaries, representing 360°. The U-phase is calculated directly using the SINE_PTR, while the V-phase adds 3200 (240°), and the W-phase adds 1600(120°). Figure 7-3 shows how the pointers are related.

Since the table is only 2400 bytes long, some manipulation of the pointers is done to wrap them around to the correct table location. The software also keeps track of whether the phase is in the negative or positive part of the cycle.

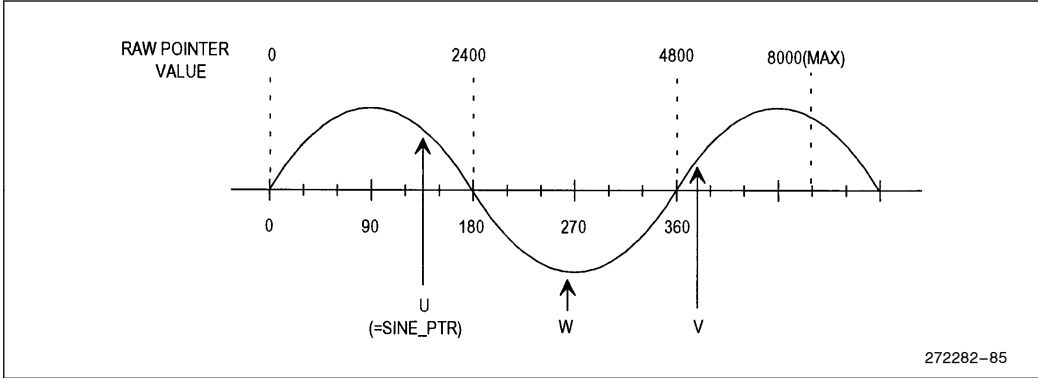


Figure 7-3. Raw Pointer Values

7.1.2 FREQUENCY-TO-VOLTAGE LOOK-UP TABLE

A frequency/voltage look-up table is also used in this example (called RAMDA in the program listing). This table is read to determine what the RMS output voltage is for each modulation frequency. The RAMDA look-up table stores the V/F relationship that is desired for a particular motor.

The RMS output voltage is determined by the percentage variation in the PWM duty cycles. Referring to Figure 7-1, note that for a WFG duty cycle of 50%, the average output voltage will be 0V. This is called the phase neutral point. As the duty cycle is changed upward to 100%, the average voltage will increase to +Vsupply, and as the duty cycle is lowered to 0%, the average voltage will decrease to -Vsupply.

The amount of variation in the PWM duty cycle is called the modulation depth. Figure 7.4 shows the effect of changing the modulation depth from 100% to 20%. The RMS voltage output will change from Vsupply to 0.2 × Vsupply. Note that the modulation frequency remains unchanged.

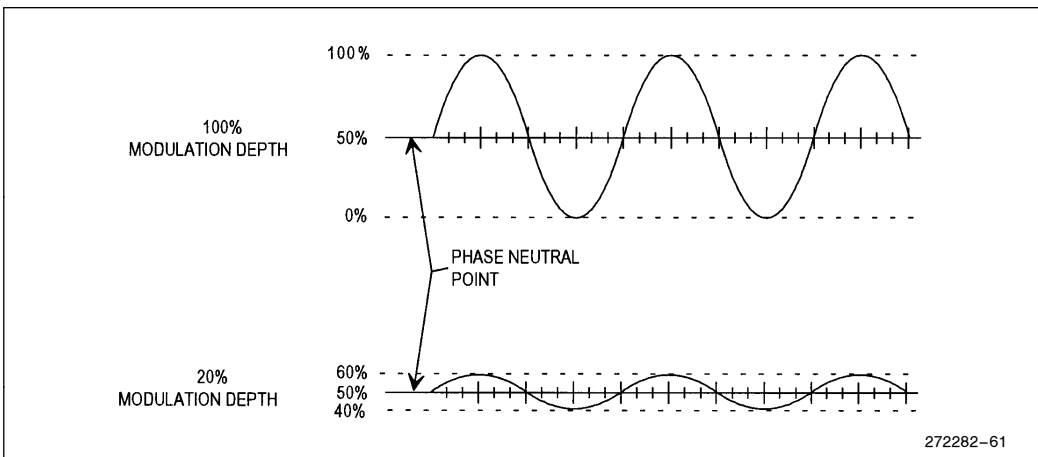


Figure 7-4. Modulation Depth

After the RAMDA and sine values are read from the tables, the PWM duty cycle is calculated using the following formulae:

$$U_PHASE = NEUTRAL + (RAMDA_VALUE \times SINE_VALUE) [0^\circ \text{ to } 180^\circ]$$

$$U_PHASE = NEUTRAL - (RAMDA_VALUE \times SINE_VALUE) [180^\circ \text{ to } 360^\circ]$$

This calculation gives a 32-bit result, of which only the most significant word is used to load the WG_COMP register. The phase neutral value (TC1_4) is equated to 417, which is 1/2 of the maximum value of WG_RELOAD, giving a 50% duty cycle.

Finally, after the WG_COMP registers are loaded, the SINE_PTR is stepped to its next value. The size of the steps taken through the sine look-up table determine what the modulation frequency is. As this frequency increases, the steps become larger, and for a given carrier frequency the sine wave will take on a more stair-step appearance.

7.1.3 OUTPUT WAVEFORMS

Figure 7-5 shows when the outputs are asserted. Note that each waveform represents WGx - WGx#. A value of +1 indicates that WGx is asserted, while a value of -1 indicates that WGx# is asserted. A value of 0 indicates that both outputs are either asserted (we hope not!) or deasserted. This is a convenient way to display this information.

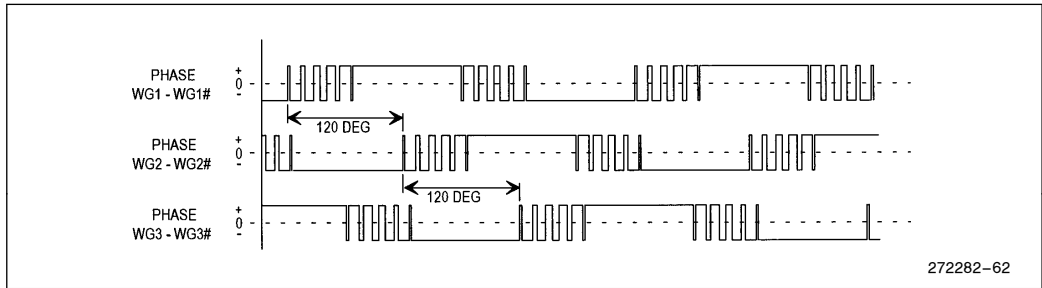


Figure 7-5. WFG Output Drive

After the PWMs are integrated by a motor or transformer (typically an excellent low-pass filter), they will appear as sine waves, illustrated in Figure 7-6.

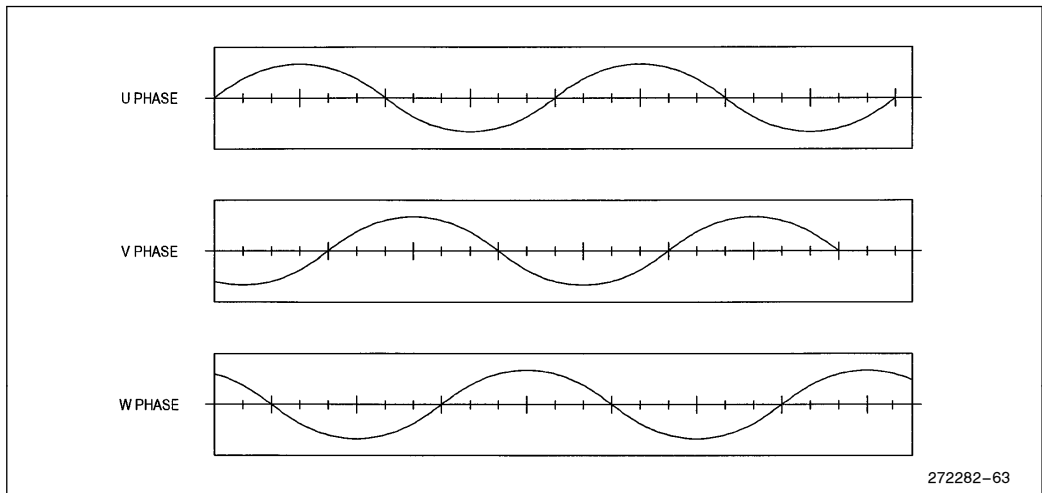


Figure 7-6. WFG Output Waveforms after Low-Pass Filter

7.2 Detailed Program Description

7.2.1 CONSTANT DECLARATIONS (Lines 1–42)

Lines 5–9 define some of the operating conditions for the program. Minimum and maximum frequency limits are established and the carrier frequency/period is defined.

Lines 16–30 define register variables that are used later in the program. Lines 32–41 define names for the upper and lower half of long (32-bit) variables, and the FCOMMAND word variable.

7.2.2 INTERRUPT VECTORS AND CCB (Lines 43–57)

Lines 48 and 49 fill in the interrupt vector table with the locations of the WG__INT and EXT__INT interrupt routines. Only conventional interrupts are used in this program example, no PTS routines.

Lines 56 and 57 define the chip configuration bytes, CCB and CCB1. These need to be configured for the particular system that this program is run on.

7.2.3 MAIN PROGRAM (Lines 58–93)

Lines 62–69 define the program starting location (2080H), set up the stack at 0200H, and disable and clear out any pending interrupts.

Line 70 unmarks the PI and EXTINT interrupts. Since the PI interrupt is a shared interrupt, line 73 unmarks the WG interrupt bit in PI__MASK.

Lines 77–85 initializes the variable values, and the pointers into the SINE and RAMDA look-up tables.

Lines 86–89 initialize the WFG output, protection and reload registers. Line 90 calls the WG__INT interrupt routine, which calculates and loads the WG__COMP registers, thus establishing the initial PWM duty cycles. WG__INT is described in detail in Section 7.2.4. Since WG__COUNT is stopped, the WG__COMP registers will be immediately loaded with these initial values.

Line 91 starts the WG__COUNT counter running, and line 92 sets the PI interrupt bit. This forces WG__INT to be called again immediately after line 93, which enables the interrupts. This is necessary in order to load the WG__COMP buffer registers with the next set of values.

7.2.4 POLLING ROUTINE (Lines 94–126)

Lines 102–114 read a value input on port 3, compare it against the FMIN and FMAX values and check that it is within FMIN and FMAX bounds. The value is then stored in FCOMMAND, and output on port 4, which is connected to some LED's as a monitor. FCOMMAND is used by the following code to set-up the step-size the pointers take through the sine look-up table. Note that this routine must be modified to run on any system which uses external memory!

Lines 118 and 119 load the RAMDA pointer with FCOMMAND value, and multiply it by 8 (by shifting left 3). Line 120 disables the interrupts, to protect the following code from unintended variable modification.

Line 121 calculates the offset into the RAMDA table based on the $8 \times$ FCOMMAND value, and loads the table value into the RAMDA__ACT variable. RAMDA__ACT is used in the WG__INT routine in calculating the modulation “depth” of the PWM, thus controlling what the voltage will be for a given output frequency.

Line 123 loads the SINE__STEP variable with the FCOMMAND value, establishing the “modulation” frequency, the frequency of the output sine wave. Line 124 forces the steps to occur only on even addresses, otherwise a bogus value can be accessed from the table.

Lines 125–126 re-enable the interrupts, and loop back to line 98, where port 3 is read again, starting the update process again.

7.2.5 WG INTERRUPT ROUTINE (Lines 127–211)

The WG__INT interrupt routine is entered every carrier period. It calculates the new pointer positions, and the new values for the WG__COMP registers. These calculations are performed for each phase, and then loaded into the WG__COMP registers. The logic of these routines is a bit confusing, so only the top-level of each segment is detail here. Refer to the flow charts (Figures 7-7 and 7-8) for a more detailed understanding.

Line 133 saves the current CPU status.

Lines 141–144 calculate the U__PHASE value for positive portions of the sine wave, while lines 145–154 calculate the U__PHASE value for negative portions.

Lines 155–166 calculate the V__PHASE value for positive portions of the sine wave, while lines 167–176 calculate the V__PHASE value for negative portions.

Lines 177–188 calculate the W__PHASE value for positive portions of the sine wave, while lines 189–198 calculate the W__PHASE value for negative portions.

Lines 202–204 load the WG__COMP registers with the new duty cycles.

Lines 205–208 increment the SINE__PTR to the next position in the sine table, and wrap-around the pointer as necessary.

Lines 210–211 restore the CPU flags and return to the polling loop.

7.2.6 EXTINT INTERRUPT ROUTINE (Lines 212–223)

The EXTINT interrupt routine shown here is just a “hook” for some interrupt code that the user would use to shut-down the device in response to some external condition.

7.2.7 SINE LOOK-UP TABLE (Lines 224–1429)

The SINE look-up table is discussed in Section 7-1. This table consists of 1200 word values representing the sine function normalized to $32,767 = \sin(90)$. These are input as an include file, file name SINES.INC.

7.2.8 RAMDA LOOK-UP TABLE (Lines 1430–2235)

The RAMDA look-up table controls the modulation depth of the PWMs for any given modulation frequency. This table consists of 400 word values for 0.25 Hz steps 0 Hz to 200 Hz. This table establishes the volts per frequency transfer function for a given motor, and is very application dependent.

7.3 Top 5 Issues for the 3-Phase Induction Motor Control Example

1. WG__INT execution time must be less than the carrier period.
2. The SINE look-up table only needs 0° to 90° to contain the necessary information for the pointers. However, 180° or 360° tables allow faster pointer calculation.
3. The RAMDA (modulation depth) table should be optimized for the motor being used. Simple V/F relationships can be calculated instead of using a table.
4. It is best to maintain an odd number of carrier periods per 180° of modulation in order to minimize harmonic distortion.
5. The WG registers are initialized twice: The first time with the counter not running, which loads the registers directly, and the second time immediately after the counter is started, which loads the buffers.

7.4 Program Example

```

1  ;*****
2  ; WAVEFORM GENERATOR CARRIER FREQUENCY
3  ;*****
4  ;
5  CARRIER_FREQ EQU          4800          ;4.8KHz
6  HALF_PERIOD EQU           833           ;(1/CARRIER_FREQ)/0.125E-6/2
7  NEUTRAL EQU              HALF_PERIOD/2 ;sine wave zero cross point
8  MIN_FREQ EQU             6             ;minimum inverter frequency
9  MAX_FREQ EQU             200           ;maximum inverter frequency
10 ;
11 ;*****
12 ;USER DEFINED REGISTERS
13 ;*****
14 RSEG          AT          LAH
15
16 U_PHASE:      DSL          1
17 V_PHASE:      DSL          1
18 W_PHASE:      DSL          1
19 TEMP_PHASE:   DSL          1
20 FMAX:         DSW          1
21 FMIN:         DSW          1
22 TC1_2:        DSW          1
23 TC1_4:        DSW          1
24 SINE_PTR:     DSW          1
25 SINE_PTR1:    DSW          1
26 SINE_STEP:    DSW          1
27 RAMDA_PTR:    DSW          1
28 RAMDA_ACT:    DSW          1
29 FCOMMAND:     DSW          1
30 TEMP1:        DSB          1
31
32 U_PHASE_L EQU (U_PHASE) :WORD
33 U_PHASE_H EQU (U_PHASE+2) :WORD
34 V_PHASE_L EQU (V_PHASE) :WORD
35 V_PHASE_H EQU (V_PHASE+2) :WORD
36 W_PHASE_L EQU (W_PHASE) :WORD
37 W_PHASE_H EQU (W_PHASE+2) :WORD
38 TEMP_PHASE_L EQU (TEMP_PHASE) :WORD
39 TEMP_PHASE_H EQU (TEMP_PHASE+2) :WORD
40 FCOMMAND_L EQU (FCOMMAND) :BYTE
41 FCOMMAND_H EQU (FCOMMAND + 1) :BYTE
42 ;

```

7.4 Program Example (Continued)

```

43 ;*****
44 ; INTERRUPT VECTOR TABLE
45 ;*****
46 CSEG AT 203AH
47
48 DCW WG_INT ;INT 13
49 DCW EXT_INT ;INT 14
50
51 ;*****
52 ; CHIP CONFIGURATION BYTES
53 ;*****
54 CSEG AT 2018H
55 ;
56 DCW 20CFH ;CCB
57 DCW 20DCH ;CCB1
58 ;
59 ;*****
60 ; MAIN ROUTINE
61 ;*****
62 CSEG AT 2080H
63 MAIN_START:
64 DI ;disable interrupt
65 LD SP,#0200H ;set-up stack pointer
66 ;
67 ;Clear interrupt mask register
68 ;
69 CLRB INT_MASK ;reset interrupt mask register
70 LDB INT_MASK1,#60H ;enable WG_COUNTER and EXTINT
71 ;interrupt sources
72 LDB WSR,#3EH ;map 64 bytes to 1F80H-1FBFH
73 LDB PI_MASK_WO,#10H ;set-up peripheral interrupt mask reg.
74 ;
75 ;Initialize WG
76 ;
77 LD FMIN,#MIN_FREQ ;set minimum inverter frequency
78 LD FMAX,#MAX_FREQ ; maximum inverter frequency
79 LD TC1_4,#NEUTRAL ;sine wave phase neutral point
80 LD TC1_2,#HALF_PERIOD ;half of carrier period
81 CLR SINE_PTR ;clear sine table pointer
82 LD SINE_STEP,FMIN ;set initial inverter frequency
83 LD RAMDA_PTR,#48 ;set initial ramda
84 LD RAMDA_ACT,RAMDA[RAMDA_PTR]
85 LDB WSR,#3FH ;map 64 bytes to 1FC0H-1FFFH
86 LD WG_OUT_WO,#07FFH ;set WG output register
87 LDB WG_PROTECT_WO,#05H ;enable WG output,sampling circuit
88 ;and rising edge trigger
89 LD WG_RELOAD_WO,TC1_2 ;set half carrier period
90 CALL WG_INT ;initialize WG_COMPARE register
91 LD WG_CON_WO,#0428H ;Mode0,td=5us,start counter
92 LDB INT_PEND1,#20H ;set next values to WG_COMPAREn
93 EI ;enable interrupt
94 ;
    
```

7.4 Program Example (Continued)

```

95 ;*****
96 ; POLLING ROUTINE
97 ;*****
98 LOOP:
99 ;
100 ;load modulation frequency
101 ;
102     LDB     FCOMMAND_L,P3PIN[0] ;load modulation frequency from DIP SW
103     CMP     FCOMMAND,FMIN       ;check lowest frequency
104     BGE     NEXT1                ; if it's lower than FMIN
105     LD      FCOMMAND,FMIN       ; then set FMIN
106     BR     NEXT2
107 NEXT1:
108     CMP     FCOMMAND,FMAX       ;check highest frequency
109     BLE     NEXT2                ; if it's higher than FMAX
110     LD      FCOMMAND,FMAX       ; then set FMAX
111 NEXT2:
112     LDB     TEMP1,FCOMMAND       ;display frequency command on LED
113     NOTB   TEMP1
114     STB    TEMP1,P4REG[0]
115 ;
116 ;Set RAMDA and modulation frequency
117 ;
118     LD      RAMDA_PTR,FCOMMAND   ;RAMDA_PTR=FCOMMAND/0.25*2
119     SHL    RAMDA_PTR,#3         ; due to 0.25Hz step & word access
120     DI                                           ;protect from individual modification
121     LD      RAMDA_ACT,RAMDA[RAMDA_PTR]
122                                           ;refer to V/F table
123     LD      SINE_STEP,FCOMMAND   ;set modulation frequency
124     ANDB   SINE_STEP,#0FEH      ;disable odd address access
125     EI
126     SJMP   LOOP
127 ;
128 ;*****
129 ; WG_COUNTER INTERRUPT ROUTINE
130 ;*****
131 ;
132 WG_INT:
133     PUSHA                                ;save PSW,INT_MASK,INT_MASK1,WSR
134 ;
135 ;U,V,W-phase modification
136 ;
137 U_PHASE_START:
138     LD      WSR,#3FH                ;map 64 bytes to 1FCOH-1FFFH
139     CMP     SINE_PTR,#2400          ;check zero cross point on sine u-wave
140     BGT     U_PHASE_NEG            ; if pointer > 2400 then do negative
141 U_PHASE_POS:
142     MULU   U_PHASE,RAMDA_ACT,SINE[SINE_PTR] ;U_PHASE=NEUTRAL+RAMDA*SIN(u)
143     ADD    U_PHASE_H,TC1_4,U_PHASE_H
144     BR     U_PHASE_END

```

7.4 Program Example (Continued)

```

145  U_PHASE_NEG:
146      SUB     SINE_PTR1,SINE_PTR,#2400 ;get negative sine value
147      LD      U_PHASE_L,RO             ;U_PHASE=NEUTRAL-RAMDA*SIN(u)
148      LD      U_PHASE_H,TCL_4
149      MULU    TEMP_PHASE,RAMDA_ACT,SINE[SINE_PTR1]
150      SUB     U_PHASE_L,TEMP_PHASE_L
151      SUBC    U_PHASE_H,TEMP_PHASE_H
152      BGE     U_PHASE_END             ;if U_PHASE < 0
153      CLR     U_PHASE_H               ; then set U_PHASE=0
154  U_PHASE_END:
155  V_PHASE_START:
156      ADD     SINE_PTR1,SINE_PTR,#3200 ;get sine table pointer +120'
157      CMP     SINE_PTR1,#4800         ; check pointer overflow
158      BLE     V_PHASE_l               ; if pointer > 4800
159      SUB     SINE_PTR1,#4800         ; then wraparound pointer
160  V_PHASE_l:
161      CMP     SINE_PTR1,#2400         ;check zero cross point on sine v-wave
162      BGT     V_PHASE_NEG             ; if pointer >2400 then do negative
163  V_PHASE_POS:
164      MULU    V_PHASE,RAMDA_ACT,SINE[SINE_PTR1]
165      ADD     V_PHASE_H,TCL_4,V_PHASE_H
166      BR      V_PHASE_END
167  V_PHASE_NEG:
168      SUB     SINE_PTR1,SINE_PTR1,#2400 ;get negative sine value
169      LD      V_PHASE_L,RO             ;V_PHASE=NEUTRAL-RAMDA*SIN(v)
170      LD      V_PHASE_H,TCL_4
171      MULU    TEMP_PHASE,RAMDA_ACT,SINE[SINE_PTR1]
172      SUB     V_PHASE_L,TEMP_PHASE_L
173      SUBC    V_PHASE_H,TEMP_PHASE_H
174      BGE     V_PHASE_END             ;if V_PHASE < 0
175      CLR     V_PHASE_H               ; then set V_PHASE=0
176  V_PHASE_END:
177  W_PHASE_START:
178      ADD     SINE_PTR1,SINE_PTR,#1600 ;get sine table pointer -120'
179      CMP     SINE_PTR1,#4800         ; check pointer overflow
180      BLE     W_PHASE_l               ; if pointer > 4800
181      SUB     SINE_PTR1,#4800         ; then wraparound pointer
182  W_PHASE_l:
183      CMP     SINE_PTR1,#2400         ;check zero cross point on sine w-wave
184      BGT     W_PHASE_NEG             ; if pointer < 0 then do negative
185  W_PHASE_POS:
186      MULU    W_PHASE,RAMDA_ACT,SINE[SINE_PTR1]
187      ADD     W_PHASE_H,TCL_4,W_PHASE_H
188      BR      W_PHASE_END
189  W_PHASE_NEG:
190      SUB     SINE_PTR1,SINE_PTR1,#2400 ;get negative sine value
191      LD      W_PHASE_L,RO             ;W_PHASE=NEUTRAL-RAMDA*SIN(w)
192      LD      W_PHASE_H,TCL_4
193      MULU    TEMP_PHASE,RAMDA_ACT,SINE[SINE_PTR1]
194      SUB     W_PHASE_L,TEMP_PHASE_L

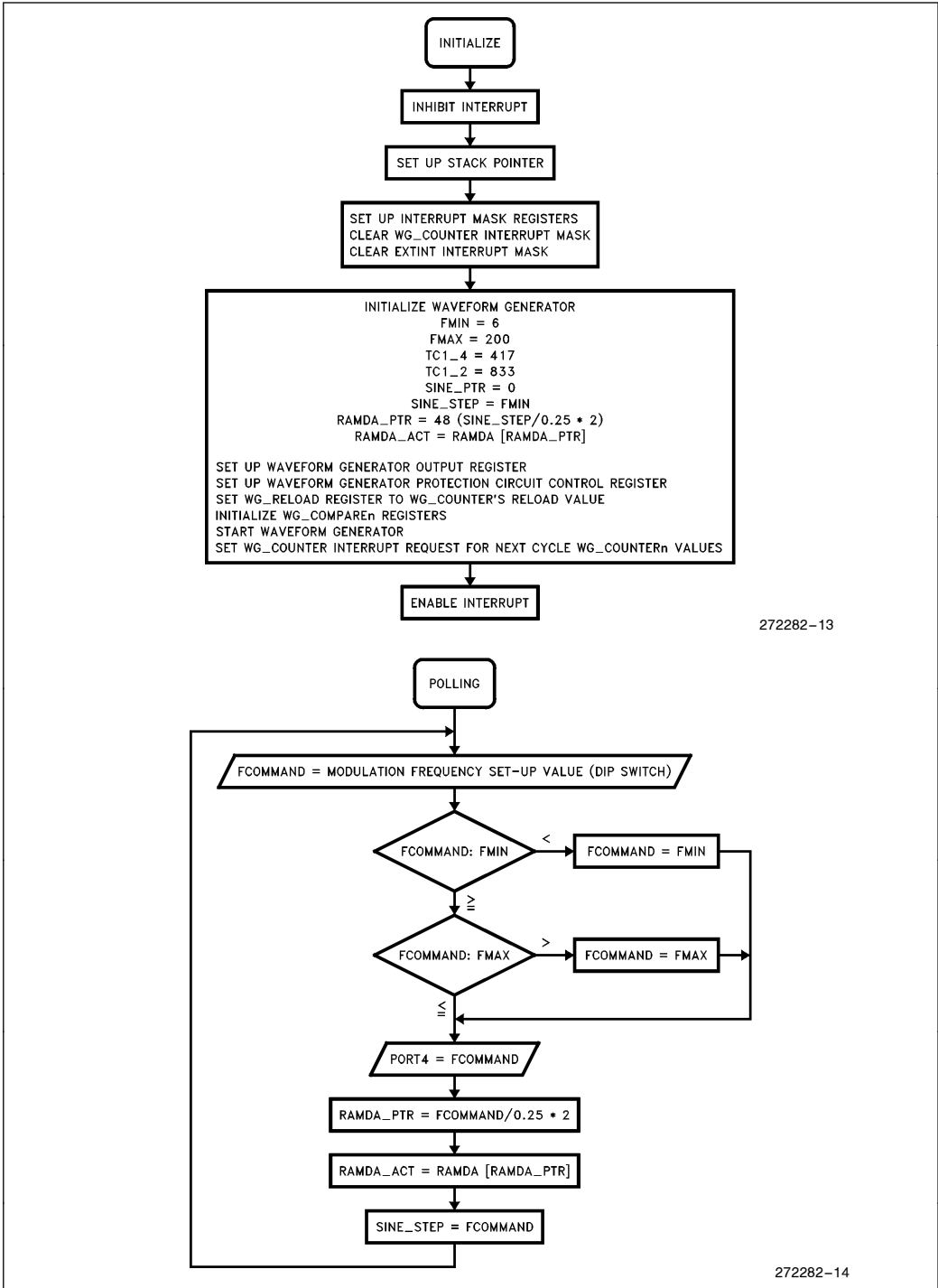
```

7.4 Program Example (Continued)

```

195         SUBC    W_PHASE_H,TEMP_PHASE_H
196         BGE     W_PHASE_END           ;if W_PHASE < 0
197         CLR     W_PHASE_H           ; then set W_PHASE=0
198 W_PHASE_END:
199 ;
200 ;Modify PWM duty cycle, it will be loaded on next WG interrupt
201 ;
202         LD      WG_COMPL_WO,U_PHASE_H ;set next value to WG_COMPARE1 reg.
203         LD      WG_COMP2_WO,V_PHASE_H ;set next value to WG_COMPARE2 reg.
204         LD      WG_COMP3_WO,W_PHASE_H ;set next value to WG_COMPARE3 reg.
205         ADD     SINE_PTR,SINE_STEP   ;set next sine table pointer
206         CMP     SINE_PTR,#4800       ; if pointer makes overflow
207         BLT     WG_INT_END           ; then reset pointer
208         CLR     SINE_PTR             ; otherwise complete the modification
209 WG_INT_END:
210         POPA                    ;load PSW,INT_MASK,INT_MASK1,WSR
211         RET                      ;return to main loop
212 ;
213 ;*****
214 ; EXTERNAL INTERRUPT ROUTINE
215 ;*****
216 ;
217 EXT_INT:
218         PUSHA                    ;save PSW,INT_MASK,INT_MASK1,WSR
219         -
220         -
221         -
222         POPA                    ;load PSW,INT_MASK,INT_MASK1,WSR
223         RET                      ;return to main loop
224 ;
225 ;*****
226 ;Sine table for sinusoidal 3 phase PWM generation
227 ;*****
228 ;
229 SINE:
230         DCW     00000H           ;value by word (0.15° step)
231         -
232         -                       ; -
233         830    DCW     07FFFH           ;sin(90)
234         -
235         -                       ; -
1429        DCW     00056H           ;sin(179.85)
1430 ;
1431 ;*****
1432 ;V/F table for sinusoidal 3 phase PWM generation
1433 ;*****
1434 ;
1435 RAMDA:
1436        DCW     00000H           ;value by word (0.25Hz step)
1437        -
1438        -                       ; -
1836        DCW     00478H           ;Ramda(100Hz)
1439        -
1440        -                       ; -
2235        DCW     004B0H           ;Ramda(200Hz)
2236
2237 END

```

272282-13

272282-14

Figure 7-7. Flow Chart—3-Phase A.C. Induction Motor Initialization

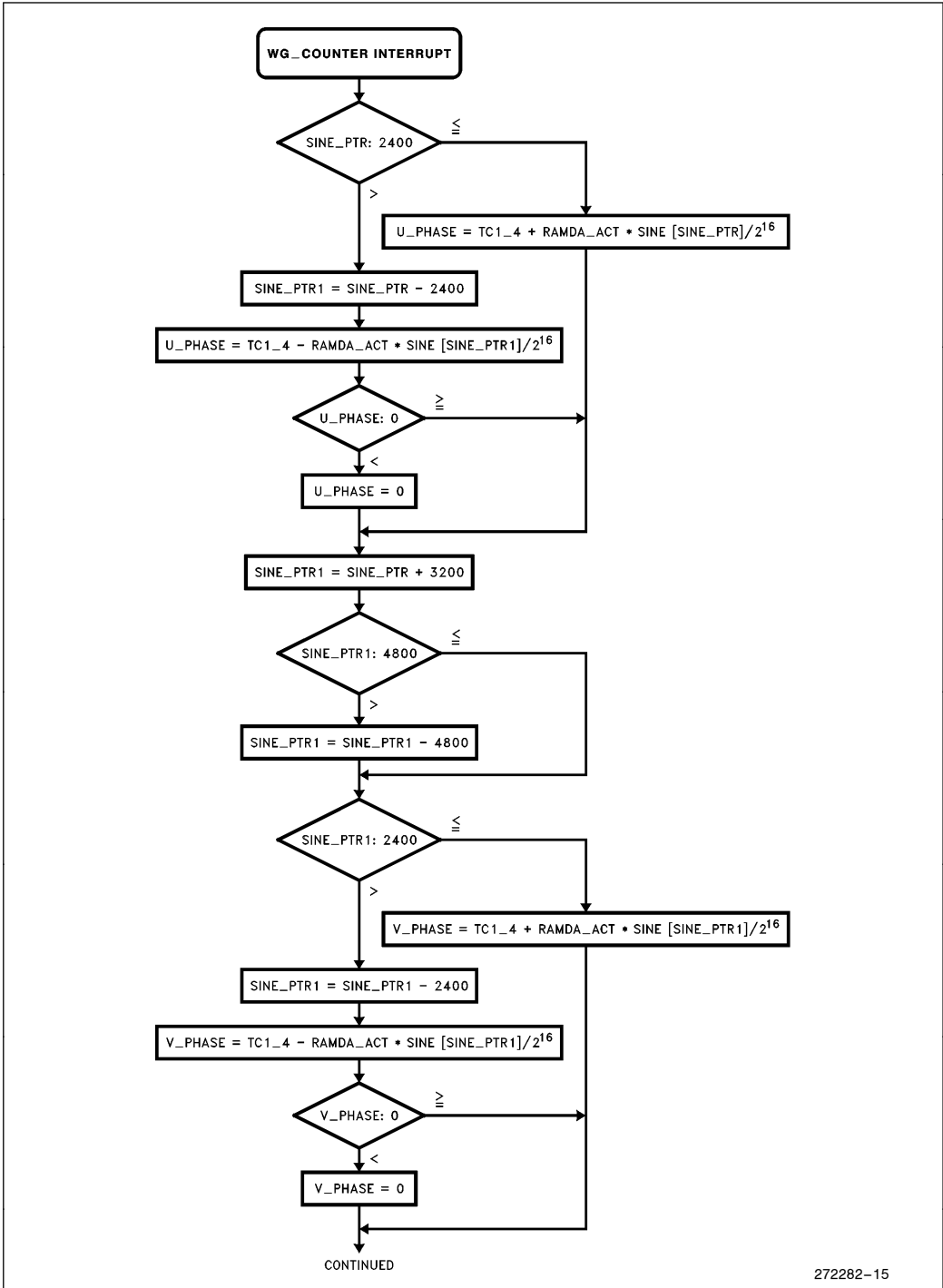
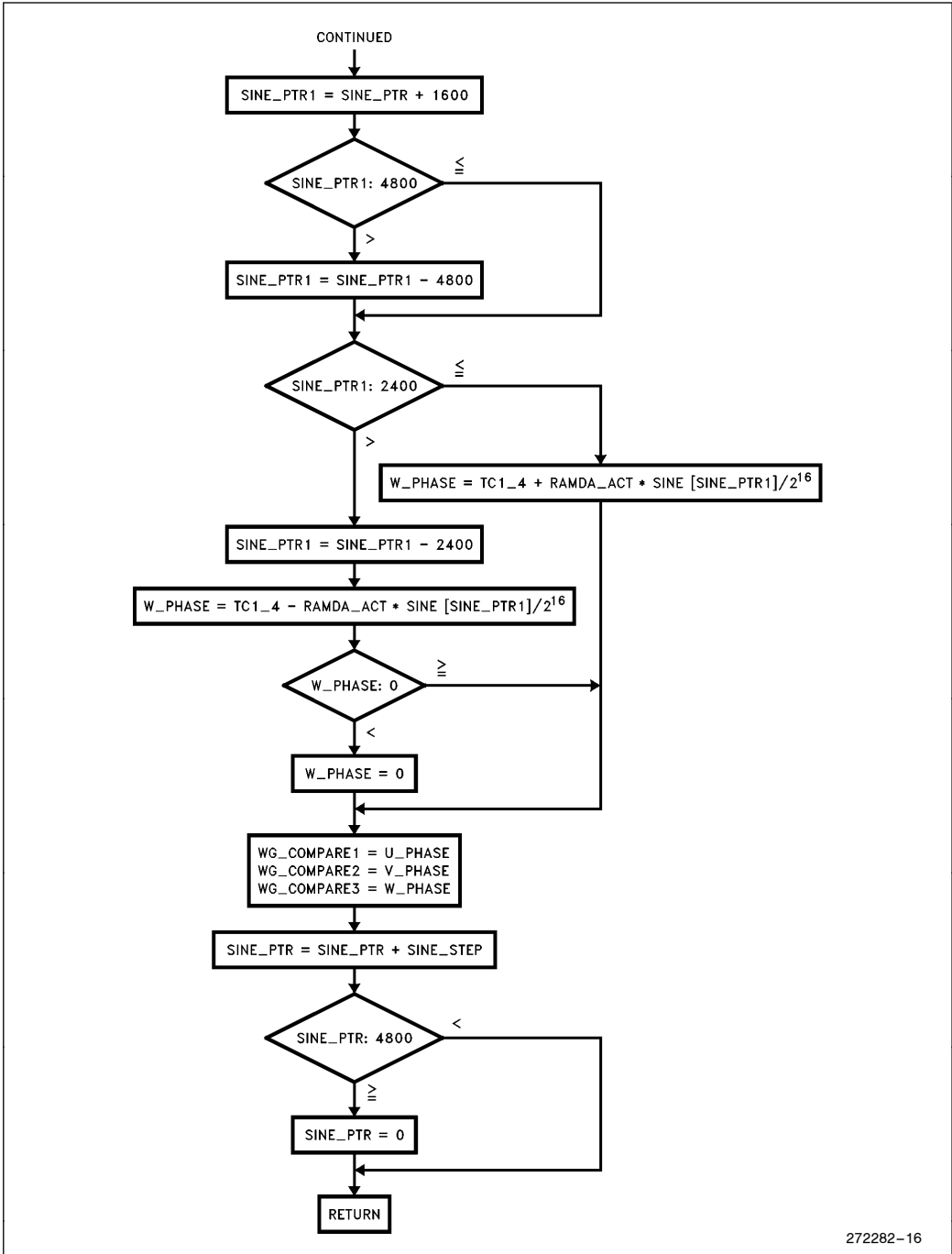


Figure 7-8. Flow Chart—3-Phase A.C. Induction Motor WG Interrupt Routine



272282-16

Figure 7-8. Flow Chart—3-Phase A.C. Induction Motor WG Interrupt Routine (Continued)

8.0 3-PHASE DC BRUSHLESS MOTOR CONTROL PROGRAM SAMPLE

8.1 Introduction

The following program example shows how the WFG can be used to generate a 3-phase output suitable for driving a DC brushless motor. Figure 8-1 shows a high-level view of how the 8XC196MC/MD will drive each phase of the motor. Each phase of the motor is driven from a complimentary output power driver, with Q1 driven from the WGx output, and Q2 driven by the WGx# outputs.

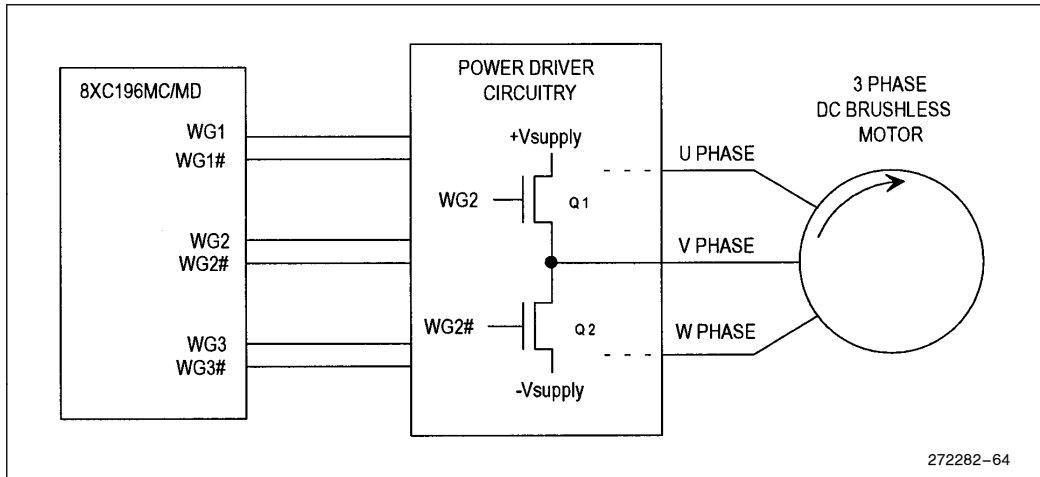


Figure 8-1. 3-Phase DC Brushless Motor System

In this example, the PWM carry frequency is 20 kHz. The modulation frequency is adjustable from 6 Hz to 200 Hz, and the duty cycle of the PWM controls the amount of motor torque. A look-up table establishes the relationship between modulation frequency and duty cycle of the PWM.

In this example, the drive signals for the three motor phases U, V, and W are divided into 6 distinct segments, named PHASE0 through PHASE5. The motor speed is determined by the time duration of these phases, and is controlled by an EPA timer.

This example is an “open loop” speed control, with no speed sensor feedback. The duration of the phase segments is determined by reading a DIP switch and converting this value to a time value loaded into the EPA timer. Each EPA interrupt sets up the duration of the next phase segment.

8.1.1 DRIVE WAVEFORMS

The output drive waveforms for the 3-phase DC brushless motor are illustrated in Figure 8-2. There are several algorithms for driving DC brushless motors; this example shows only one possible method. Note that each waveform represents $WG_x-WG_x\#$. A positive value indicates that WG_x is asserted, while a negative value indicates that $WG_x\#$ is asserted. A value of 0 indicates that both outputs are deasserted. This is a convenient way to display this information.

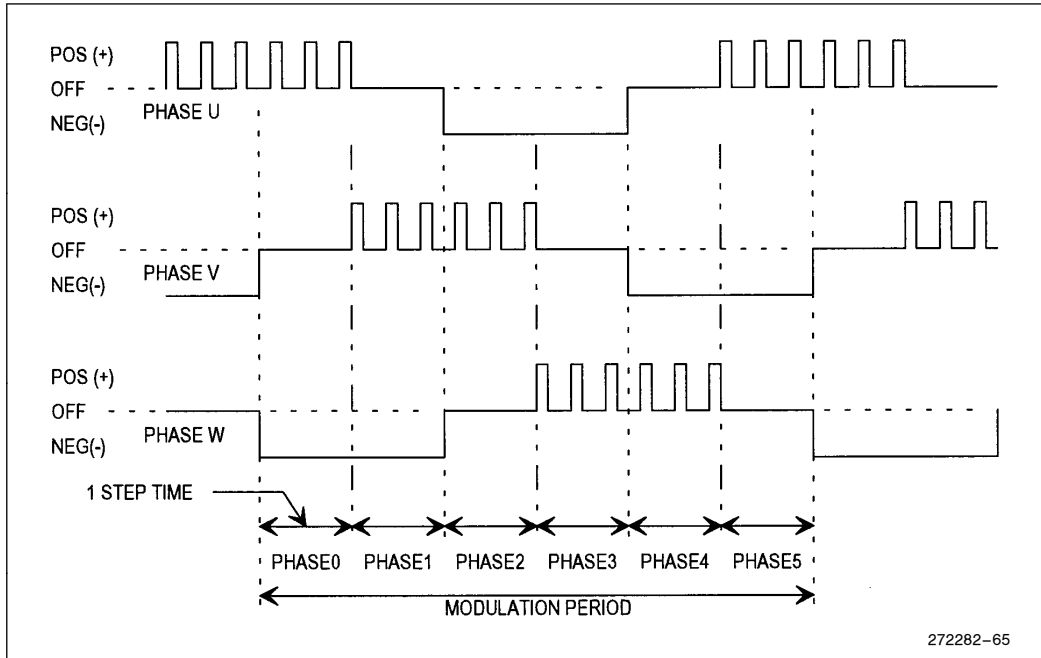


Figure 8-2. DC Brushless Phase Drive Waveforms

Each cycle of the drive waveform is divided into 6 unique phases, labeled PHASE0 through PHASE5. These should not be confused with the 3 phases U, V and W which are the motor windings. The pulse burst on each output is a fixed duty cycle, and creates an average positive DC level during the burst.

Table 8-1 illustrates that state of each output as the 6 phases are stepped through. An “On” means that the transistor connected to the respective output is on, and conversely “Off” means it is off. Note that the sequencing of the outputs is such that there is no possibility that both transistors can be on at the same time. Thus, no dead-time is necessary in this example.

Table 8-1. WFG Output Sequencing

Phase	WG1 (U)	WG1# (U#)	WG2 (V)	WG2# (V#)	WG3 (W)	WG3# (W#)	WG_OUT
0	WFG	Off	Off	Off	Off	On	2112H
1	Off	Off	WFG	Off	Off	On	2218H
2	Off	On	WFG	Off	Off	Off	2209H
3	Off	On	Off	Off	WFG	Off	2421H
4	Off	Off	Off	On	WFG	Off	2424H
5	WFG	Off	Off	On	Off	Off	2106H

In this example the outputs are “On” when the respective output is low. This is determined by the OP bits loaded into WG_OUT. Refer to Section 6.9 for more information on the output configurations.

After the PWMs are integrated by a motor or transformer (typically an excellent low-pass filter), they will approximate a trapezoidal waveform, illustrated in Figure 8-3.

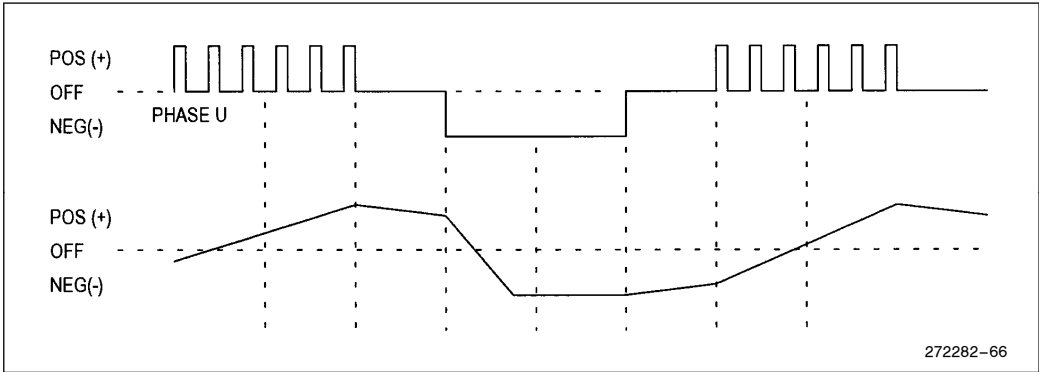


Figure 8-3. WFG Output Waveform after Low-Pass Filter

8.1.2 FREQUENCY-TO-VOLTAGE LOOK-UP TABLE

The graph in Figure 8-4 shows the current vs. speed characteristic used in this example. Note that maximum current amplitude is applied at minimum speed.

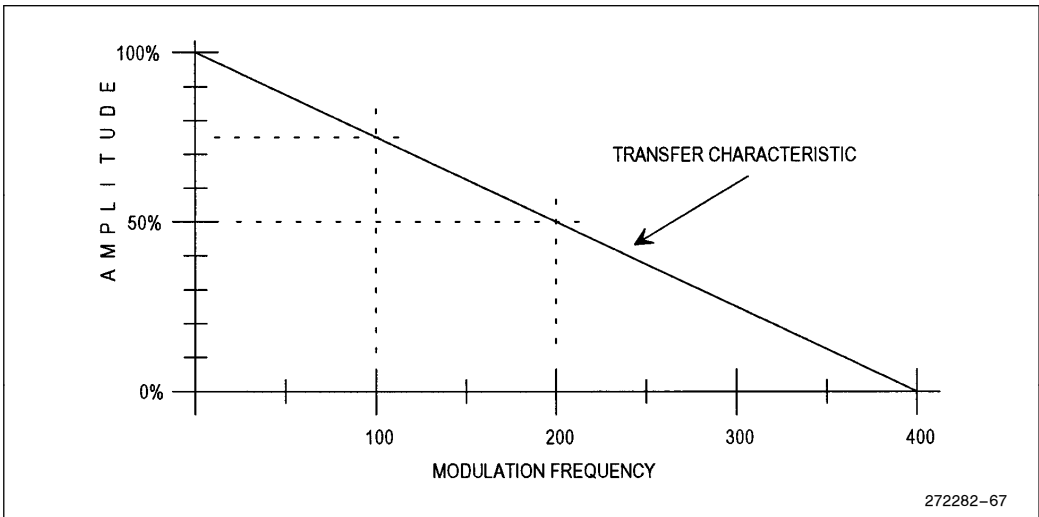


Figure 8-4. Amplitude vs Modulation Frequency Function

A lookup table is used to implement the transfer characteristic in Figure 8-4 (called AMP in the program listing). The table values are normalized to 100% = 65535 (0FFFFH), and entries are in 0.25 Hz intervals from 0 Hz–200 Hz. Since the frequency is software limited to 200 Hz, the table stops at 200 Hz (801 entries, 1602 bytes).

This table is read to determine the amplitude for each modulation frequency, and the PWM duty cycle is calculated in the *set amplitude* routine using the following formula:

$$\text{WG_COMP} = \frac{\text{AMP} \times \text{WG_RELOAD}}{65,536}$$

where

WG_COMP = the value to load into the phase compare register (variable name AMPLITUDE)

AMP = value from table

WG_RELOAD = carrier period (variable name PI)

AMP × WG_RELOAD yields a 32-bit result. By using only the most significant word of the result, the division by 65,536 is accomplished.

An EPA capture/compare register is set up to generate an interrupt for each of the 6 phases of the modulation period. This interrupt routine checks to see which phase needs to be serviced, updates that output register WG_OUT (selecting the new output pin configuration), and loads the WG_COMP registers. Note that this example uses WFG mode 3, which updates all the WG registers when the EPA PFE bit is set. See Section 6.2 and 6.6 for further explanation.

The time between EPA interrupts is called the step time (variable name STEP), and is $\frac{1}{6}$ of the modulation period (due to the 6 phases).

8.2 Detailed Program Description

8.2.1 CONSTANT DECLARATIONS (Lines 1–27)

Lines 5–8 define some of the operating conditions for the program. Minimum and maximum frequency limits are established and the carrier frequency/period is defined.

Lines 16–27 define register variables that are used later in the program.

8.2.2 INTERRUPT VECTORS AND CCB (Lines 28–48)

Lines 35, 39 and 40 fill in the interrupt vector table with the locations of the CAPCOM0_INT, WG_INT and EXT_INT interrupt routines. Only conventional interrupts are used in this program example, no PTS routines.

Lines 47 and 48 define the chip configuration bytes, CCB and CCB1. These need to be configured for the particular system that this program is run on.

8.2.3 MAIN PROGRAM INITIALIZATION (Lines 49–97)

Lines 54–57 define the program starting location (2080H), disable interrupts, and set up the stack at 0200H.

Lines 61 and 62 unmask the CAPCOMP0, PI and EXTINT interrupts. Since the PI interrupt is a shared interrupt, line 65 unmask the WG interrupt bit in PI_MASK.

Lines 69–75 initializes the variable values, and sets the initial modulation frequency to the minimum value.

Lines 76–78 calculate the initial amplitude using the look-up table.

Line 79 initializes the phase register to phase0.

Lines 81–82 loads the carrier period into the WG__RELOAD register and enables the interrupts.

Line 83 sets the interrupt pending bit for the PI interrupt, which results in an immediate call to the WG__INT interrupt routine. This routine checks which phase is active (0–5) and loads the compare register for that channel. Note that WG__COUNT has not yet started, so the values are transferred directly from the buffer to the compare register.

Lines 84–90 initialize the WFG output, protection, and control registers. The WFG counter is then started. PWM output will start at this time.

Line 91 again sets the interrupt pending bit for the PI interrupt, which results in an immediate call to the WG__INT interrupt routine. This causes the compare **buffer** register to be loaded with the **next** set of values, since WG__COUNT is now running.

Lines 93–96 set up the EPA CAPCOMP0 channel to interrupt at the end of the first phase. The compare mode with Peripheral Function Enable (PFE) is used.

8.2.4 POLLING ROUTINE (Lines 98–134)

Lines 107–120 read a value input on port 3, compare it against the FMIN and FMAX values and check that it is within FMIN and FMAX bounds. The value is then stored in FCOMMAND, and output on port 4, which is connected to some LED's as a monitor. FCOMMAND is used by the following code to set-up the step-size the pointers take through the AMP look-up table. Note that due to using port 4 this routine must be modified to run on any system which uses external memory!

Lines 124 and 125 uses FCOMMAND to form a pointer and get the amplitude from the AMP look-up table. Line 126 disables the interrupts, to protect the following code from unintended variable modification.

Lines 127–128 calculates the value AMPLITUDE, which is later loaded into WG__COMP establishing the duty cycle of the PWM.

Lines 129–132 calculate the value of the STEP variable from the FCOMMAND value, establishing the modulation frequency.

Lines 133–134 re-enable the interrupts, and loop back to line 107, where port 3 is read again, starting the update process again.

8.2.5 WG INTERRUPT ROUTINE (Lines 135–173)

The WG__INT interrupt routine is entered every carrier period. Lines 141–152 determine which of the 6 phases is being serviced, and branches to the appropriate routine.

Lines 153–173 are the routines for the 6 phases. Each routine updates its corresponding WG__COMP register, and returns.

8.2.6 CAPCOMP0 INTERRUPT ROUTINE (Lines 174–225)

The CAPCOMP0 interrupt routine is called each time the EPA CAPCOMP0 channel times out, at the end of each of the 6 phases. Its purpose is to reassign the output pins as required, and to update the PWM duty cycle.

Lines 180–191 determine which of the 6 phases is active, and branches to the appropriate routine.

Lines 192–218 service the active phase. First, the WG__OUT register is loaded with the new output configuration. Next, the WG__COMP registers are updated in the following manner:

```
if PHASE0, update WG__COMP1 only
if PHASE1, update WG__COMP1 and WG__COMP2
if PHASE2, update WG__COMP2 only
if PHASE3, update WG__COMP2 and WG__COMP3
if PHASE4, update WG__COMP3 only
if PHASE5, update WG__COMP3 and WG__COMP1
```

Note that if the phase interrupt is during the “center” of a PWM burst, only that WG__COMP register is updated. If the phase interrupt is at the end of old/beginning of new PWM, both the old and new WG__COMP registers are updated.

Line 220 increments the phase counter, and line 223 loads the time for the next CAPCOMP0 interrupt (1 step).

Lines 224–225 restore the CPU flags and return to the polling loop.

8.2.7 EXTINT INTERRUPT ROUTINE (Lines 226–237)

The EXTINT interrupt routine shown here is just a “hook” for some interrupt code that the user would use to shut-down the device response to some external condition.

8.2.8 AMPLITUDE LOOK-UP TABLE (Lines 244–1044)

The amplitude look-up table is discussed in Section 8.1.2. This table consists of 800 word values representing the amplitude vs. modulation frequency function. These are input as an include file, file name AMP.INC.

8.3 Top 4 Issues for the D.C. Brushless Motor Control Example

1. WG__INT execution time must be less than the carrier period.
2. The AMP table should be optimized for the motor being used. Simple amplitude-to-frequency relationships can be calculated instead of using a table.
3. The WG registers are initialized twice: The first time with the counter not running, which loads the registers directly, and the second time immediately after the counter is started, which loads the buffers.
4. The outputs are initialized to one phase **behind** what the initial value of the PHASE variable is.

8.4 Program Example

```

1  ;*****
2  ; WAVEFORM GENERATOR CARRIER FREQUENCY
3  ;*****
4  ;
5  CARRIER_FREQ EQU          20000          ;20KHz
6  PERIOD        EQU          400           ;(1/CARRIER_FREQ)/0.125E-6
7  MIN_FREQ      EQU          6            ;minimum inverter frequency
8  MAX_FREQ      EQU          200          ;maximum inverter frequency
9  ;
10 ;*****
11 ;USER DEFINED REGISTERS
12 ;*****
13 ;
14 RSEG          AT            1AH
15
16 STEP:         DSL           1
17 AMPLITUDE:    DSL           1
18 TC:           DSW           1
19 FMAX:         DSW           1
20 FMIN:         DSW           1
21 AMP_PTR:      DSW           1
22 FCOMMAND:     DSW           1
23 P_TEMP:       DSB           1
24 TEMP1:        DSB           1
25 PHASE:        DSB           1
26
27 FCOMMAND_L    EQU           (FCOMMAND):BYTE
28 ;
29 ;*****
30 ; INTERRUPT VECTOR TABLE
31 ;*****
32 ;
33 CSEG          AT            2004H
34
35              DCW            CAPCOMPO_INT ;INT 02
36
37 CSEG          AT            203AH
38
39              DCW            WG_INT      ;INT 13
40              DCW            EXT_INT     ;INT 14
41
42 ;*****
43 ; CHIP CONFIGURATION BYTES
44 ;*****

```

8.4 Program Example (Continued)

```

45  CSEG  AT          2018H
46  ;
47          DCW          20CFH          ;CCB
48          DCW          20DCH          ;CCB1
49  ;
50  ;*****
51  ; MAIN ROUTINE
52  ;*****
53  ;
54  CSEG  AT          2080H
55  MAIN_START:
56          DI                      ;disable interrupt
57          LD          SP,#0200H      ;set-up stack pointer
58  ;
59  ;Clear interrupt mask register
60  ;
61          LDB          INT_MASK,#04H  ;enable CAPCOMPO interrupt
62          LDB          INT_MASK1,#60H ;enable PI and EXTINT
63          ;                    ;interrupt sources
64          LDB          WSR,#3EH      ;map 64 bytes to 1F80H-1FBFH
65          LDB          PI_MASK_WO,#10H ;set-up peripheral interrupt mask reg.
66  ;
67  ;Initialize WG
68  ;
69          LD          TC,#PERIOD      ;set PWM carrier period
70          LD          FMIN,#MIN_FREQ  ;set minimum inverter frequency
71          LD          FMAX,#MAX_FREQ  ; maximum inverter frequency
72          LD          STEP,CLOCK_I    ;set initial inverter frequency
73          LD          STEP+2,CLOCK_I+2 ; init. freq.=CLOCK_I/FMIN
74          MULUB       FCOMMAND,FMIN,#6 ; init. STEP=init. freq./6
75          DIVU        STEP,FCOMMAND   ; (6=phase number)
76          MULUB       AMP_PTR,FMIN,#8 ;set initial amplitude
77          MULU        AMPLITUDE,TC,AMP[AMP_PTR]
78          LD          AMPLITUDE,AMPLITUDE+2
79          CLRB        PHASE           ;reset phase counter
80          LDB          WSR,#3FH      ;map 64 bytes to 1FC0H-1FFFH
81          LD          WG_RELOAD_WO,TC ;set PWM carrier period
82          EI          ;enable interrupt
83          LDB          INT_PEND1,#20H ;set next values to WG_COMPAREN
84          LD          WG_OUT_WO,#0106H ;0000 0001 0000 0110B
85          ;                    ;set WG output register to phase 5
86          LD          WG_OUT_WO,#2106H ;0010 0001 0000 0110B
87          ;                    ;set SYNC bit
88          LDB          WG_PROTECT_WO,#05H ;enable WG output,sampling circuit
89          ;                    ;and rising edge trigger
90          LD          WG_CONT_WO,#3400H ;Mode3,td=0.0us,start counter
91          LDB          INT_PEND1,#20H ;set next values to WG_COMPAREN
92          LDB          WSR,#3DH      ;map 64 bytes to 1F40H-1F7FH
93          LDB          T1CONTROL_WO,#0C1H ;timer 1 enable, up count, clock
94          ;                    ;internal, pre-scale=div 2
95          LDB          CAPCOMPO_CON_WO,#44H ;comparator, peripheral enable
96          ADD         CAPCOMPO_TIME_WO,TIMER1_WO,STEP
97          ;                    ;set interrupt period
98  ;
    
```

8.4 Program Example (Continued)

```

99 ;*****
100 ; POLLING ROUTINE
101 ;*****
102 ;
103 LOOP:
104 ;
105 ;load modulation frequency
106 ;
107     CLR     FCOMMAND           ;clear FCOMMAND register
108     LDB     FCOMMAND_L,P3PIN[0] ;load modulation frequency from DIP SW
109     CMP     FCOMMAND,FMIN      ;check lowest frequency
110     BGE     NEXT1             ;if it's lower than FMIN
111     LD      FCOMMAND,FMIN      ;then set to FMIN
112     BR      NEXT2
113 NEXT1:
114     CMP     FCOMMAND,FMAX      ;check highest frequency
115     BLE     NEXT2             ;if it's higher than FMAX
116     LD      FCOMMAND,FMAX      ;then set to FMAX
117 NEXT2:
118     LDB     TEMP1,FCOMMAND      ;display frequency command on LED
119     NOTB    TEMP1
120     STB     TEMP1,P4REG[0]
121 ;
122 ;Set Amplitude
123 ;
124     LD      AMP_PTR,FCOMMAND    ;AMP_PTR=FCOMMAND/0.25*2
125     SHL     AMP_PTR,#3         ;due to 0.25Hz step & word access
126     DI                                     ;protect from individual modification
127     MULU    AMPLITUDE,TC,AMP[AMP_PTR] ;refer to AMP table
128     LD      AMPLITUDE,AMPLITUDE+2
129     LD      STEP,CLOCK_I       ;set inverter frequency
130     LD      STEP+2,CLOCK_I+2
131     MULUB   FCOMMAND,#6
132     DIVU    STEP,FCOMMAND
133     EI
134     SJMP   LOOP
135 ;
136 ;*****
137 ; WG_COUNTER INTERRUPT ROUTINE
138 ;*****
139 ;
140 WG_INT:
141     PUSHA
142     LDB     WSR,#3FH           ;save PSW,INT_MASK,INT_MASK1,WSR
143     CMPB   PHASE,#0           ;map 64 bytes to 1FC0H-1FFFH
144     BE     PHASE0_WG         ;sieve by phase
145     CMPB   PHASE,#1
146     BE     PHASE1_WG
147     CMPB   PHASE,#2
148     BE     PHASE2_WG
149     CMPB   PHASE,#3
150     BE     PHASE3_WG
151     CMPB   PHASE,#4
152     BE     PHASE4_WG

```

8.4 Program Example (Continued)

```

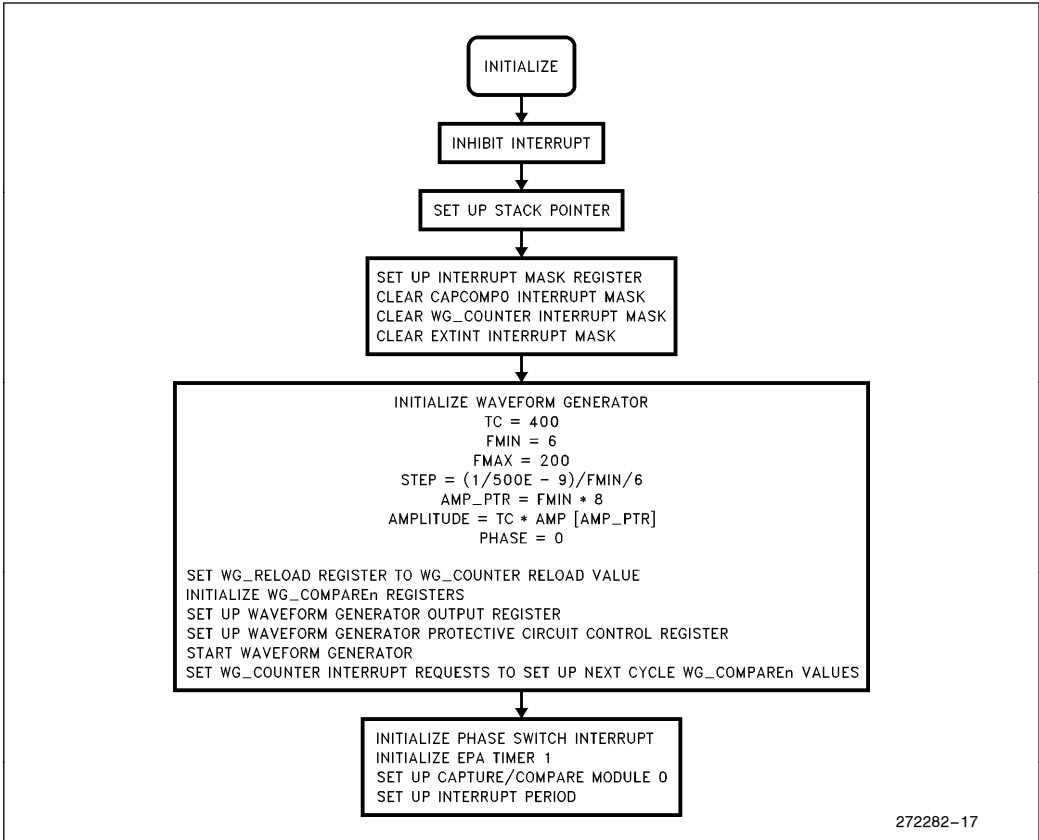
153 PHASE5_WG:                ;if phase=5 then update WG_COMP3
154     LD      WG_COMP3_W0,AMPLITUDE
155     BR      WG_INT_END
156 PHASE0_WG:                ;if phase=0 then update WG_COMP3
157     LD      WG_COMP3_W0,AMPLITUDE
158     BR      WG_INT_END
159 PHASE1_WG:                ;if phase=1 then update WG_COMP1
160     LD      WG_COMP1_W0,AMPLITUDE
161     BR      WG_INT_END
162 PHASE2_WG:                ;if phase=2 then update WG_COMP1
163     LD      WG_COMP1_W0,AMPLITUDE
164     BR      WG_INT_END
165 PHASE3_WG:                ;if phase=3 then update WG_COMP2
166     LD      WG_COMP2_W0,AMPLITUDE
167     BR      WG_INT_END
168 PHASE4_WG:                ;if phase=4 then update WG_COMP2
169     LD      WG_COMP2_W0,AMPLITUDE
170     BR      WG_INT_END
171 WG_INT_END:
172     POPA
173     RET                    ;load PSW,INT_MASK,INT_MASK1,WSR
174 ;                          ;return to main loop
175 ;*****
176 ; CAPCOMPO INTERRUPT ROUTINE
177 ;*****
178 ;
179 CAPCOMPO_INT:
180     PUSHA                    ;save PSW,INT_MASK,INT_MASK1,WSR
181     LDB     WSR,#3FH         ;map 64 bytes to 1FCOH-1FFFH
182     CMPB   PHASE,#0         ;sieve by phase
183     BE     PHASE0_CCMO
184     CMPB   PHASE,#1
185     BE     PHASE1_CCMO
186     CMPB   PHASE,#2
187     BE     PHASE2_CCMO
188     CMPB   PHASE,#3
189     BE     PHASE3_CCMO
190     CMPB   PHASE,#4
191     BE     PHASE4_CCMO
192 PHASE5_CCMO:                ;if phase=5 then do this
193     LD      WG_OUT_W0,#2106H ;0010 0001 0000 0110B
194     LD      WG_COMP3_W0,AMPLITUDE ;update WG_COMP3
195     LD      WG_COMP1_W0,AMPLITUDE ;update WG_COMP1
196     CLR    PHASE            ;clear phase
197     BR     CAPCOMPO_INT_END
198 PHASE0_CCMO:                ;if phase=0 then do this
199     LD      WG_OUT_W0,#2112H ;0010 0001 0001 0010B
200     LD      WG_COMP1_W0,AMPLITUDE ;update WG_COMP1
201     BR     CAPCOMPL_1
    
```

8.4 Program Example (Continued)

```

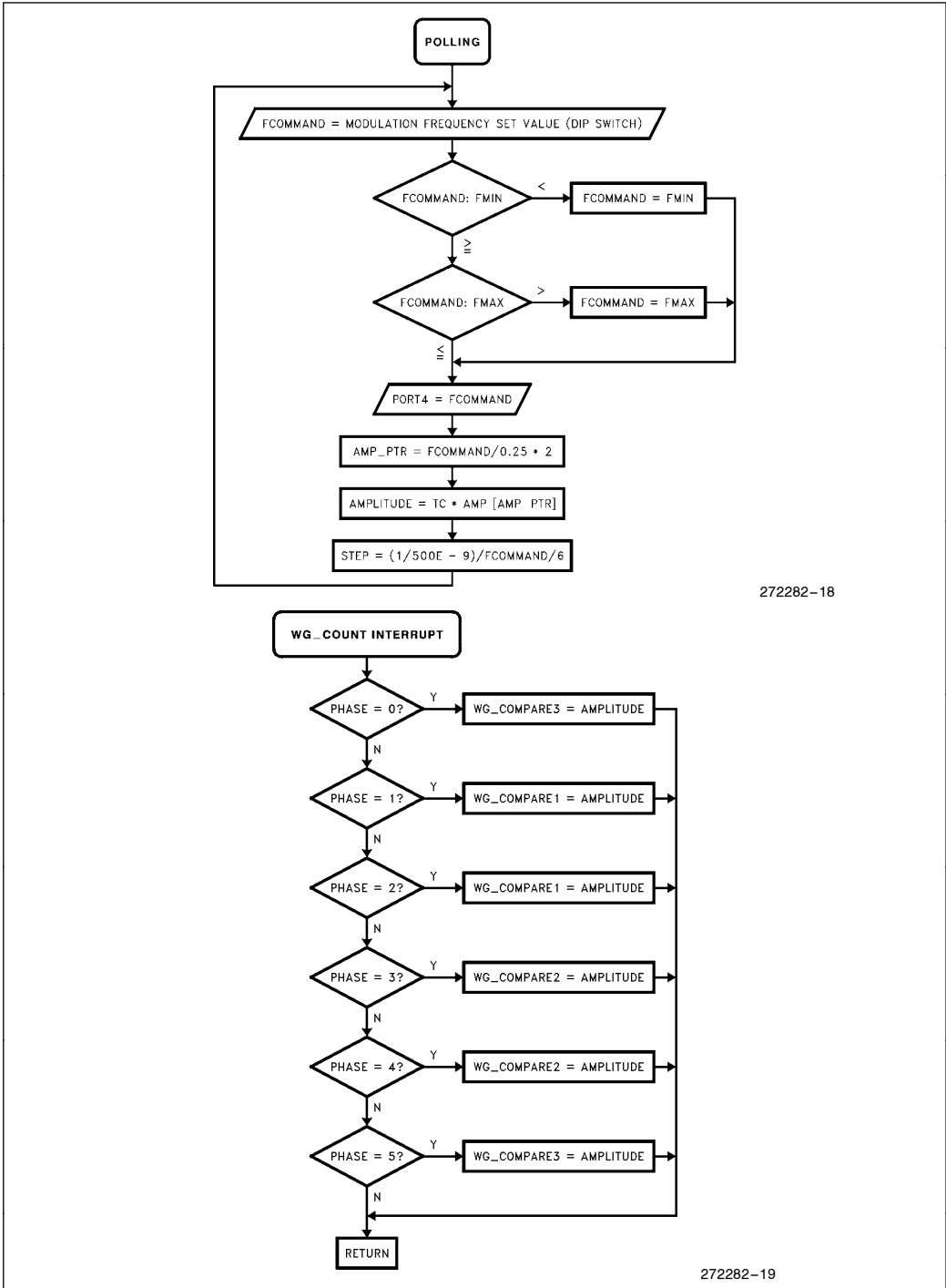
202 PHASE1_CCM0:                ;if phase=1 then do this
203     LD     WG_OUT_W0,#2218H  ;0010 0010 0001 1000B
204     LD     WG_COMP1_W0,AMPLITUDE ;update WG_COMP1
205     LD     WG_COMP2_W0,AMPLITUDE ;update WG_COMP2
206     BR     CAPCOMPO_1
207 PHASE2_CCM0:                ;if phase=2 then do this
208     LD     WG_OUT_W0,#2209H  ;0010 0010 0000 1001B
209     LD     WG_COMP2_W0,AMPLITUDE ;update WG_COMP2
210     BR     CAPCOMPO_1
211 PHASE3_CCM0:                ;if phase=3 then do this
212     LD     WG_OUT_W0,#2421H  ;0010 0100 0010 0001B
213     LD     WG_COMP2_W0,AMPLITUDE ;update WG_COMP2
214     LD     WG_COMP3_W0,AMPLITUDE ;update WG_COMP3
215     BR     CAPCOMPO_1
216 PHASE4_CCM0:                ;if phase=4 then do this
217     LD     WG_OUT_W0,#2424H  ;0010 0100 0010 0100B
218     LD     WG_COMP3_W0,AMPLITUDE ;update WG_COMP3
219 CAPCOMPO_1:
220     INCB   PHASE              ;increment phase
221 CAPCOMPO_INT_END:
222     LDB   WSR,#3DH           ;map 64 bytes to 1F40H-1F7FH
223     ADD   CAPCOMPO_TIME_W0,STEP;set next interrupt period
224     POPA  PSW,INT_MASK,INT_MASK1,WSR ;load PSW,INT_MASK,INT_MASK1,WSR
225     RET                       ;return to main loop
226 ;
227 ;*****
228 ; EXTERNAL INTERRUPT ROUTINE
229 ;*****
230 ;
231 EXT_INT:
232     PUSHA                      ;save PSW,INT_MASK,INT_MASK1,WSR
233     -
234     -
235     -
236     POPA                      ;load PSW,INT_MASK,INT_MASK1,WSR
237     RET                       ;return to main loop
238 ;
239 ;*****
240 ; DATA TABLE
241 ;*****
242 ;
243 AMP:                                ;value by word (0.25Hz step)
-                                     ; -
268     DCW   64551              ;Amplitude(6Hz) 98.5%
-                                     ; -
644     DCW   49151              ;Amplitude(100Hz) 75%
-                                     ; -
1044    DCW   32768              ;Amplitude(200Hz) 50%
1045
1046 CLOCK_I:
1047     DCL   2000000             ;1/5E-6 (EPA timer clock)
1048
1049 END

```



272282-17

Figure 8-5. Flow Chart—3-Phase D.C. Brushless Motor Initialization



272282-18

272282-19

Figure 8-6. Flow Chart—3-Phase D.C. Brushless Motor WG_COUNT Interrupt

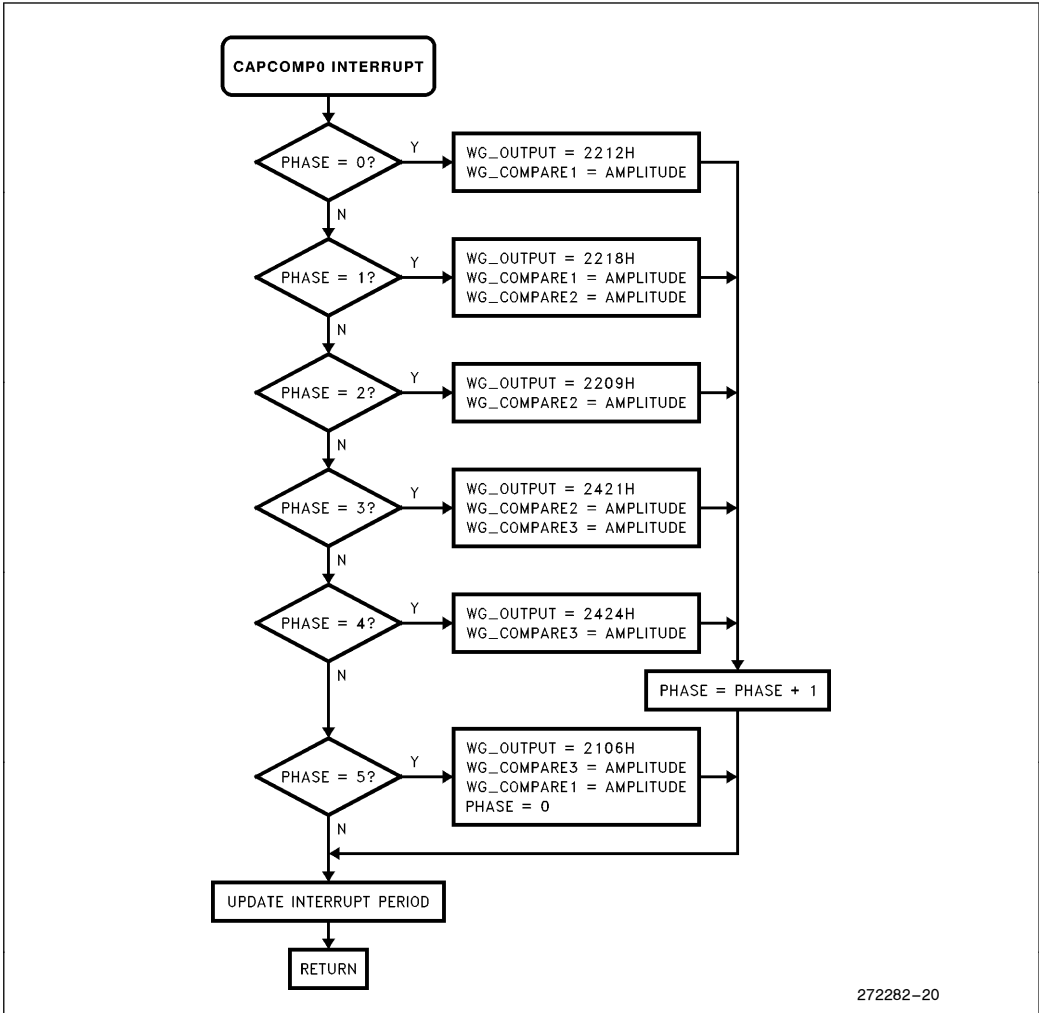


Figure 8-7. Flow Chart—3-Phase D.C. Brushless Motor EPA Interrupt

272282-20

9.0 QUADRI-PHASE STEPPING MOTOR CONTROL PROGRAM SAMPLE

9.1 Introduction

The following program example shows how the WFG can be used to generate a 4-phase output suitable for driving stepper motor. Figure 9-1 shows a high-level view of how the 8XC196MC/MD will drive each phase of the motor. Note that this is a unipolar design requiring only a single power supply.

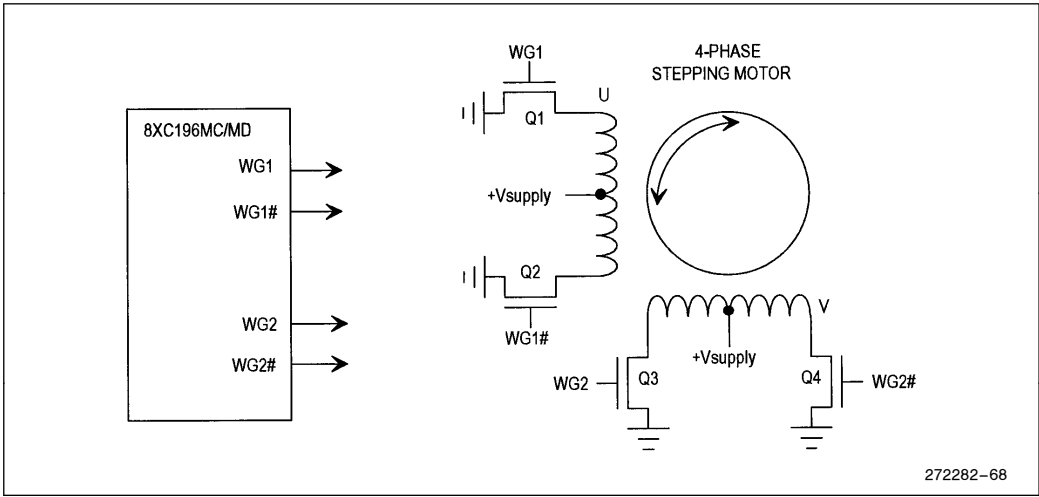


Figure 9-1. 4-Phase Stepper Motor System

In this example, the PWM carrier frequency is 20 kHz. The modulation frequency is adjustable from 6 Hz to 200 Hz, and the duty cycle of the PWM controls the amount of motor torque. A look-up table establishes the relationship between modulation frequency and duty cycle of the PWM.

9.1.1 DRIVE WAVEFORMS

The output drive waveform for the 4-phase stepper motor are illustrated in Figure 9-2. There are several algorithms for driving stepper motors; this example shows only one possible method.

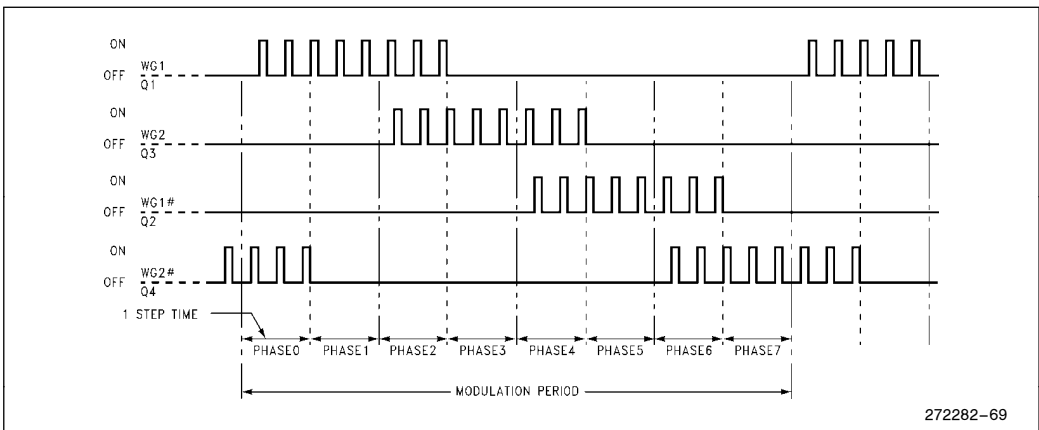


Figure 9-2. 4-Phase Stepper Drive Waveforms

Each cycle of the drive waveform is divided into 8 unique phases, labeled PHASE0 through PHASE7. These should not be confused with the 2 output phases U and V which drive the motor windings. The pulse burst on each output is a fixed duty cycle, and creates an average positive DC level during the burst.

Table 9-1 illustrates the states of each output as the 8 phases are stepped through. An “On” means that the transistor connected to the respective output is on, and conversely “Off” means it is off. Note that with the unipolar drive dead-time is not required.

Table 9-1. WFG Output Sequencing

Phase	WG1 (U)	WG1 # (U #)	WG2 (V)	WG2 # (V #)	WG_OUT
0	WFG	Off	Off	WFG	2306H
1	WFG	Off	Off	Off	2302H
2	WFG	Off	WFG	Off	230AH
3	Off	Off	WFG	Off	2308H
4	Off	WFG	WFG	Off	2309H
5	Off	WFG	Off	Off	2301H
6	Off	WFG	Off	WFG	2305H
7	Off	Off	Off	WFG	2304H

In this example the outputs are “On” when the respective output is low. This is determined by the OP bits loaded into WG_OUT. Refer to Section 6.9 for more information on the output configurations.

Figure 9-3 illustrates the averaged voltage output waveforms seen at the motor. Note that as the modulation frequency is increased, the average voltage applied to the motor winding is decreased. Also note that when the frequency is changed upward, the last pulse at the old frequency is cut short, so a smooth transition to the higher frequency is made. Note that the PWM duty cycle changes immediately when the frequency is changed, such that the average voltage applied to all driven windings is immediately reduced.

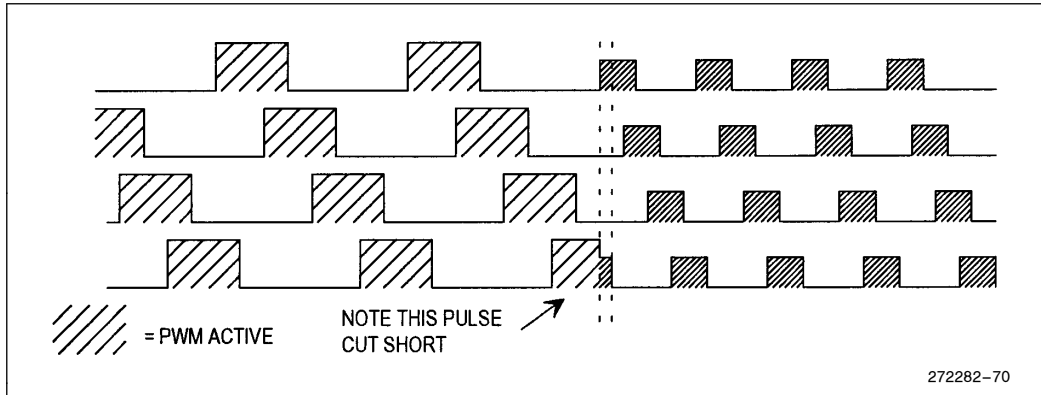


Figure 9-3. Average Voltage Output Waveform at Motor

The graph in Figure 9-4 shows the current vs speed characteristics used in this example. Note that maximum amplitude (duty cycle) is applied at minimum speed.

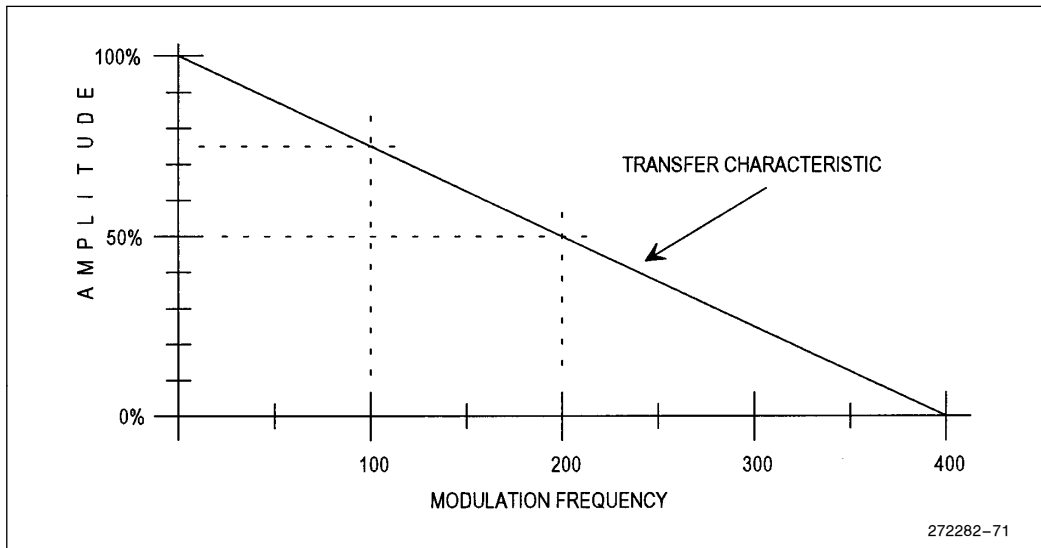


Figure 9-4. Amplitude vs Modulation Frequency Function

A look-up table is used to implement the transfer characteristic in Figure 9-4 (called AMP in the program listing). The table values are normalized to 100% = 65535 (0FFFFH), and entries are in 0.25 Hz intervals from 0 Hz–200 Hz. Since the frequency is software limited to 200 Hz, the table stops at 200 Hz (801 entries, 1602 bytes).

This table is read to determine the amplitude for each modulation frequency, and the PWM duty cycle is calculated in the *set amplitude* routine using the following formula:

$$WG_COMP = \frac{AMP \times WG_RELOAD}{65,536}$$

where

- WG_COMP = the value to load into the phase compare register (variable name AMPLITUDE)
- AMP = value from table
- WG_RELOAD = carrier period (variable name PI)

AMP × WG_RELOAD yields a 32-bit result. By using only the most significant word of the result, the division by 65,536 is accomplished.

An EPA capture/compare register is set up to generate an interrupt for each of the 8 phases of the modulation period. This interrupt routine checks to see which phase needs to be serviced, updates the output register WG_OUT (selecting the new output pin configuration), and loads the WG_COMP registers.

The time between EPA interrupts is called the step time (variable name STEP), and is 1/8 of the modulation period (due to the 8 phases).

9.2 Detailed Program Description

Note that this program is very similar to the DC brushless motor program in Section 8.

9.2.1 CONSTANT DECLARATIONS (Lines 1–29)

Lines 5–9 define some of the operating conditions for the program. Minimum and maximum frequency limits are established, the carrier frequency/period and the clock frequency for the EPA channel are defined.

Lines 16–29 define register variables that are used later in the program.

9.2.2 INTERRUPT VECTORS AND CCB (Lines 30–49)

Lines 37 and 41 fill in the interrupt vector table with the locations of the CAPCOMP0_INT and EXT_INT interrupt routines. Only conventional interrupts are used in this program example, no PTS routines.

Lines 48 and 49 define the chip configuration bytes, CCB and CCB1. These need to be configured for the particular system that this program is run on.

9.2.3 MAIN PROGRAM INITIALIZATION (Lines 50–95)

Lines 55–58 define the program starting location (2080H), disable interrupts, and set up the stack at 0200H.

Lines 62 and 63 unmask the CAPCOMP0 and EXTINT interrupts. Line 65 masks the WG interrupt bit in PI_MASK, as this is not used.

Lines 69–74 initializes the variable values, and sets the initial modulation frequency to the minimum value.

Lines 75–77 calculate the initial amplitude using the look-up table.

Line 78 initializes the phase register to phase1.

Lines 80–81 loads the carrier period into the WG_RELOAD register and enables the interrupts.

Lines 84–89 initialize the WFG output, protection, and control registers. Note that WG_COUNT has not yet started, so the values are transferred directly to the compare register.

Line 90 starts the WFG counter. PWM output will start at this time.

Lines 92–95 set up the EPA CAPCOMP0 channel to interrupt at the end of the first phase. The compare only mode is used.

9.2.4 POLLING ROUTINE (Lines 96–134)

Lines 105–118 read a value input on port 3, compare it against the FMIN and FMAX values and check that it is within FMIN and FMAX bounds. The value is then stored in FCOMMAND, and output on port 4, which is connected to some LED's as a monitor. FCOMMAND is used by the following code to set-up the step-size the pointers take through the AMP look-up table. Note that this routine must be modified to run on any system which uses external memory!

Line 122 forms the pointer into the AMP table by multiplying FCOMMAND by 8. Line 123 then disables the interrupts to protect the following code from unintended variable modification.

Lines 124 and 125 gets the amplitude from the AMP look-up table.

Lines 126–128 calculate the value of the STEP variable from the FCOMMAND value, establishing the modulation frequency. This is later loaded into the EPA time register.

Lines 129–130 re-enable the interrupts, and loop back to line 105, where port 3 is read again, starting the update process again.

9.2.5 CAPCOMP0 INTERRUPT ROUTINE (Lines 131–186)

The CAPCOMP0 interrupt routine is called each time the EPA CAPCOMP0 channel times out, at the end of each of the 8 phases. Its purpose is to reassign the output pins as required, and to update the PWM duty cycle.

Lines 137–152 determine which of the 8 phases is active, and branches to the appropriate routine.

Lines 153–176 service the active phase. The WG_OUT register is loaded with the new output configuration according to the conditions below:

```

if PHASE0, update WG_OUT with 2306H
if PHASE1, update WG_OUT with 2302H
if PHASE2, update WG_OUT with 230AH
if PHASE3, update WG_OUT with 2308H
if PHASE4, update WG_OUT with 2309H
if PHASE5, update WG_OUT with 2301H
if PHASE6, update WG_OUT with 2305H
if PHASE7, update WG_OUT with 2304H and reset phase counter to 0.

```

Line 179 increments the phase counter, and lines 181–182 update the WG_COMP registers to the new duty cycle.

Line 184 loads the time for the next CAPCOMP0 interrupt (1 step).

Lines 185–186 restore the CPU flags and return to the polling loop.

9.2.6 EXTINT INTERRUPT ROUTINE (Lines 187–198)

The EXTINT interrupt routine shown here is just a “hook” for some interrupt code that the user would use to shut-down the device in response to some external condition.

9.2.7 AMPLITUDE LOOK-UP TABLE (Lines 199–1005)

The amplitude look-up table is discussed in Section 9.1.2. This table consists of 800 word values representing the amplitude vs modulation frequency function. These are input as an include file, file name APM.INC.

9.3 Top 3 Issues for the 4-Phase Stepper Motor Control Example

1. The AMP table should be optimized for the motor being used. Simple amplitude-to-frequency relationships can be calculated instead of using a table.
2. The WG registers are initialized twice: The first time with the counter not running, which loads the registers directly, and the second time immediately after the counter is started, which loads the buffers.
3. The outputs are initialized to one phase **behind** what the initial value of the phase value is.

9.4 Program Example

```

1  ;*****
2  ; WAVEFORM GENERATOR CARRIER FREQUENCY
3  ;*****
4  ;
5  CARRIER_FREQ EQU          20000          ;20KHz
6  PERIOD        EQU          400           ;(1/CARRIER_FREQ)/0.125E-6
7  CLOCK_P      EQU          62500         ;EPA clock frequency @ 1/64
8  MIN_FREQ     EQU          6             ;minimum pulse rate
9  MAX_FREQ     EQU          200           ;maximum pulse rate
10 ;
11 ;*****
12 ;USER DEFINED REGISTERS
13 ;*****
14 ;
15 RSEG          AT              LAH
16
17 STEP:         DSL             1
18 AMPLITUDE:   DSL             1
19 TC:          DSW             1
20 CLOCK_I:     DSW             1
21 FMAX:        DSW             1
22 FMIN:        DSW             1
23 AMP_PTR:     DSW             1
24 FCOMMAND:    DSW             1
25 P_TEMP:     DSB             1
26 TEMP1:      DSB             1
27 PHASE:      DSB             1
28
29 FCOMMAND_L   EQU              (FCOMMAND) :BYTE
30 ;
31 ;*****
32 ; INTERRUPT VECTOR TABLE
33 ;*****
34 ;
35 CSEG         AT              2004H
36
37             DCW              CAPCOMPO_INT ;INT 02
38
39 CSEG         AT              203CH
40
41             DCW              EXT_INT     ;INT 14
42
43 ;*****
44 ; CHIP CONFIGURATION BYTES
45 ;*****
46 CSEG         AT              2018H
47 ;
48             DCW              20CFH      ;CCB
49             DCW              20DCH      ;CCB1
50 ;
    
```

9.4 Program Example (Continued)

```

51 ;*****
52 ; MAIN ROUTINE
53 ;*****
54 ;
55 CSEG AT 2080H
56 MAIN_START:
57     DI ;disable interrupt
58     LD SP,#0200H ;set-up stack pointer
59 ;
60 ;Clear interrupt mask register
61 ;
62     LDB INT_MASK,#04H ;enable CAPCOMPO interrupt
63     LDB INT_MASK1,#40H ;enable EXTINT interrupt
64     LDB WSR,#3EH ;map 64 bytes to 1F80H-1FEFH
65     CLRBI PI_MASK_WO ;clear peripheral interrupt mask reg.
66 ;
67 ;Initialize WG
68 ;
69     LD TC,#PERIOD ;set PWM carrier period
70     LD FMIN,#MIN_FREQ ;set minimum pulse rate
71     LD FMAX,#MAX_FREQ ;set maximum pulse rate
72     LD STEP,#CLOCK_P ;set initial pulse rate
73     LD STEP+2,#0
74     DIVU STEP,FMIN
75     MULUB AMP_PTR,FMIN,#8 ;set initial amplitude
76     MULU AMPLITUDE,TC,AMP[AMP_PTR]
77     LD AMPLITUDE,AMPLITUDE+2
78     LDB PHASE,#1 ;initialize phase counter
79     LDB WSR,#3FH ;map 64 bytes to 1FC0H-1FFFH
80     LD WG_RELOAD_WO,TC ;set PWM carrier period
81     EI ;enable interrupt
82     LD WG_COMPL1_WO,AMPLITUDE ;set initial PWM duty cycle
83     LD WG_COMP2_WO,AMPLITUDE
84     LD WG_OUT_WO,#0306H ;0000 0011 0000 0110B
85     ;set WG output register to phase0
86     LD WG_OUT_WO,#2306H ;0010 0011 0000 0110B
87     ;set SYNC bit
88     LDB WG_PROTECT_WO,#05H ;enable WG output,sampling circuit
89     ;and rising edge trigger
90     LD WG_CON_WO,#2400H ;Mode2,td=0.0us,start counter
91     LDB WSR,#3DH ;map 64 bytes to 1F40H-1F7FH
92     LDB T1CONTROL_WO,#0C6H ;timer 1 enable, up count, clock
93     ;internal, pre-scale=div 64
94     LDB CAPCOMPO_CON_WO,#40H ;comparator only
95     ADD CAPCOMPO_TIME_WO,TIMER1_WO,STEP
96 ;
97 ;*****
98 ; POLLING ROUTINE
99 ;*****
100 ;
101 LOOP:
102 ;
103 ;load motor pulse rate
104 ;

```


9.4 Program Example (Continued)

```

105         CLR     FCOMMAND           ;clear FCOMMAND register
106         LDB     FCOMMAND_L,P3PIN[0] ;load pulse rate from DIP SW
107         CMP     FCOMMAND,FMIN      ;check lowest rate
108         BGE     NEXT1              ; if it's lower than FMIN
109         LD      FCOMMAND,FMIN      ; then set FMIN
110         BR      NEXT2
111 NEXT1:
112         CMP     FCOMMAND,FMAX      ;check highest rate
113         BLE     NEXT2              ; if it's higher than FMAX
114         LD      FCOMMAND,FMAX      ; then set FMAX
115 NEXT2:
116         LDB     TEMP1,FCOMMAND     ;display pulse rate on LED
117         NOTB    TEMP1
118         STB     TEMP1,P4REG[0]
119         ;
120         ;Set Amplitude
121         ;
122         MULUB   AMP_PTR,FCOMMAND,#8 ;set amplitude
123         DI      ;protect from individual modification
124         MULU    AMPLITUDE,TC,AMP[AMP_PTR] ;refer to AMP table
125         LD      AMPLITUDE,AMPLITUDE+2
126         LD      STEP,CLOCK_P       ;set pulse rate
127         LD      STEP+2,#0
128         DIVU    STEP,FCOMMAND
129         EI
130         SJMP   LOOP
131         ;
132         ;*****
133         ; CAPCOMPO INTERRUPT ROUTINE
134         ;*****
135         ;
136 CAPCOMPO_INT:
137         PUSHA
138         LDB     WSR,#3FH           ;save PSW,INT_MASK,INT_MASK1,WSR
139         CMPB    PHASE,#0           ;map 64 bytes to 1FCOH-1FFFH
140         BE      PHASE0_CCMO        ;sieve by phase
141         CMPB    PHASE,#1
142         BE      PHASE1_CCMO
143         CMPB    PHASE,#2
144         BE      PHASE2_CCMO
145         CMPB    PHASE,#3
146         BE      PHASE3_CCMO
147         CMPB    PHASE,#4
148         BE      PHASE4_CCMO
149         CMPB    PHASE,#5
150         BE      PHASE5_CCMO
151         CMPB    PHASE,#6
152         BE      PHASE6_CCMO
153 PHASE7_CCMO:
154         LD      WG_OUT_WO,#2304H   ;if phase=7 then do this
155         CLR     PHASE               ;0010 0011 0000 0100B
156         BR      CAPCOMPO_INT_END   ;clear phase
157 PHASE0_CCMO:
158         LD      WG_OUT_WO,#2306H   ;if phase=0 then do this
159         BR      CAPCOMPO_1         ;0010 0011 0000 0110B
160 PHASE1_CCMO:
161         LD      WG_OUT_WO,#2302H   ;if phase=1 then do this
162         BR      CAPCOMPO_1         ;0010 0011 0000 0010B

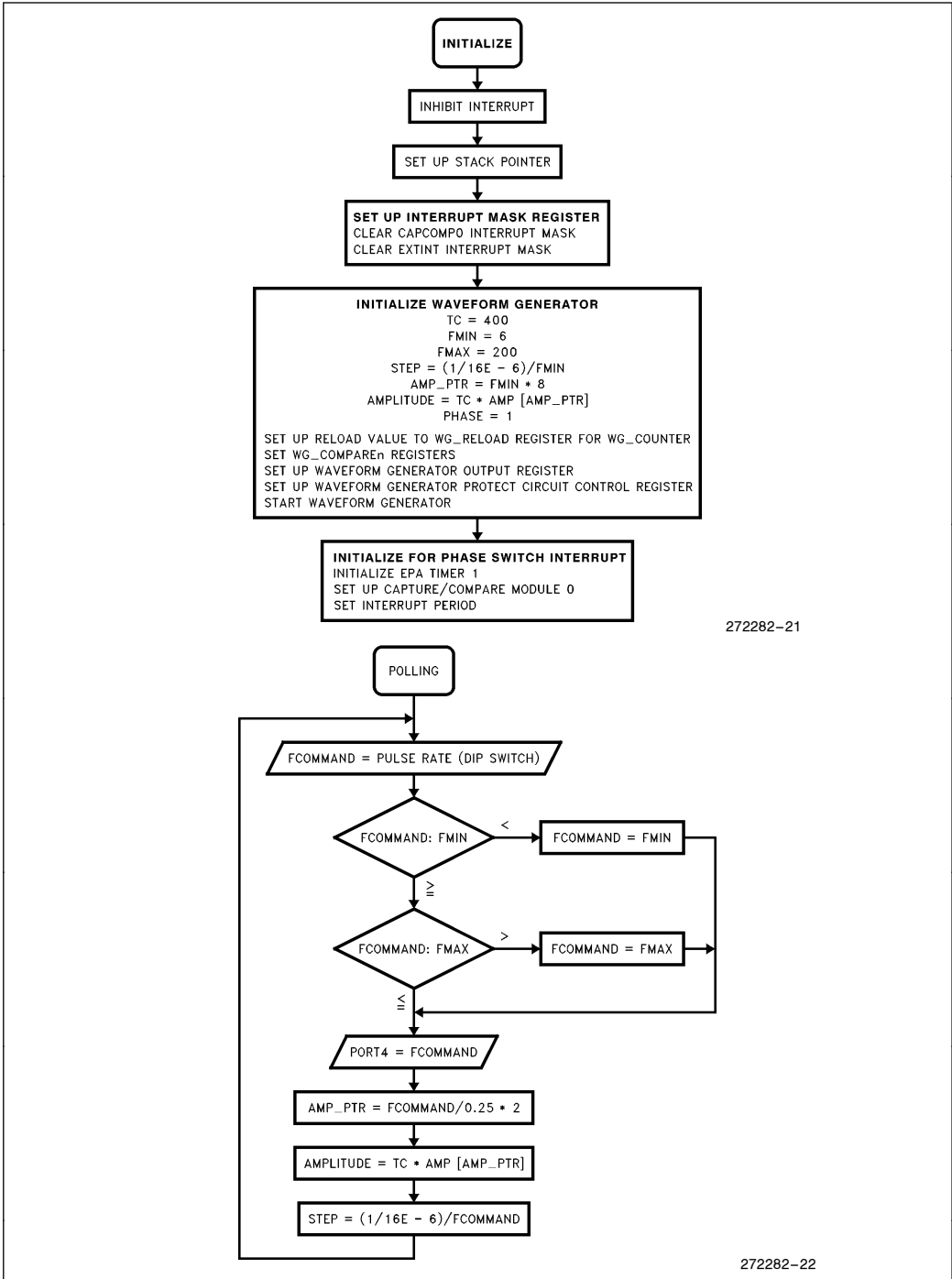
```

9.4 Program Example (Continued)

```

163 PHASE2_CCMO:                ;if phase=2 then do this
164     LD     WG_OUT_WO,#230AH  ;0010 0011 0000 1010B
165     BR     CAPCOMPO_1
166 PHASE3_CCMO:                ;if phase=3 then do this
167     LD     WG_OUT_WO,#2308H  ;0010 0011 0000 1000B
168     BR     CAPCOMPO_1
169 PHASE4_CCMO:                ;if phase=4 then do this
170     LD     WG_OUT_WO,#2309H  ;0010 0011 0000 1001B
171     BR     CAPCOMPO_1
172 PHASE5_CCMO:                ;if phase=5 then do this
173     LD     WG_OUT_WO,#2301H  ;0010 0011 0000 0001B
174     BR     CAPCOMPO_1
175 PHASE6_CCMO:                ;if phase=6 then do this
176     LD     WG_OUT_WO,#2305H  ;0010 0011 0000 0101B
177
178 CAPCOMPO_1:
179     INCB   PHASE              ;increment phase counter
180 CAPCOMPO_INT_END:
181     LD     WG_COMP1_WO,AMPLITUDE ;update WG_COMPn
182     LD     WG_COMP2_WO,AMPLITUDE
183     LDB   WSR,#3DH           ;map 64 bytes to 1F40H-1F7FH
184     ADD   CAPCOMPO_TIME_WO,STEP ;set next interrupt period
185     POPA
186     RET                      ;load PSW,INT_MASK,INT_MASK1,WSR
187                               ;return to main loop
188 ;*****
189 ; EXTERNAL INTERRUPT ROUTINE
190 ;*****
191 ;
192 EXT_INT:
193     PUSHA                      ;save PSW,INT_MASK,INT_MASK1,WSR
194     -
195     -
196     -
197     POPA                      ;load PSW,INT_MASK,INT_MASK1,WSR
198     RET                      ;return to main loop
199 ;
200 ;*****
201 ; DATA TABLE
202 ;*****
203 ;
204 AMP:                               ;value by word (0.25Hz step)
205     -
206     -
229     DCW   64551              ;Amplitude(6Hz) 98.5%
207     -
208     -
605     DCW   49151              ;Amplitude(100Hz) 75%
209     -
210     -
1005    DCW   32768              ;Amplitude(200Hz) 50%
1216
1217 END

```



272282-21

272282-22

Figure 9-5. Flow Chart—4-Phase Stepper Motor Initialization

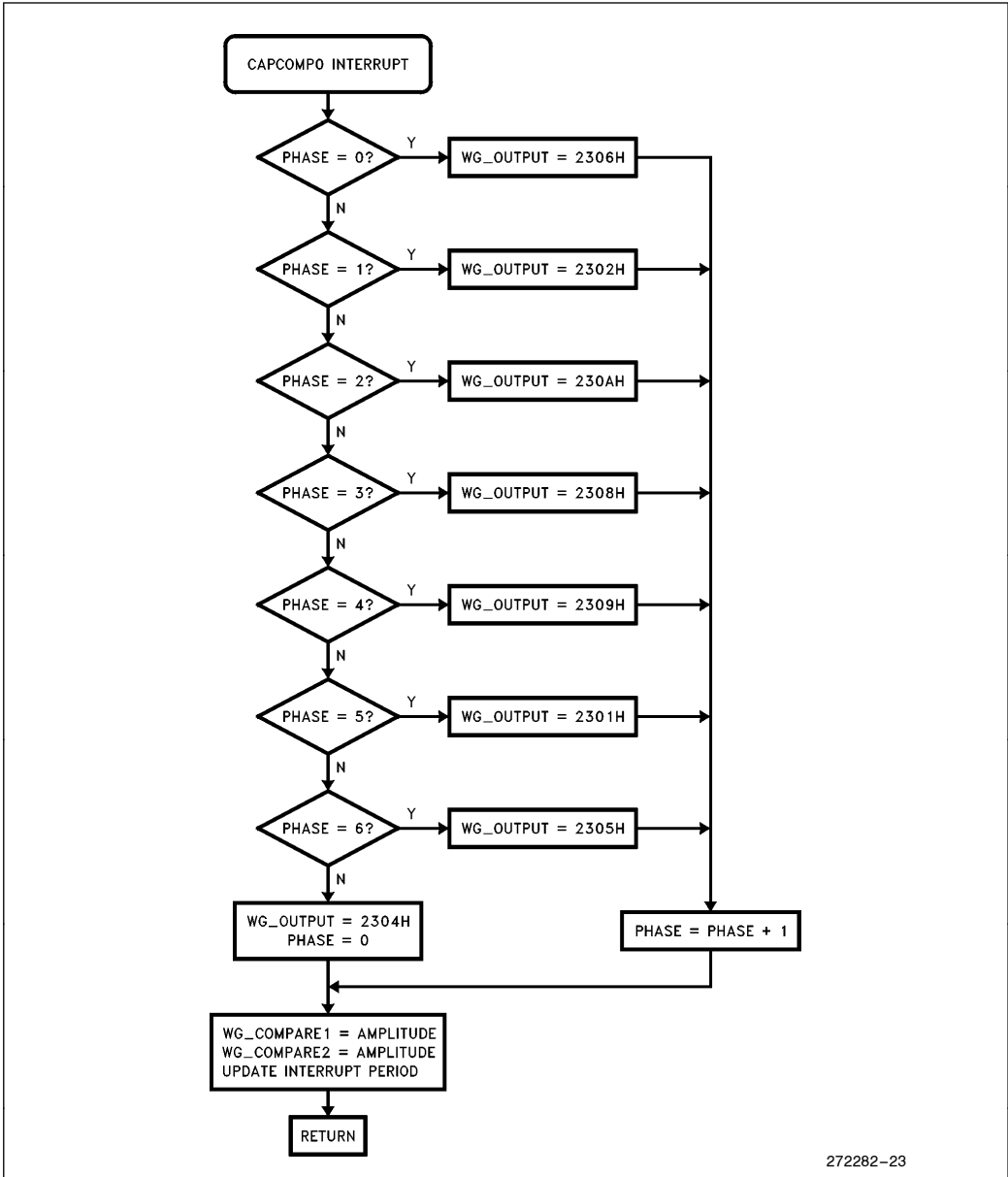


Figure 9-6. Flow Chart—4-Phase Stepper Motor EPA Interrupt

272282-23

10.0 THE FREQUENCY GENERATOR (MD ONLY)

10.1 Introduction

The Frequency Generator (FG) peripheral produces a fixed 50% duty cycle waveform that can vary in frequency from 4 kHz–1 MHz (@ $F_{XTAL} = 16 \text{ MHz}$). There are 2 Special Function Registers (SFR's) directly associated with the Frequency Generator, $FREQ_GEN$ and $FREQ_CNT$. Additionally, P7.7 must be configured for its special function in order for the frequency to be output. Figure 10-1 shows a block diagram of the FG.

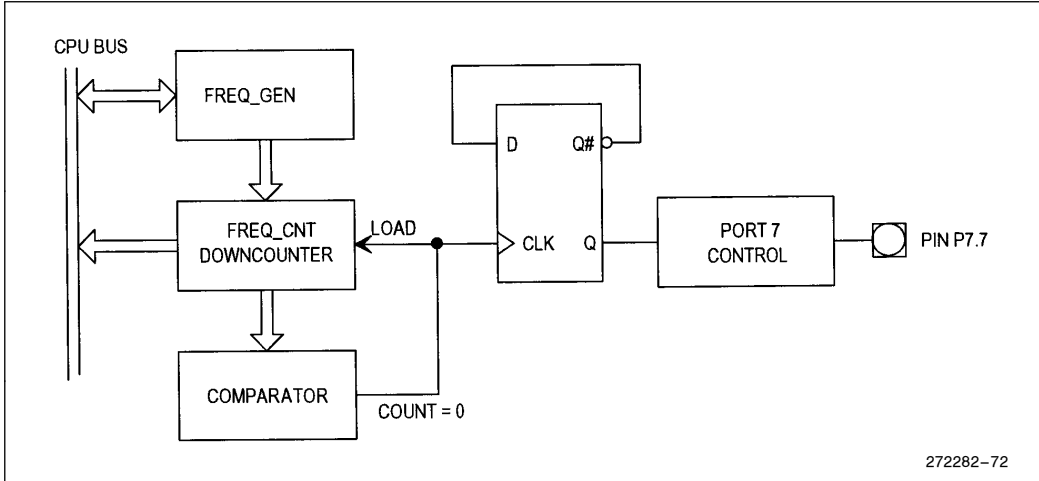


Figure 10-1. The Frequency Generator

When a value is written $FREQ_GEN$, it is transferred to the down counter. The down counter counts down to 0, and reloads from $FREQ_GEN$. Each load toggles the D flip-flop, thus producing the 50% duty cycle output.

Figures 10-2 and 10-3 detail the $FREQ_GEN$ and $FREQ_CNT$ SFR's. $FREQ_GEN$ is an 8-bit read/write register which controls the frequency of the FG. To calculate the value to load into $FREQ_GEN$, use the following formula:

$$FREQ_GEN_VALUE = \frac{F_{XTAL}}{16 \times FREQ_OUT} - 1$$

where

- $FREQ_GEN_VALUE$ = 8-bit value to load into $FREQ_GEN$
- F_{XTAL} = Frequency input on XTAL1 pin, MHz
- $FREQ_OUT$ = Output frequency, MHz

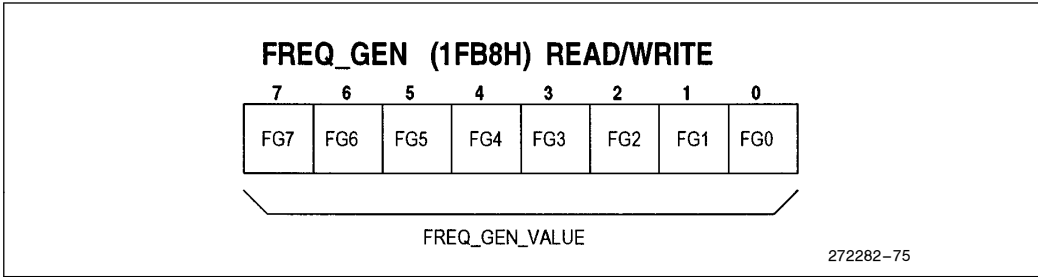


Figure 10-2. FREQ_GEN Register

The FREQ_CNT Register is a read-only SFR. It can be monitored to know the current value of the down-count.

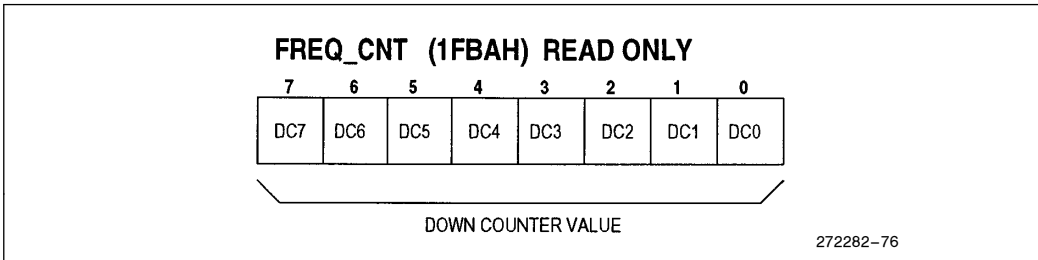


Figure 10-3. FREQ_CNT Register

10.2 Using the Frequency Generator

One application for the FG is to drive an infrared (IR) LED to transmit remote control data and/or control signals. Figure 10-4 illustrates how the IR remote control system works.

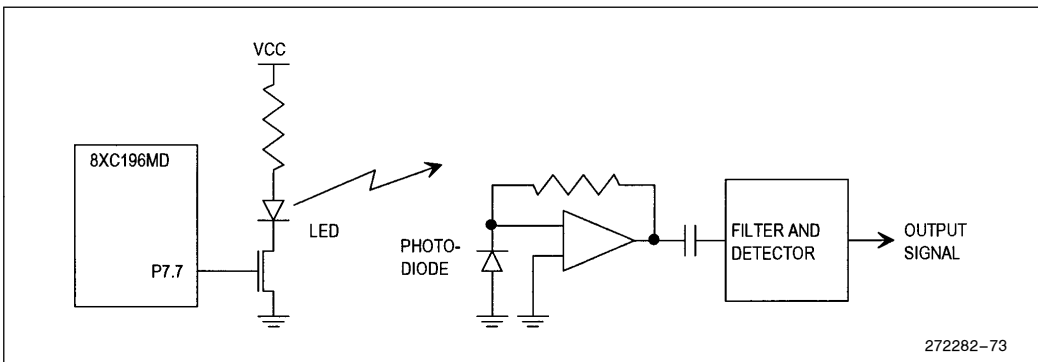


Figure 10-4. Infrared Remote Control Block Diagram

The FG is set to a carrier frequency in the 40 kHz range, and is switched on and off by writing to the port 7.7 mode register. Information is transmitted in a serial format; many coding schemes are possible, Figure 10-5 illustrates the code sequence used in the following software example.

Zeros are represented by a 1 ms carrier burst followed by a 1 ms pause. Ones are represented by a 2 ms carrier burst followed by a 2 ms pause.

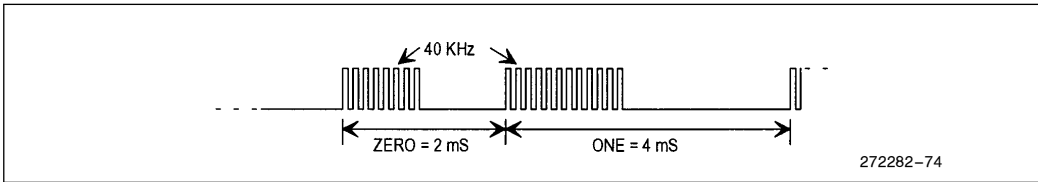


Figure 10-5. Encoding of Zeroes and Ones

At the receiving end, a photodiode receives the light pulses. Since there is a great deal of ambient light, a high-pass filter rejects the low-frequency background light, while allowing the 40 kHz carrier to pass. This carrier is amplified and detected to reproduce the original pulse sequence.

10.3 Program to Transmit Data via Frequency Generator

The following software transmits a block of data serially using the FG. A carrier frequency of 40 kHz is selected. The EPA COMPARE3 channel provides the time base for the ones and zeroes.

10.3.1 CONSTANT AND VARIABLE DECLARATIONS (Lines 1–81)

Lines 24–30 define the constants used by this program. These can be changed at assembly time as required.

Lines 36–48 define SFR locations as seen through a window. Required to allow using compact read-modify-write instructions.

Lines 54–55 define the location of the Frequency Generator SFR's.

Lines 65–81 define the variables used by the program.

10.3.2 MAIN PROGRAM (Lines 82–132)

Lines 87–93 define the program location and set up the interrupts and stack.

Lines 95–99 initialize port 7.7 for I/O, and set the pin low.

Lines 101–105 fill the data buffer with a “fill” character. An application would normally place a block of data here.

Lines 107–111 start timer1 operating with a 1 μ s clock period, load the carrier frequency into the FG, and enable the interrupts.

Lines 117–125 issue a 1 ms pulse on P2.0 for use with an oscilloscope monitor.

Lines 127–129 initialize the buffers and flag register needed by the sending interrupt routine.

Line 130 starts the buffer send by setting the interrupt pending bit for the COMPARE3 module.

Lines 131–132 loop waiting for the buffer send to complete. When done, the program starts over again.

10.3.3 COMPARE3 INTERRUPT ROUTINE

The Compare3 interrupt routine is entered each time the Compare3 channel times out. The interrupt can occur for several different reasons. The type of service needed is tracked by a “flag” register, which contains 5 bits determining the current status of the serial data stream. Those conditions are as follows:

1. A “one” is being transmitted
2. A “zero” is being transmitted
3. The next bit needs to be fetched
4. The next byte needs to be fetched
5. The buffer is finished being transmitted

Since the logic of these routines is a bit confusing, refer to the flow charts, Figures 10-6 and 10-7 for an explanation of how this routine works.

10.4 Program Example

Program to Transmit Data via Frequency Generator

```

$debug
$nolist
#include (c:\ecm\196mc\mc.inc)
$list
;
;*****
; PROGRAM FREQ.A96
;
; This program transmits a block of data
; by gating the frequency generator on and off.
;
; ONEs are represented by a long carrier burst
; followed by a long space (no carrier).
; ZERO's are represented by a short carrier burst
; followed by a short space, thus generating a MFM waveform
;
; This program is assembled to run on the MD demo board
;
;
;*****
; Program equates
;*****
;
zero_time          equ          1000          ;1 ms
zero_space_time    equ          zero_time
one_time           equ          2000          ;2 ms
one_space_time     equ          one_time
carrier_freq       equ          25           ;40 KHz
buf_size           equ          8            ;size of data buffer (bytes)
fill_char          equ          10100011b    ;initial data for buffer
;
;*****
; SFR equates in a 32-byte window
;*****
;
p2_mode_w          equ          0f0H         ;WSR = 7EH 1FD0H
p2_dir_w           equ          0F2H         ;WSR = 7EH 1FD2H
p2_reg_w           equ          0F4H         ;WSR = 7EH 1FD4H
p2_pin_w           equ          0F6H         ;WSR = 7EH 1FD6H
;
p7_mode_w          equ          0f1H         ;WSR = 7EH 1FD1H
p7_dir_w           equ          0F3H         ;WSR = 7EH 1FD3H
p7_reg_w           equ          0F5H         ;WSR = 7EH 1FD5H
p7_pin_w           equ          0F7H         ;WSR = 7EH 1FD7H
;
timer1_w           equ          0FAH         ;WSR = 7BH 1F7AH
comp3_con_w        equ          0E4H         ;WSR = 7BH 1F64H
comp3_time_w       equ          0E6H         ;WSR = 7BH 1F66H
;
;*****
;other sfr equates
;*****
;

```

272282-24

10.4 Program Example (Continued)

Program to Transmit Data via Frequency Generator (Continued)

```

54.  freq_gen          equ   1FB8H
55.  freq_cnt         equ   1FBAH
56.  ;
57.  ;*****
58.  ; Variable storage area
59.  ;*****
60.  ;
61.  rseg at 30h
62.  ;
63.  rism:            dsb   13                ;reserved for RISM
64.  ;
65.  rseg at 40h
66.  ;
67.  temp:            dsw   1
68.  temp1:           dsw   1
69.  temp2:           dsw   1
70.  buf_start:      dsw   1
71.  byte_cnt:       dsb   1
72.  bit_cnt:        dsb   1
73.  flag:           dsb   1
74.      ;bit 0 = zero being sent
75.      ;bit 1 = one being sent
76.      ;bit 5 = get next bit
77.      ;bit 6 = get next byte
78.      ;bit 7 = buffer send in progress
79.  ;
80.  xmit_buf:        dsb   buf_size         ;block of data to send
81.  shift_reg:      dsb   1
82.  ;
83.  ;*****
84.  ; main program
85.  ;*****
86.  ;
87.  cseg at 0e000h   ;RISM user space
88.  ;
89.  start:          di                ;set up interrupts
90.      dpts
91.      andb   int_pend1,#10000000b      ;
92.      orb   int_mask1,#00000010b      ;unmask compare3
93.      ld    sp,#0200h                 ;set up stack
94.  ;
95.      ldb   wsr,#7EH                  ;move SFR;s into window
96.      andb  p7_reg_w,#01111111b      ;P7.7 = low
97.      andb  p7_dir_w,#01111111b      ;P7.7 comp output
98.      andb  p7_mode_w,#01111111b     ;P7.7 = I/O
99.      ldb   wsr,r0
100. ;
101.      ld    temp,#xmit_buf            ;initialize buffer data
102.      ldb   temp1,#fill_char
103.      ldb   temp2,#buf_size
104.  fill:          stb   temp1,[temp]+
105.      djnz  temp2,fill
106. ;
107.      ldb   temp,#11000010b          ;init EPA timer1

```

272282-25

10.4 Program Example (Continued)

Program to Transmit Data via Frequency Generator (Continued)

```

108.          stb    temp,t1control[r0]          ;1 uS ticks
109.          ldb    temp,#carrier_freq        ;load carrier freq
110.          stb    temp,freq_gen[r0]        ;into freq gen
111.          ei                               ;enable interrupts
112.          ;
113.          ;*****
114.          ; Now send buffer out as serial data bytes
115.          ;*****
116.          ;
117.          ld     wsr,#7EH
118.          ld     temp,#0400
119.          andb   p2_reg_w,#11111110b      ;strobe p2.0 to sync
120.          andb   p2_dir_w,#11111110b      ;scope
121.          andb   p2_mode_w,#11111110b    ;
122.          orb    p2_reg_w,#00000001b      ;set pin high
123.          djnzw temp,$                     ;pause
124.          andb   p2_reg_w,#11111110b      ;set pin low
125.          ld     wsr,r0
126.          ;
127.          ld     buf_start,#xmit_buf       ;pointer reg
128.          ld     byte_cnt,#buf_size        ;number to send
129.          ldb    flag,#11000000b          ;set running & get byte flag
130.          ldb    int_pend1,#00000010b     ;force interrupt
131. wait:      jbs   flag,7,wait              ;loop here until done
132.          ljmp   start                    ;then start over
133.          ;
134.          ;
135.          ;*****
136.          ; Compare3 interrupt routine
137.          ;*****
138.          ;
139.          cseg at 0f120H                   ;comp3 Int vector demo board
140.          ;
141.          pusha
142.          jbs    flag,1,one_space          ;jump if one being xmit
143.          jbs    flag,0,zero_space        ;jump if zero being xmit
144.          jbs    flag,5,get_bit           ;get next bit
145.          jbs    flag,6,get_byte         ;get next byte
146.          sjmp   all_done                 ;if nothing set, done
147.          ;
148. get_byte:  andb   flag,#10111111b        ;clear get byte flag
149.          ldb    shift_reg,[buf_start]+   ;get byte to send into temp
150.          ldb    bit_cnt,#8                ;# of bits to send per char
151.          cmpb   byte_cnt,#0              ;see if last byte has been xmit
152.          jne    dec_byte_cnt             ;no!
153.          ljmp   all_done                 ;yes!
154. dec_byte_cnt: decb  byte_cnt              ;decrement byte count
155.          ;
156. get_bit:   andb   flag,#11011111b        ;clear get bit flag
157.          shlb   shift_reg,#1             ;shift MSB into carry flag
158.          jc     send_one                 ;send a one
159.          ;else send zero
160.          ;
161. send_zero: orb    flag,#00000001b        ;set zero's flag
    
```

272282-26

10.4 Program Example (Continued)

Program to Transmit Data via Frequency Generator (Continued)

```

162.          ldb   wsr,#7BH           ;set up EPA
163.          ldb   comp3_con_w,#01000000b
164.          add   comp3_time_w,timer1_w,#zero_time
165.          ldb   wsr,#7EH
166.          orb   p7_mode_w,#10000000b   ;start freq out
167.          ldb   wsr,r0
168.          sjmp  done
169.  send_one:
170.          orb   flag,#00000010b       ;set one's flag
171.          ldb   wsr,#7BH           ;set up EPA
172.          ldb   comp3_con_w,#01000000b
173.          add   comp3_time_w,timer1_w,#one_time
174.          ldb   wsr,#7EH
175.          orb   p7_mode_w,#10000000b   ;start freq out
176.          ldb   wsr,r0
177.          sjmp  done
178.  ;
179.  zero_space: andb  flag,#11111110b       ;turn off zero flag
180.          ldb   wsr,#7EH
181.          andb  p7_mode_w,#01111111b   ;turn off freq out
182.          ldb   wsr,#7BH           ;set up EPA
183.          ldb   comp3_con_w,#01000000b
184.          add   comp3_time_w,timer1_w,#zero_space_time
185.          ldb   wsr,r0
186.          sjmp  next
187.  ;
188.  one_space:  andb  flag,#11111101b       ;turn off one flag
189.          ldb   wsr,#7EH
190.          andb  p7_mode_w,#01111111b   ;turn off freq out
191.          ldb   wsr,#7BH           ;set up EPA
192.          ldb   comp3_con_w,#01000000b
193.          add   comp3_time_w,timer1_w,#one_space_time
194.          ldb   wsr,r0
195.  ;
196.  next:      decb  bit_cnt               ;decrement bit count
197.          cmpb  bit_cnt,#00H           ;check if 8 bits sent
198.          jne   next1                  ;no, get next bit
199.          orb   flag,#01000000b       ;yes, get next byte
200.          sjmp  done                   ;done for now
201.  next1:    orb   flag,#00100000b     ;set get bit flag
202.          sjmp  done
203.  ;
204.  all_done:  ldb   flag,#00H           ;clear all flags
205.  done:     popa
206.          ret
207.  ;
208.  end

```

272282-27

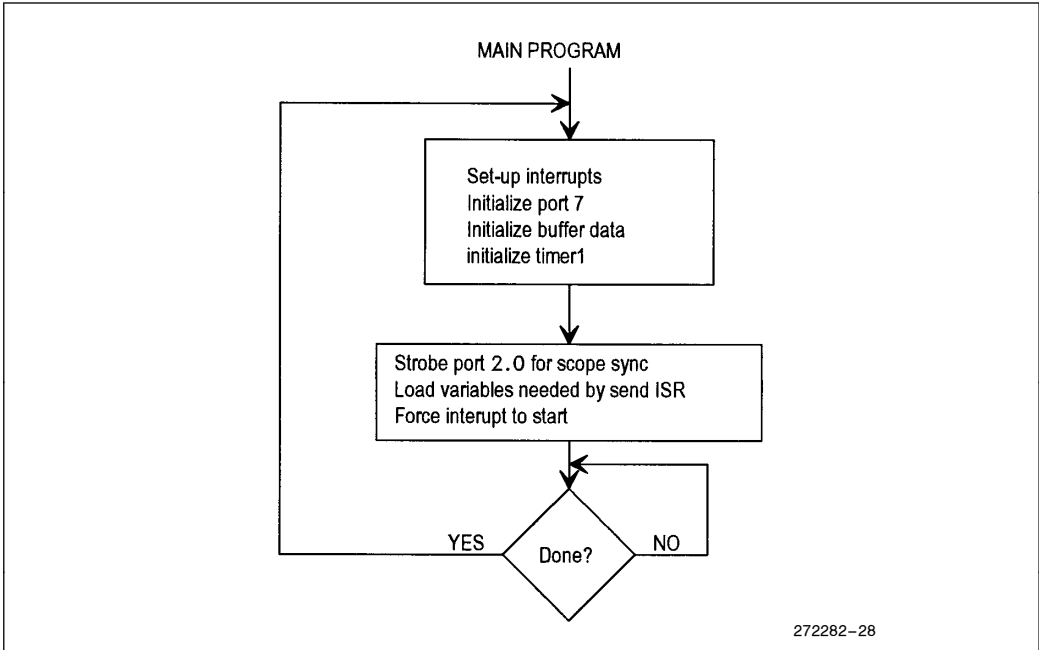
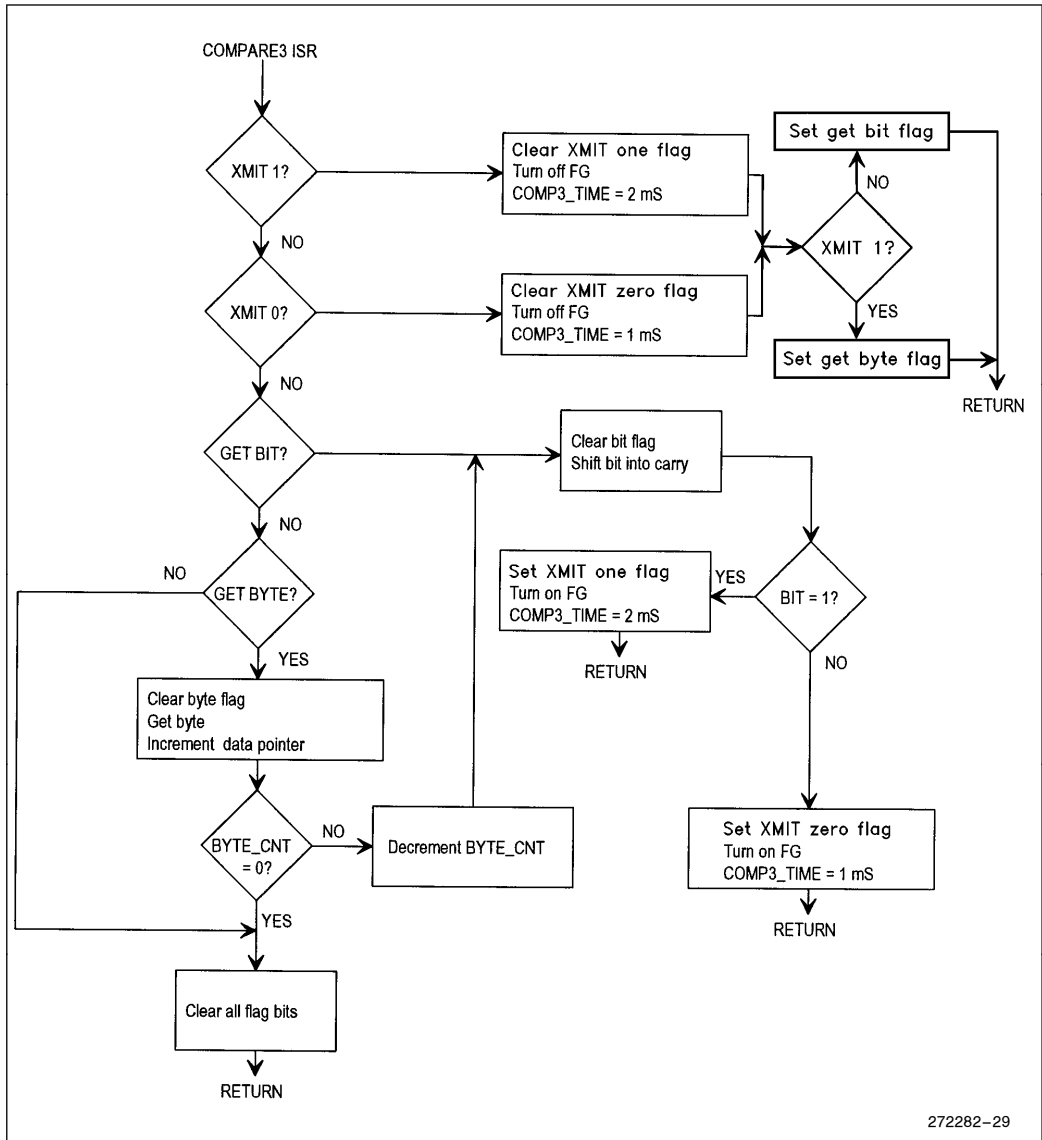


Figure 10-6. Flow Chart—Frequency Generator Initialization



272282-29

Figure 10-7. Interrupt Routine Flow Diagram