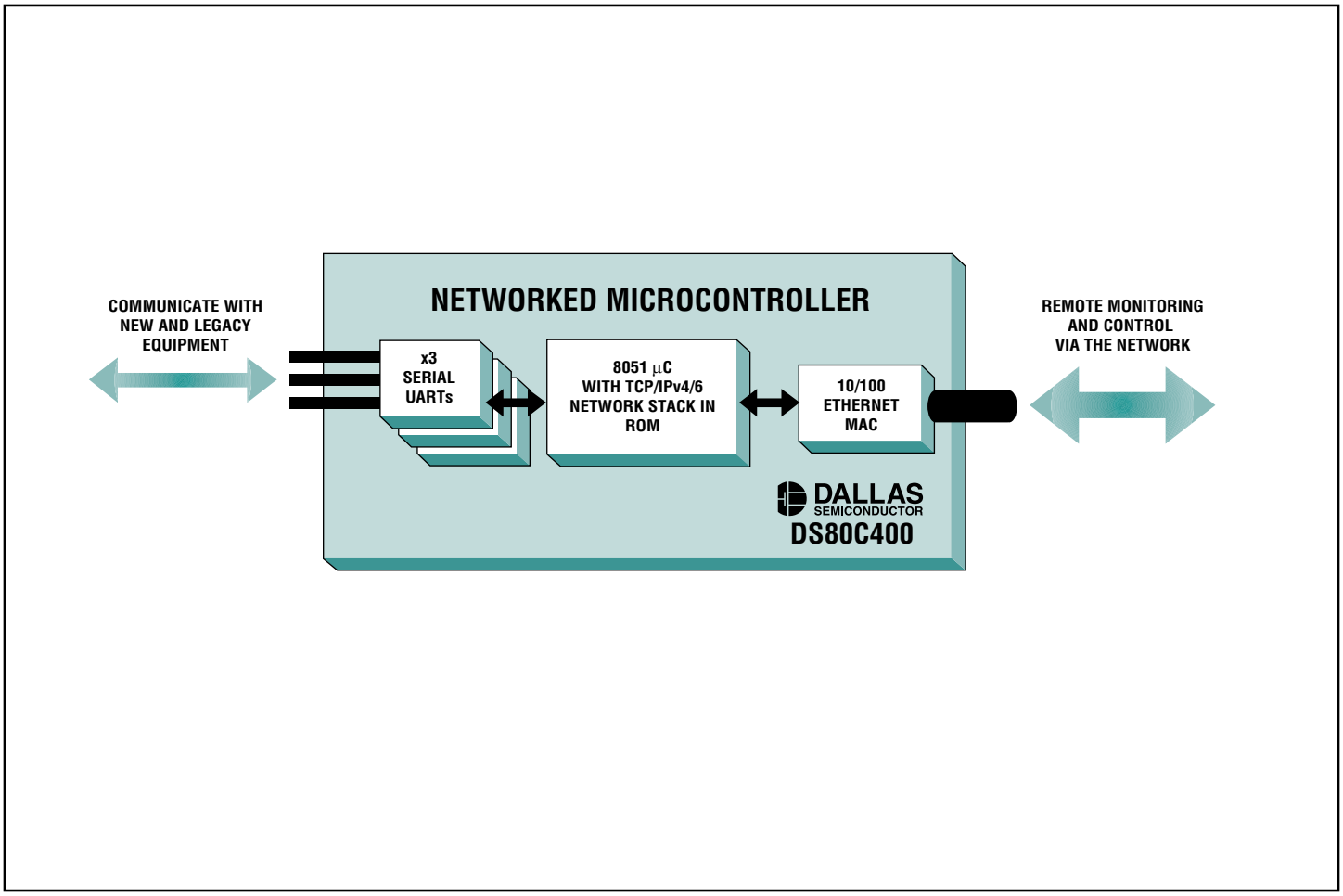


SECTION 1: INTRODUCTION

ADDENDUM TO SECTION 1: INTRODUCTION

This document is provided as a supplement to the *High-Speed Microcontroller User's Guide*, covering new or modified features specific to the DS80C400. ***This document must be used in conjunction with the High-Speed Microcontroller User's Guide, available from Dallas Semiconductor.*** Addenda are arranged by section number, which correspond to sections in the *High-Speed Microcontroller User's Guide*.

The following additions and changes, with respect to the *High-Speed Microcontroller User's Guide*, are contained in this document. This document is a work in progress, and updates/additions are added when available.

SECTION 2: Ordering Information

Information on new member of the high-speed microcontroller family has been added.

SECTION 3: Architecture

Description of new architectural features for the DS80C400 is included in this section.

SECTION 4: Programming Model

Descriptions of new and modified special function registers in the DS80C400 have been included.

SECTION 5: CPU Timing

SECTION 6: Memory Access

SECTION 7: Power Management

SECTION 8: Reset Conditions

SECTION 9: Interrupts

SECTION 10: Parallel I/O

SECTION 11: Programmable Timers

SECTION 12: Serial I/O

SECTION 13: Timed-Access Protection

SECTION 14: Real-Time Clock

No changes. Not applicable to the DS80C400.

SECTION 15: Battery Backup

No changes. Not applicable to the DS80C400.

SECTION 16: Instruction Set Details

SECTION 17: Troubleshooting

Information on the software breakpoint mode has been added.

SECTION 18: Microcontroller Development Support

SECTION 19: Controller Area Network (CAN) Module

SECTION 20: Arithmetic Accelerator

SECTION 21: 1-Wire Bus Master

SECTION 22: Ethernet Media Access Controller

SECTION 23: Embedded DS80C400 Silicon Software

SECTION 1: INTRODUCTION

The DS80C400 is the third-generation microcontroller in the Dallas Semiconductor 8051 family. It is derived from the DS87C520, but adds a full CAN 2.0B controller, a 16/32-bit arithmetic accelerator, a 1-Wire® bus master, and an IEEE 802.3-compliant Ethernet media access controller. It incorporates the 8051-compatible high-speed microcontroller core, which has been redesigned to reduce the original 8051's 12 clocks per instruction cycle to four clocks, while using less power. The DS80C400 offers a maximum system clock speed of 75MHz. The DS80C400 also supports a larger program space, data memory space, and stack memory.

The DS80C400 supports three programmable address modes. The 16-bit 8051 address mode of operation is identical with the original 8051 operation. The 24-bit paged address mode is fully compatible with the 8051 operation, but is still capable of supporting a larger memory address range within a multiple page mode configuration. The 24-bit contiguous address mode is supported by a full 24-bit program counter and has eight instructions modified to operate in the 24-bit address range. The 24-bit contiguous address mode requires assembler, compiler, and linker support. The DS80C400 also supports an extended stack in 1kB of internal data RAM.

The DS80C400 provides four data pointers, and implements programmable features that are capable of modifying the INC DPTR instruction to actually decrement the active data pointer, automatically toggle the selection of the data pointer, and automatically increment/decrement the select data pointer.

FEATURES

- Seven bidirectional parallel ports
- Four 16-bit timers/counters with one up/down timer, capture, and baud-rate generation features
- Power-on reset flag
- Stop mode exit on interrupts, reset, and CAN bus activity
- 256 bytes of scratchpad memory
- Low-power CMOS
- High-speed, four clocks-per-machine cycle architecture
- Clock rates: DC to 75MHz (18.75 MIPS)
- Minimum instruction cycle of 53ns
- 24-bit program/data address memory access
- Program counter with selectable 16-bit, 24-bit paged, or 24-bit contiguous mode
- 16MB external interface
- 64kB on-chip ROM for bootstrap loader
- Supports network boot over Ethernet using DHCP and TFTP
- Full application-accessible TCP/IP network stack
- Supports IPv4 and IPv6
- Implements UDP, TCP, DHCP, ICMP, and IGMP
- Preemptive, priority-based task scheduler
- MAC address acquisition from IEEE-registered DS2502-E48
- 9kB on-chip data SRAM memory
- Four data pointers with auto INC/DEC function
- Extended 1kB stack
- High-speed math accelerator for 16/32-bit multiply and divide calculations
- One's complement adder
- 1-Wire bus master
- Ethernet controller supports 100/10Mbps full-duplex and half-duplex operation
- Three serial port UARTs with framing error detection and automatic address recognition

1-Wire is a registered trademark of Dallas Semiconductor.

16 interrupt sources, 6 external and 10 internal with three levels of interrupt nesting and two programmable priority levels

Crash-proof, bandgap-referenced power-fail warning; voltage sense reset; and automatic power-up reset timeout

Programmable system clock divide control of crystal oscillator. Options include:

- Divide-by-1–18.75MHz max. crystal
- Divide-by-2–37.5MHz max. crystal
- Divide-by-4–Standard operation
- Divide-by-1024–Low-speed/power

Status register to verify active-interrupt nesting and real-time serial port transmit/receive activity

User-selectable multiplexed or nonmultiplexed external address/data interface

Programmable watchdog timer

Programmable clock-out and reset-out for additional external stand-alone CAN support

Full CAN 2.0B controller:

- 15 message centers
- Standard 11-bit or extended 29-bit identification modes
- Two data byte masks and associated IDs for DeviceNet™, SDS, and other higher-layer CAN protocol
- External transmit disable for autobaud
- SIESTA low-power mode

100-pin QFP package

SECTION 2: ORDERING INFORMATION

The high-speed microcontroller family follows the part-numbering convention shown below. Note that all combinations of devices are not currently available. Please refer to individual data sheets for the available versions.

PART	TEMP RANGE	PIN-PACKAGE	MAX CLOCK SPEED (MHz)
DS80C400-FNY	-40°C to +85°C	100-pin LQFP	75

SECTION 3: ARCHITECTURE

The DS80C400 is designed to provide direct compatibility to all of the traditional 80C32 functions, including a 256-byte special function register (SFR) SRAM memory, a third timer (timer 2), and serial port framing-error detection and automatic address recognition. Features on the DS80C400 that are compatible with the DS87C520 include a bandgap-based power monitor for interrupt and reset, timed-access protection, programmable on-board data memory (expanded to 9kB x 8 on the DS80C400), programmable system-clock divide ratios, two serial ports, and a programmable watchdog timer. Expanding on these features, the DS80C400 also contains an expanded interrupt capability of 16 interrupts with two programmable interrupt priorities, levels for 15 of the interrupts, and a third-level interrupt priority for power-fail. Additional features include, a math accelerator, a one's complement adder, a 1-Wire bus master, a full CAN 2.0B processor, an IEEE 802.3-compliant Ethernet media access controller, a selectable external multiplexed or nonmultiplexed address/data interface, 16-bit, 24-bit paged or 24-bit contiguous addressing operation, and internally decoded chip enables.

The DS80C400 is designed to function similarly to the DS80C390 and run with external program and data memory. The DS80C400 has been designed to operate with an extended 24-bit address map and to support external memories with a minimum of external logic. The DS80C400 also supports an optional extended stack pointer and a 1kB stack memory.

CPU CORE AND CPU REGISTERS

The CPU core of the DS80C400 executes the same binary-compatible instruction set as that of the 80C32. The principal difference between the core of the DS80C400 and the 80C32 is the number of clocks required to execute specific instructions. The DS80C400 uses a divide-by-4 of the crystal oscillator, and the 80C32 functions with a divide-by-12 of the crystal oscillator. A machine cycle in the DS80C400 defaults to four periods of the crystal oscillator. A machine cycle in the 80C32 is interpreted as 12 cycles of the oscillator. The four MOVX data memory instructions of the DS80C400 have the additional capability of being stretched (external data memory bus access only) from the original data memory access (read or write) time. The MOVX instruction ranges from two machine cycles to 12 machine cycles across eight programmable settings. This MOVX stretch control is user-selectable with the MD2, MD1, and MD0

DeviceNet is a trademark of OpenDeviceNet Vendor Association Inc.

bits in the clock control register. The ability to do an instruction-based decrement of the DPTR registers is also now supported, through additional control bits in the DPS1 and DPS SFRs.

The DS80C400 supports one of three different addressing modes, as selected by software through the AM1 and AM0 bits in the ACON SFR. The microcontroller functions in either the traditional 16-bit address mode, a 24-bit paged address mode, or in a 24-bit contiguous program mode. The microprocessor defaults after a reset to the traditional 16-bit mode, which is identical to the DS80C320 (A23–A16 are forced to 00h). The 24-bit paged address mode is binary code compliant with traditional compilers for the standard 16-bit address range, but allows for up to 16MB of program and 4MB of data memory. A new address page SFR implements an internal bank-switching mechanism in response to a certain set of call/return instructions. The 24-bit contiguous mode requires a 24-bit address compiler that supports contiguous program flow over the entire 24-bit address range by the addition of an operand and/or cycles to eight basic instructions (without the need of bank switching).

The instruction is fetched and sent over the 8-bit internal data bus to the instruction register. The ALU performs math functions, logical operations, and makes comparisons and general decisions. The ALU primarily uses the accumulator and the B register as either the source or destination for most operations.

All peripherals and operations that are not explicit instructions in the DS80C400 are controlled by SFRs. The accumulator is the primary register used in the CPU. It is the source or destination for most operations. The B register is used as the second 8-bit argument in multiply and divide operations. When not used in these operations, the B register can be used as a general-purpose register.

The program status word (PSW) contains a selection of bit flags that include the carry flag, auxiliary carry flag, general-purpose flag, register bank select, overflow flag, and parity flag.

The data pointers are used in accessing program or data memory with the MOVC or MOVX instruction. Two pairs of pointers are provided, simplifying source and destination address tracking when moving data from one memory area to another memory area or to a memory-mapped peripheral.

The DS80C400 provides a stack in either the original 8052 scratchpad area or a 1kB programmable area of the on-chip SRAM. The stack pointer register or register pair, when using the extended 1kB stack, denotes the last used location at the top of the stack.

There are three internal buses, which include a 24-bit address bus and two 8-bit data buses. The address bus provides addresses for op code/operand fetching. The DA data bus is used for addressing SFRs, fetching instructions and operands from external memory, and providing addresses for the internal stack. The DB data bus is used for data exchange between SFRs and the output of all ALU operations.

SECTION 4: PROGRAMMING MODEL

The DS80C400 microprocessor is based on the industry-standard 80C32. The core is an accumulator-based architecture using internal registers for data storage and peripheral control. It executes the standard 8051 instruction set. This section provides a brief description of each architecture feature. Details concerning the programming model, instruction set, and register description are provided in Section 4.

The high-speed microcontroller uses several distinct memory areas. These are registers, program memory, and data memory. Registers serve to control on-chip peripherals and as RAM. Note that registers (on-chip RAM) are separate from data memory. Registers are divided into three categories including directly addressed on-chip RAM, indirectly addressed on-chip RAM, and SFRs. As follows, the program and data memory areas are discussed under *Memory Map*, and the registers are discussed under *Registers Map*.

MEMORY MAP

The DS80C400 microcontroller defaults to the memory compatibility of the 8051. This device can address up to 1kB of on-chip SRAM. In addition to the standard 16-bit address mode, the DS80C400 can operate in 24-bit paged or 24-bit contiguous address mode. The DS80C400 has four internal memory areas: 256 bytes of scratchpad RAM, 9kB of configurable SRAM, 256 bytes of RAM reserved for the CAN message centers, and 64kB of embedded ROM firmware. A 22-bit address bus and an 8-bit data bus operating in multiplexed or demultiplexed mode can address 16MB of external memory. By configuring the SFRs, eight available chip-enable pins are used to access 16MB of external program memory. Also, 4MB of external data memory is accessible by configuring four peripheral chip-enable bits in the SFRs. The addresses of the program and data segments can overlap since they are accessed in different ways. Program memory is fetched by the microprocessor automatically. These addresses are never written by software. There is one instruction (MOVC) that is used to explicitly read the program area. This is commonly used to read look-up tables. The data memory area is accessed explicitly using the MOVX instruction. This instruction provides multiple ways of specifying the target address. In addition, the DS80C400 can be configured to permit a merged von Neumann-style program/data memory space. Detailed descriptions of the memory mapping alternatives are discussed in a separate section of this user's guide supplement.

REGISTER MAP

The register map is separate from the program and data memory areas mentioned above. A separate class of instructions is used to access the registers. There are 256 potential register location values. In practice, the high-speed microcontroller has 256 bytes of scratchpad RAM and up to 128 SFRs. This is possible since the upper 128 scratchpad RAM locations can only be accessed indirectly. That is, the contents of a working register, described later, designate the RAM location. Thus, a direct reference to one of the upper 128 locations must be an SFR access. Direct RAM is reached at locations 0 to 7Fh (0–127). SFRs are accessed directly between 80h and FFh (128–255). The RAM locations between 128 and 255 can be reached through an indirect reference to those locations.

Scratchpad RAM is available for general-purpose data storage. It is commonly used in place of off-chip RAM when the total data contents are small. When off-chip RAM is needed, the scratchpad area still provides the fastest general-purpose access. Within the 256 bytes of RAM, there are several special-purpose areas, which are described as follows.

Bit-Addressable Locations

In addition to direct register access, some individual bits in both the RAM and SFR area are also accessible. In the scratchpad RAM area, registers 20h to 2Fh are bit addressable. This provides 128 (16 x 8) individual bits available to software. The type of instruction distinguishes a bit access from a full register access. In the SFR area, any register location ending in a 0 or 8 is bit addressable.

Working Registers

As part of the lower 128 bytes of RAM, there are four banks of general-purpose working registers, each bank containing registers R0–R7. The bank is selected by bits in the program status word register. Since there are four banks, the currently selected bank is used by any instruction using R0–R7. This allows software to change context by switching banks. The working registers also allow their contents to be used for indirect addressing of the upper 128 bytes of RAM. Thus, an instruction can designate the value stored in R0, for example, to address the upper RAM. This value might be the result of another calculation.

Stack

Another use of the scratchpad area is for the programmer's stack. This area is selected using the stack pointer (SP: 81h) SFR. Whenever a call or interrupt is invoked, the return address is placed on the stack. It also is available to the programmer for variables, etc. The stack pointer defaults to 07h on reset, but can be relocated as needed. A convenient location would be the upper RAM area (> 7Fh), since this is only available indirectly. The SP points to the last used value. Therefore, the next value placed on the stack is put at SP + 1. Each PUSH or CALL increments the SP by the appropriate value. Each POP or RET decrements, as well.

The DS80C400 supports an optional 10-bit (1kB) stack. This greatly increases programming efficiency and allows the device to support large programs. When enabled by setting the stack address (SA) bit in the ACON register, 1kB of the 9kB internal SRAM is allocated for use as the stack. The 10-bit address is formed by concatenating the lower 2 bits of the extended stack pointer (ESP: 9Bh) and the 8-bit stack pointer (SP: 81h). The exact address of the 1kB is dependent on the setting of the IDM1-0 bits.

SPECIAL-FUNCTION REGISTER MAPS

Most of the unique features of the high-speed microcontroller family are controlled by bits in SFRs located in unused locations in the 8052 SFR map. This allows for increased functionality, while maintaining complete instruction set compatibility. The SFRs reside in register locations 80h–FFh and are accessed using direct addressing. SFRs that end in 0h or 8h are bit addressable.

The Special Function Register Map table indicates the names and locations of the SFRs used by the DS80C400. The Special Function Register Location table shows individual bits in those registers. Bits protected by the timed-access function are shaded. The Special Function Register Reset Values table indicates the reset state of all SFR bits. Following these tables is a complete description of DS80C400 SFRs that are new to the 8051 architecture, or have new or modified functionality.

SPECIAL-FUNCTION REGISTER MAP

START ADDRESS	SFR NAMES								END ADDRESS
80	P4	SP	DPL	DPH	DPL1	DPH1	DPS	PCON	87
88	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON		8F
90	P1	EXIF	P4CNT	DPX		DPX1	C0RMS0	C0RMS1	97
98	SCON0	SBUF0		ESP	AP	ACON	C0TMA0	C0TMA1	9F
A0	P2	P5	P5CNT	C0C	C0S	C0IR	C0TE	C0RE	A7
A8	IE	SADDR0	SADDR1	C0M1C	C0M2C	C0M3C	C0M4C	C0M5C	AF
B0	P3	P6	P6CNT	C0M6C	C0M7C	C0M8C	C0M9C	C0M10C	B7
B8	IP	SADEN0	SADEN1	C0M11C	C0M12C	C0M13C	C0M14C	C0M15C	BF
C0	SCON1	SBUF1			PMR	STATUS	MCON	TA	C7
C8	T2CON	T2MOD	RCAP2L	RCAP2H	TL2	TH2	COR		CF
D0	PSW	MCNT0	MCNT1	MA	MB	MC	MCON1	MCON2	D7
D8	WDCON	SADDR2	BPA1	BPA2	BPA3				DF
E0	ACC	OCAD		CSRD	CSRA	EBS	BCUD	BCUC	E7
E8	EIE		MXAX	DPX2		DPX3	OWMAD	OWMDR	EF
F0	B	SADEN2	DPL2	DPH2	DPL3	DPH3	DPS1	STATUS1	F7

SPECIAL-FUNCTION REGISTER LOCATION

REGISTER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	ADDRESS
P4									80h
SP									81h
DPL									82h
DPH									83h
DPL1									84h
DPH1									85h
DPS	ID1	ID0	TSL	AID	SEL1	—	—	SEL	86h
PCON	SMOD_0	SMOD0	OFDF	OFDE	GF1	GF0	STOP	IDLE	87h
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88h
TMOD	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	89h
TL0									8Ah
TL1									8Bh
TH0									8Ch
TH1									8Dh
CKCON	WD1	WD0	T2M	T1M	T0M	MD2	MD1	MD0	8Eh
P1									90h
EXIF	IE5	IE4	IE3	IE2	CKRY	RGMD	RGSL	BGS	91h
P4CNT	—	—	P4CNT.5	P4CNT.4	P4CNT.3	P4CNT.2	P4CNT.1	P4CNT.0	92h
DPX									93h
DPX1									95h
C0RMS0									96h
C0RMS1									97h

SPECIAL-FUNCTION REGISTER LOCATION (CONTINUED)

REGISTER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	ADDRESS
SCON0	SM0/FE_0	SM1_0	SM2_0	REN_0	TB8_0	RB8_0	TI_0	RI_0	98h
SBUF0									99h
ESP	—	—	—	—	—	—	ESP.1	ESP.0	9Bh
AP									9Ch
ACON	—	—	MROM	BPME	BROM	SA	AM1	AM0	9Dh
C0TMA0									9Eh
C0TMA1									9Fh
P2									A0h
P5									A1h
P5CNT	—	CAN0BA	—	—	C0_I/O	P5CNT.2	P5CNT.1	P5CNT.0	A2h
C0C	ERIE	STIE	PDE	SIESTA	CRST	AUTOB	ERCS	SWINT	A3h
C0S	BSS	EC96/128	WKS	RXS	TXS	ER2	ER1	ER0	A4h
C0IR	INTIN7	INTIN6	INTIN5	INTIN4	INTIN3	INTIN2	INTIN1	INTIN0	A5h
C0TE									A6h
C0RE									A7h
IE	EA	ES1	ET2	ES0	ET1	EX1	ET0	EX0	A8h
SADDR0									A9h
SADDR1									AAh
C0M1C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	ABh
C0M2C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	ACh
C0M3C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	ADh
C0M4C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	AEh
C0M5C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	AFh
P3									B0h
P6									B1h
P6CNT	—	—	P6CNT.5	P6CNT.4	P6CNT.3	P6CNT.2	P6CNT.1	P6CNT.0	B2h
C0M6C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	B3h
C0M7C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	B4h
C0M8C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	B5h
C0M9C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	B6h
C0M10C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	B7h
IP	—	PS1	PT2	PS0	PT1	PX1	PT0	PX0	B8h
SADEN0									B9h
SADEN1									BAh
C0M11C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	BBh
C0M12C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	BCh
C0M13C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	BDh
C0M14C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	BEh
C0M15C	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP	BFh
SCON1	SM0/FE_1	SM1_1	SM2_1	REN_1	TB8_1	RB8_1	TI_1	RI_1	C0h
SBUF1									C1h
PMR	CD1	CD0	SWB	CTM	4X/$\overline{2X}$	ALEOFF	—	—	C4h
STATUS	PIP	HIP	LIP	—	SPTA1	SPRA1	SPTA0	SPRA0	C5h
MCON	IDM1	IDM0	CMA	—	PDCE3	PDCE2	PDCE1	PDCE0	C6h

SPECIAL-FUNCTION REGISTER LOCATION (CONTINUED)

REGISTER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	ADDRESS
TA									C7h
T2CON	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	C8h
T2MOD	—	—	—	D13T1	D13T2	—	T2OE	DCEN	C9h
RCAP2L									CAh
RCAP2H									CBh
TL2									CCh
TH2									CDh
COR	IRDACK	—	—	C0BPR7	C0BPR6	COD1	COD0	XCLKOE	CEh
PSW	CY	AC	F0	RS1	RS0	OV	F1	P	D0h
MCNT0	LSHIFT	CSE	SCE	MAS4	MAS3	MAS2	MAS1	MAS0	D1h
MCNT1	MST	MOF	SCB	CLM	—	—	—	—	D2h
MA									D3h
MB									D4h
MC									D5h
MCON1	—	—	—	—	PDCE7	PDCE6	PDCE5	PDCE4	D6h
MCON2	WPIF	WPR2	WPR1	WPR0	WPE3	WPE2	WPE1	WPE0	D7h
WDCON	SMOD_1	POR	EPF1	PF1	WDIF	WTRF	EWT	RWT	D8h
SADDR2									D9h
BPA1									DAh
BPA2									DBh
BPA3									DCh
ACC									E0h
OCAD									E1h
CSRD									E3h
CSRA									E4h
EBS	FPE	RBF	—	BS4	BS3	BS2	BS1	BS0	E5h
BCUD									E6h
BCUC	BUSY	EPMF	TIF	RIF	BC3	BC2	BC1	BC0	E7h
EIE	EPMIE	C0IE	EAIE	EWDI	EWPI	ES2	ET3	EX2-5	E8h
MXAX									EAh
DPX2									EBh
DPX3									EDh
OWMAD	—	—	—	—	—	A2	A1	A0	EEh
OWMDR									EFh
B									F0h
SADEN2									F1h
DPL2									F2h
DPH2									F3h
DPL3									F4h
DPH3									F5h
DPS1	ID3	ID2	—	—	—	—	—	—	F6h
STATUS1	—	—	—	—	V1PF	V3PF	SPTA2	SPRA2	F7h
EIP	EPMIP	C0IP	EAIP	PWDI	PWPI	PS2	PT3	PX2-5	F8h
P7									F9h
TL3									FBh
TH3									FCh
T3CM	TF3	TR3	T3M	SMOD_2	GATE	C/$\overline{T3}$	M1	M0	FDh
SCON2	SM0/FE_2	SM1_2	SM2_2	REN_2	TB8_2	RB8_2	TI_2	RI_2	FEh
SBUF2									FFh

SPECIAL-FUNCTION REGISTER RESET VALUES

REGISTER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	ADDRESS
P4	1	1	1	1	1	1	1	1	80h
SP	0	0	0	0	0	1	1	1	81h
DPL	0	0	0	0	0	0	0	0	82h
DPH	0	0	0	0	0	0	0	0	83h
DPL1	0	0	0	0	0	0	0	0	84h
DPH1	0	0	0	0	0	0	0	0	85h
DPS	0	0	0	0	0	1	0	0	86h
PCON	0	0	SPECIAL	0	0	0	0	0	87h
TCON	0	0	0	0	0	0	0	0	88h
TMOD	0	0	0	0	0	0	0	0	89h
TL0	0	0	0	0	0	0	0	0	8Ah
TL1	0	0	0	0	0	0	0	0	8Bh
TH0	0	0	0	0	0	0	0	0	8Ch
TH1	0	0	0	0	0	0	0	0	8Dh
CKCON	0	0	0	0	0	0	0	1	8Eh
P1	1	1	1	1	1	1	1	1	90h
EXIF	0	0	0	0	SPECIAL	SPECIAL	SPECIAL	0	91h
P4CNT	1	1	1	1	1	1	1	1	92h
DPX	0	0	0	0	0	0	0	0	93h
DPX1	0	0	0	0	0	0	0	0	95h
C0RMS0	0	0	0	0	0	0	0	0	96h
C0RMS1	0	0	0	0	0	0	0	0	97h
SCON0	0	0	0	0	0	0	0	0	98h
SBUF0	0	0	0	0	0	0	0	0	99h
ESP	1	1	1	1	1	1	0	0	9Bh
AP	0	0	0	0	0	0	0	0	9Ch
ACON	1	1	0	0	SPECIAL	0	0	0	9Dh
C0TMA0	0	0	0	0	0	0	0	0	9Eh
C0TMA1	0	0	0	0	0	0	0	0	9Fh
P2	1	1	1	1	1	1	1	1	A0h
P5	1	1	1	1	1	1	1	1	A1h
P5CNT	1	0	0	0	0	0	0	0	A2h
C0C	0	0	0	0	1	0	0	1	A3h
C0S	0	0	0	0	0	0	0	0	A4h
C0IR	0	0	0	0	0	0	0	0	A5h
C0TE	0	0	0	0	0	0	0	0	A6h
C0RE	0	0	0	0	0	0	0	0	A7h
IE	0	0	0	0	0	0	0	0	A8h
SADDR0	0	0	0	0	0	0	0	0	A9h
SADDR1	0	0	0	0	0	0	0	0	AAh
C0M1C	0	0	0	0	0	0	0	0	ABh
C0M2C	0	0	0	0	0	0	0	0	ACh
C0M3C	0	0	0	0	0	0	0	0	ADh
C0M4C	0	0	0	0	0	0	0	0	AEh
C0M5C	0	0	0	0	0	0	0	0	AFh
P3	1	1	1	1	1	1	1	1	B0h
P6	1	1	1	1	1	1	1	1	B1h
P6CNT	0	0	0	0	0	0	0	0	B2h
C0M6C	0	0	0	0	0	0	0	0	B3h
C0M7C	0	0	0	0	0	0	0	0	B4h
C0M8C	0	0	0	0	0	0	0	0	B5h
C0M9C	0	0	0	0	0	0	0	0	B6h
C0M10C	0	0	0	0	0	0	0	0	B7h

SPECIAL-FUNCTION REGISTER RESET VALUES (CONTINUED)

REGISTER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	ADDRESS
IP	1	0	0	0	0	0	0	0	B8h
SADEN0	0	0	0	0	0	0	0	0	B9h
SADEN1	0	0	0	0	0	0	0	0	BAh
C0M11C	0	0	0	0	0	0	0	0	BBh
C0M12C	0	0	0	0	0	0	0	0	BCh
C0M13C	0	0	0	0	0	0	0	0	BDh
C0M14C	0	0	0	0	0	0	0	0	BEh
C0M15C	0	0	0	0	0	0	0	0	BFh
SCON1	0	0	0	0	0	0	0	0	C0h
SBUF1	0	0	0	0	0	0	0	0	C1h
PMR	1	0	0	0	0	0	1	1	C4h
STATUS	0	0	0	1	0	0	0	0	C5h
MCON	0	0	0	1	0	0	0	0	C6h
TA	1	1	1	1	1	1	1	1	C7h
T2CON	0	0	0	0	0	0	0	0	C8h
T2MOD	1	1	0	0	0	1	0	0	C9h
RCAP2L	0	0	0	0	0	0	0	0	CAh
RCAP2H	0	0	0	0	0	0	0	0	CBh
TL2	0	0	0	0	0	0	0	0	CCh
TH2	0	0	0	0	0	0	0	0	CDh
COR	0	1	1	0	0	0	0	0	CEh
PSW	0	0	0	0	0	0	0	0	D0h
MCNT0	0	0	0	0	0	0	0	0	D1h
MCNT1	0	0	0	0	1	1	1	1	D2h
MA	0	0	0	0	0	0	0	0	D3h
MB	0	0	0	0	0	0	0	0	D4h
MC	0	0	0	0	0	0	0	0	D5h
MCON1	1	1	1	1	0	0	0	0	D6h
MCON2	0	0	0	0	0	0	0	0	D7h
WDCON	0	SPECIAL	0	SPECIAL	0	SPECIAL	SPECIAL	0	D8h
SADDR2	0	0	0	0	0	0	0	0	D9h
BPA1	0	0	0	0	0	0	0	0	DAh
BPA2	0	0	0	0	0	0	0	0	DBh
BPA3	0	0	0	0	0	0	0	0	DCh
ACC	0	0	0	0	0	0	0	0	E0h
OCAD	0	0	0	0	0	0	0	0	E1h
CSRD	0	0	0	0	0	0	0	0	E3h
CSRA	0	0	0	0	0	0	0	0	E4h
EBS	0	1	1	0	0	0	0	0	E5h
BCUD	0	0	0	0	0	0	0	0	E6h
BCUC	0	0	0	0	0	0	0	0	E7h
EIE	0	0	0	0	0	0	0	0	E8h
MXAX	0	0	0	0	0	0	0	0	EAh
DPX2	0	0	0	0	0	0	0	0	EBh
DPX3	0	0	0	0	0	0	0	0	EDh
OWMAD	0	0	0	0	0	1	1	1	EEh
OWMDR	0	0	0	0	0	0	0	0	EFh
B	0	0	0	0	0	0	0	0	F0h

SPECIAL-FUNCTION REGISTER RESET VALUES (CONTINUED)

REGISTER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	ADDRESS
SADEN2	0	0	0	0	0	0	0	0	F1h
DPL2	0	0	0	0	0	0	0	0	F2h
DPH2	0	0	0	0	0	0	0	0	F3h
DPL3	0	0	0	0	0	0	0	0	F4h
DPH3	0	0	0	0	0	0	0	0	F5h
DPS1	0	0	1	1	1	1	1	1	F6h
STATUS1	1	1	1	1	0	0	0	0	F7h
EIP	0	0	0	0	0	0	0	0	F8h
P7	1	1	1	1	1	1	1	1	F9h
TL3	0	0	0	0	0	0	0	0	FBh
TH3	0	0	0	0	0	0	0	0	FCh
T3CM	0	0	0	0	0	0	0	0	FDh
SCON2	0	0	0	0	0	0	0	0	FEh
SBUF2	0	0	0	0	0	0	0	0	FFh

SPECIAL-FUNCTION REGISTERS

The DS80C400 has many unique features as compared to the standard 8052 microcontroller. These features are controlled by use of the SFRs located in the unused locations of the 8052 SFR map. While maintaining complete instruction set compatibility with the 8052, increased functionality is achieved with the DS80C400.

The description for each bit indicates its read and write access, as well as its reset state.

Port 4 (P4)

	7	6	5	4	3	2	1	0
SFR 80h	P4.7 A19	P4.6 A18	P4.5 A17	P4.4 A16	P4.3 $\overline{CE3}$	P4.2 $\overline{CE2}$	P4.1 $\overline{CE1}$	P4.0 $\overline{CE0}$
	RW-0	RW-0	RW-0	RW-0	RW-1	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

P4.7–0

Port 4 bit 7–0. This port is composed of eight pins that are user programmable as I/O, extended program memory chip enables, or extended address lines. The configuration of the eight pins is established through the programming of the port 4 control register (P4CNT). Following a reset, and if $\overline{EA}$ is low, P4.3–P4.1 are driven high and are assigned as chip enables; port pins P4.7–P4.4 and P4.0 are cleared to low state and are assigned as addresses and chip enable, respectively. Additional information on external memory interfacing is found in the port 4 control register SFR description and later sections of this user's guide supplement.

Programmable parallel port. When programmed to function as a general I/O port (through the P4CNT.7–P4CNT.0 in the port 4 control register), data written to the P4.7–P4.0 SFR bits results in setting the port I/O configuration, as well as setting the state on the corresponding port pin. A 1 written to a port 4 latch, previously programmed low (0), activates a high-current, one-shot pullup on the corresponding pin. This is followed by a static, low-current pullup, which remains on until the port is changed again. The final high state of the port pin is considered a pseudo-input mode, and can be easily overdriven from an external source. Port latches previously in a high-output state do not change, nor does the high-current one-shot fire when a 1 is loaded. Loading a 0 to a port latch results in a static, high-current pulldown on the corresponding pin. This mode is termed the I/O output state, since no weak devices are used to drive the pin. Port 4 pins, which have previously been assigned to function as an external memory interface (by the PCNT.7–PCNT.0 control bits), are not altered by a write to the port 4 SFR register.

Port 4 alternate function. Port 4 alternate function is established through the programming of the port 4 control register.

A19
Bit 7
Program/data memory address 19. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the A19 memory signal.

A18
Bit 6
Program/data memory address 18. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the A18 memory signal.

A17
Bit 5
Program/data memory address 17. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the A17 memory signal.

A16
Bit 4
Program/data memory address 16. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the A16 memory signal.

$\overline{CE3}$
Bit 3
Program memory chip enable 3. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the memory signal.

$\overline{CE2}$
Bit 2
Program memory chip enable 2. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the memory signal.

$\overline{CE1}$
Bit 1
Program memory chip enable 1. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the memory signal.

$\overline{CE0}$
Bit 0
Program memory chip enable 0. When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the memory signal.

Stack Pointer (SP)

	7	6	5	4	3	2	1	0
SFR 81h	SP.7	SP.6	SP.5	SP.4	SP.3	SP.2	SP.1	SP.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SP.7–0
Bits 7–0
Stack pointer. This stack pointer identifies current location of the stack. The stack pointer is incremented before every PUSH operation. This register defaults to 07h after reset. The reset value is used when the stack is in 8051 stack mode. When the 10-bit stack is enabled (SA = 1), this register is combined with the extended stack pointer (ESP: 9Bh) to form the 10-bit address.

Data Pointer Low 0 (DPL)

	7	6	5	4	3	2	1	0
SFR 82h	DPL.7	DPL.6	DPL.5	DPL.4	DPL.3	DPL.2	DPL.1	DPL.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPL.7–0
Bits 7–0
Data pointer low 0. This register is the low byte of the standard 8051 data pointer and contains the low-order byte of the 24-bit data address. The data pointer low byte 0 is cleared to 00h on all forms of reset.

Data Pointer High 0 (DPH)

	7	6	5	4	3	2	1	0
SFR 83h	DPH.7	DPH.6	DPH.5	DPH.4	DPH.3	DPH.2	DPH.1	DPH.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPH.7–0
Bits 7–0

Data pointer high 0. This register is the high byte of the standard 8051 data pointer and contains the middle-order byte of the 24-bit data address. The data pointer high byte 0 is cleared to 00h on all forms of reset.

Data Pointer Low 1 (DPL1)

	7	6	5	4	3	2	1	0
SFR 84h	DPL1.7	DPL1.6	DPL1.5	DPL1.4	DPL1.3	DPL1.2	DPL1.1	DL1H.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPL1.7–0
Bits 7-0

Data pointer low 1. This register is the low byte of auxiliary data pointer 1 and contains the low-order byte of the 24-bit data address. When the SEL1: 0 bits (DPS.1:0) are set to 01b, DPX1, DPL1 and DPH1 are used during DPTR operations. The data pointer low byte 1 is cleared to 00h on all forms of reset.

Data Pointer High 1 (DPH1)

	7	6	5	4	3	2	1	0
SFR 85h	DPH1.7	DPH1.6	DPH1.5	DPH1.4	DPH1.3	DPH1.2	DPH1.1	DPH1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPH1.7–0
Bits 7–0

Data pointer high 1. This register is the high byte of auxiliary data pointer 1 and contains the middle-order byte of the 24-bit data address. When the SEL1:0 bits (DPS1:0) are set to 01b, DPX1, DPL1 and DPH1 are used during DPTR operations. The data pointer high byte 1 is cleared to 00h on all forms of reset.

Data Pointer Select (DPS)

	7	6	5	4	3	2	1	0
SFR 86h	ID1	ID0	TSL	AID	SEL1	—	—	SEL
	RW-0	RW-0	RW-0	R-0	R-0	R-1	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

ID1, ID0
Bits 7-6

Increment/decrement function select. These bits define whether the INC DTPR instruction 7-6increments or decrements the active data pointer when DPTR1 or DPTR are selected by the SEL1, SEL bits.

ID1	ID0	SEL1, SEL = 00	SEL1, SEL = 01
0	0	Increment DPTR	Increment DPTR1
0	1	Decrement DPTR	Increment DPTR1
1	0	Increment DPTR	Decrement DPTR1
1	1	Decrement DPTR	Decrement DPTR1

TSL
Bit 5

Toggle select enable. When set, this bit allows the following five DPTR-related instructions to toggle the SEL bit, followingexecution of the instruction. When TSL = 0, DPTR-related instructions do not affect the state of the SEL bit. DPTR-related instructions are:

```
INC  DPTRz
MOV  DPTR, #data16
MOVC A, @A+DPTR
MOVX @DPTR, A
MOVX A, @DPTR
```

AID
Bit 4

Automatic increment/decrement enable. This bit allows three of the DPTR-related instructions to increment (or decrement) the content of the active DPTR, if enabled. The

actual function (increment or decrement) is dependent on the setting of the ID3, ID2, ID1, and ID0 bits. The active data pointer is incremented (or decremented) by 1 after execution of DPTR-related instruction when AID bit is set to logic 1. When AID is cleared to 0, a DPTR-related instruction does not affect the content of the active DPTR.

This option is affected by the following instructions:

```
MOVC A, @A+SPTR
MOVX @SPTR, A
MOVX A, @DPTR
```

Reserved
Bits 2, 1

Reserved. (Returns 10b when read.) These bits are needed to prevent carry passing through the other SFR bits in the register when using INC DPS to toggle the pointer selection.

SEL1, SEL
Bits 3, 0

Data pointer select 1, data pointer select. These bits select the active data pointer.

SEL1, SEL = 00: Use DPX, DPH and DPL as DPTR

SEL1, SEL = 01: Use DPX1, DPH1 and DPL1 as DPTR

SEL1, SEL = 10: Use DPX2, DPH2 and DPL2 as DPTR

SEL1, SEL = 11: Use DPX3, DPH3 and DPL3 as DPTR

Power Control (PCON)

	7	6	5	4	3	2	1	0
SFR 87h	SMOD_0	SMOD0	OFDF	ODFE	GF1	GF0	STOP	IDLE
	RW-0	RW-0	RW-0*	RW-0	RW-0	RW-0	RW-0	RW-0

*R = Unrestricted read, W = Unrestricted write, -n = Value after reset, * = See description*

SMOD_0
Bit 7

Serial port 0 baud-rate doubler enable. This bit enables/disables the serial baud-rate doubling function for serial port 0.

0 = Serial port 0 baud rate is defined by the baud-rate generation equation.

1 = Serial port 0 baud rate is double that defined by the baud-rate generation equation.

SMOD0
Bit 6

Framing error detection enable. This bit selects the function of the SCON0.7 and SCON1.7, and SCON2.7.

SMOD0 = 0: SCON0.7, SCON1.7, and SCON2.7 function as SM0 as defined for serial port control registers.

SMOD0 = 1: SCON0.7, SCON1.7, and SCON2.7 are converted to the framing error (FE) flag for the respective serial port.

OFDF
Bit 5

Oscillator fail-detect flag. When set, this bit indicates that the preceding reset was caused by the detection of the crystal oscillator frequency falling below approximately 30kHz, if OFDE = 1. OFDF bit must be cleared by software, and it is not altered by the crystal oscillator frequency falling below 30kHz when OFDE = 0. OFDF is not set when the processor forces the crystal to stop operation by the stop mode.

OFDE
Bit 4

Oscillator fail-detect enable. When the OFDE = 1, a system reset is generated any time the crystal oscillator frequency falls below approximately 30kHz. This bit does not force a reset when the oscillator is stopped by the software-enabled stop mode, or if the crystal is stopped when the processor is running from the internal ring oscillator. When the OFDE bit is cleared to logic 0, no reset is issued when the crystal falls below the 30kHz.

GF1
Bit 3

General-purpose user flag 1. This is a bit-addressable, general-purpose flag for software control.

GF0
Bit 2

General-purpose user flag 0. This is a bit-addressable, general-purpose flag for software control.

STOP
Bit 1

Stop mode select. Setting this bit stops program execution, halts the CPU oscillator and internal timers, and places the CPU in a low-power mode. This bit is cleared by a reset or any of the external interrupts and resumes normal program execution

IDLE
Bit 0

Idle mode select. Setting this bit stops program execution, but leaves the CPU oscillator, timers, serial ports, and interrupts active. This bit is cleared by a reset, or any of the external interrupts, and resumes normal program execution.

Timer/Counter Control (TCON)

	7	6	5	4	3	2	1	0
SFR 88h	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TF1
Bit 7

Timer 1 overflow flag. This bit indicates when timer 1 overflows its maximum count as defined by the current mode. This bit can be cleared by software and is automatically cleared when the CPU vectors to the timer 1 interrupt service routine.

0 = No timer 1 overflow has been detected.

1 = Timer 1 has overflowed its maximum count.

TR1
Bit 6

Timer 1 run control. This bit enables/disables the operation of timer 1.

0 = Timer 1 is halted.

1 = Timer 1 is enabled.

TF0
Bit 5

Timer 0 overflow flag. This bit indicates when timer 0 overflows its maximum count as defined by the current mode. This bit can be cleared by software and is automatically cleared when the CPU vectors to the timer 0 interrupt service routine, or by software.

0 = No timer 0 overflow has been detected.

1 = Timer 0 has overflowed its maximum count.

TR0
Bit 4

Timer 0 run control. This bit enables/disables the operation of timer 0.

0 = Timer 0 is halted.

1 = Timer 0 is enabled.

IE1
Bit 3

Interrupt 1 edge detect. This bit is set when an edge/level of the type defined by IT1 is detected. If IT1 = 1, this bit remains set until cleared in software or until the start of the external interrupt 1 service routine. If IT1 = 0, this bit inversely reflects the state of the $\overline{INT1}$ pin.

IT1
Bit 2

Interrupt 1 type select. This bit selects whether the $\overline{INT1}$ pin detects edge- or level-triggered interrupts.

0 = $\overline{INT1}$ is level triggered.

1 = $\overline{INT1}$ is edge triggered.

IE0
Bit 1

Interrupt 0 edge detect. This bit is set when an edge/level of the type defined by IT0 is detected. If IT0 = 1, this bit remains set until cleared in software or the start of the external interrupt 0 service routine. If IT0 = 0, this bit inversely reflects the state of the $\overline{INT0}$ pin.

IT0
Bit 0

Interrupt 0 type select. This bit selects whether the $\overline{INT0}$ pin detects edge- or level-triggered interrupts.

0 = $\overline{INT0}$ is level triggered.

1 = $\overline{INT0}$ is edge triggered.

Timer Mode Control (TMOD)

	7	6	5	4	3	2	1	0
SFR 89h	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

GATE
Bit 7

Timer 1 gate control. This bit enables/disables the ability of timer 1 to increment.

0 = Timer 1 clocks when TR1 = 1, regardless of the state of $\overline{INT1}$.

1 = Timer 1 clocks only when TR1 = 1 and $\overline{INT1}$ = 1.

C/$\overline{T}$
Bit 6

Timer 1 counter/timer select.

0 = Timer 1 is incremented by internal clocks.

1 = Timer 1 is incremented by pulses on T1 when TR1 (TCON.6) is 1.

M1, M0
Bits 5-4

Timer 1 mode select. These bits select the operating mode of timer 1.

M1	M0	Mode
0	0	Mode 0: 8 bits with 5-bit prescale
0	1	Mode 1: 16 bits
1	0	Mode 2: 8 bits with autoreload
1	1	Mode 3: Timer 1 is halted, but holds its count

GATE
Bit 3

Timer 0 gate control. This bit enables/disables the ability of timer 0 to increment.

0 = Timer 0 clocks when TR0 = 1, regardless of the state of $\overline{INT0}$.

1 = Timer 0 clocks only when TR0 = 1 and $\overline{INT0}$ = 1.

C/$\overline{T}$
Bit 2

Timer 0 counter/timer select.

0 = Timer incremented by internal clocks.

1 = Timer 1 is incremented by pulses on T0 when TR0 (TCON.4) is 1.

M1, M0
Bits 1-0

Timer 0 mode select. These bits select the operating mode of timer 0. When timer 0 is in mode 3, TL0 is started/stopped by TR0 and TH0 is started/stopped by TR1. Run control from timer 1 is then provided through the timer 1 mode selection.

M1	M0	Mode
0	0	Mode 0: 8 bits with 5-bit prescale
0	1	Mode 1: 16 bits
1	0	Mode 2: 8 bits with autoreload
1	1	Mode 3: Two 8-bit timers for timer 0; timer 1 is stopped.

Timer 0 LSB (TL0)

	7	6	5	4	3	2	1	0
SFR 8Ah	TL0.7	TL0.6	TL0.5	TL0.4	TL0.3	TL0.2	TL0.1	TL0.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TL0.7–0
Bits 7–0

Timer 0 LSB. This register contains the least significant byte of timer 0.

Timer 1 LSB (TL1)

	7	6	5	4	3	2	1	0
SFR 8Bh	TL1.7	TL1.6	TL1.5	TL1.4	TL1.3	TL1.2	TL1.1	TL1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TL1.7–0
Bits 7–0

Timer 1 LSB. This register contains the least significant byte of timer 1.

Timer 0 MSB (TH0)

	7	6	5	4	3	2	1	0
SFR 8Ch	TH0.7	TH0.6	TH0.5	TH0.4	TH0.3	TH0.2	TH0.1	TH0.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TH0.7–0

Bits 7–0

Timer 0 MSB. This register contains the most significant byte of timer 0.

Timer 1 MSB (TH1)

	7	6	5	4	3	2	1	0
SFR 8Dh	TH1.7	TH1.6	TH1.5	TH1.4	TH1.3	TH1.2	TH1.1	TH1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TH1.7–0

Bits 7–0

Timer 1 MSB. This register contains the most significant byte of timer 1.

Clock Control (CKCON)

	7	6	5	4	3	2	1	0
SFR 8Eh	WD1	WD0	T2M	T1M	T0M	MD2	MD1	MD0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

WD1, WD0
Bits 7-6

Watchdog timer mode select 1-0. These bits are used to select watchdog timeout periods for the watchdog timer function. The watchdog timer generates interrupt timeout at this periodic rate, when enabled. All watchdog timer reset timeouts follow the interrupt timeouts by 512 system clock cycles.

WD1	WD0	INTERRUPT TIMEOUT	RESET TIMEOUT
0	0	2^{17} system clocks	2^{17} + 512 system clocks
0	1	2^{20} system clocks	2^{20} + 512 system clocks
1	0	2^{23} system clocks	2^{23} + 512 system clocks
1	1	2^{26} system clocks	2^{26} + 512 system clocks

The system clock relates to the external clock as follows:

CLOCK MODE	EXTERNAL CLOCKS PER SYSTEM CLOCK
Frequency multiplier (4x)	0.25
Frequency multiplier (2x)	0.5
Divide-by-4	1
Power-management mode	256

T2M
Bit 5

Timer 2 clock select. This bit controls the division of the system clock that drives timer 2. This bit has no effect when the timer is in baud-rate generator or clock output modes. Clearing this bit to 0 maintains 8051 compatibility. This bit has no effect on instruction cycle timing.

0 = Timer 2 uses a divide-by-12 of the crystal frequency.

1 = Timer 2 uses a divide-by-4 of the system clock frequency.

T1M
Bit 4

Timer 1 clock select. This bit controls the division of the system clock that drives timer 1. Clearing this bit to 0 maintains 8051 compatibility. This bit has no effect on instruction cycle timing.

0 = Timer 1 uses a divide-by-12 of the crystal frequency.

1 = Timer 1 uses a divide-by-4 of the system clock frequency.

T0M
Bit 3

Timer 0 clock select. This bit controls the division of the system clock that drives timer 0. Clearing this bit to 0 maintains 8051 compatibility. This bit has no effect on instruction cycle timing.

0 = Timer 0 uses a divide-by-12 of the crystal frequency.

1 = Timer 0 uses a divide-by-4 of the system clock frequency.

MD2, MD1, MD0
Bits 2-0

Stretch MOVX select 2-0. These bits select the control timing for external MOVX instructions. All internal MOVX instructions to the 9kB x 8 RAM, as well as CAN 0 data memory registers, occur at the fastest two-machine cycle rate. The internal MOVX rate to the SRAM is not programmable.

MD2	MD1	MD0	STRETCH VALUE	MOVX DURATION
0	0	0	0	2 machine cycles
0	0	1	1	3 machine cycles (reset default)
0	1	0	2	4 machine cycles
0	1	1	3	5 machine cycles
1	0	0	4	9 machine cycles
1	0	1	5	10 machine cycles
1	1	0	6	11 machine cycles
1	1	1	7	12 machine cycles

Port 1 (P1)

	7	6	5	4	3	2	1	0
SFR 90h	P1.7 $\overline{INT5}$	P1.6 INT4	P1.5 $\overline{INT3}$	P1.4 INT2	P1.3 TXD1	P1.2 RXD1	P1.1 T2EX	P1.0 T2
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

P1.7–0
Bits 7–0

General-purpose I/O port 1. When serving as a general-purpose I/O port, all the pins have an alternative function as described later. P1.2–7 contains functions that are new to the 80C32 architecture. The timer 2 functions on pins P1.1-0 are available on the 80C32, but not on the 80C31. Each of the functions is controlled by several other SFRs. The associated port 1 latch bit must contain a logic 1 before the pin can be used in its alternate function capacity.

$\overline{INT5}$
Bit 7

External interrupt 5. A falling edge on this pin causes an external interrupt 5, if enabled.

INT4
Bit 6

External interrupt 4. A rising edge on this pin causes an external interrupt 4, if enabled.

$\overline{INT3}$
Bit 5

External interrupt 3. A falling edge on this pin causes an external interrupt 3, if enabled.

INT2
Bit 4

External interrupt 2. A rising edge on this pin causes an external interrupt 2, if enabled.

TXD1
Bit 3

Serial port 1 transmit. This pin transmits the serial port 1 data in serial port modes 1, 2, and 3, and emits the synchronizing clock in serial port mode 0.

RXD1
Bit 2

Serial port 1 receive. This pin receives the serial port 1 data in serial port modes 1, 2, and 3, and is a bidirectional data transfer pin in serial port mode 0.

T2EX
Bit 1

Timer 2 capture/reload trigger. A 1-to-0 transition on this pin causes the value in the T2 registers to be transferred into the capture registers, if enabled by EXEN2 (T2CON.3). When in autoreload mode, a 1-to-0 transition on this pin reloads the timer 2 registers with the value in RCAP2L and RCAP2H, if enabled by EXEN2 (T2CON.3).

T2
Bit 0

Timer 2 external input. A 1-to-0 transition on this pin causes timer 2 increment or decrement, depending on the timer configuration.

External Interrupt Flag (EXIF)

	7	6	5	4	3	2	1	0
SFR 91h	IE5	IE4	IE3	IE2	CKRY	RGMD	RGSL	BGS
	RW-0	RW-0	RW-0	RW-0	R-*	R-*	RW-*	RT-0

*R = Unrestricted read, W = Unrestricted write, T = Timed-access write only, -n = Value after reset, * = Bits 1, 2 and 3 are cleared to 000b by a power-on reset, but are unchanged by all other forms of reset.*

IE5
Bit 7

External interrupt 5 flag. This bit is set when a falling edge is detected on $\overline{\text{INT5}}$. This bit must be cleared manually by software. Setting this bit in software causes an interrupt if enabled. Please note that, when the EOWMI bit internal to the 1-Wire bus master is set to 1, the IE5 flag serves as the 1-Wire bus master interrupt flag.

IE4
Bit 6

External interrupt 4 flag. This bit is set when a rising edge is detected on INT4. This bit must be cleared manually by software. Setting this bit in software causes an interrupt, if enabled.

IE3
Bit 5

External interrupt 3 flag. This bit is set when a falling edge is detected on $\overline{\text{INT3}}$. This bit must be cleared manually by software. Setting this bit in software causes an interrupt, if enabled.

IE2
Bit 4

External interrupt 2 flag. This bit is set when a rising edge is detected on INT2. This bit must be cleared manually by software. Setting this bit in software causes an interrupt, if enabled.

CKRY
Bit 3

Clock ready. The CKRY bit indicates the status of the startup period delay used by the crystal oscillator and the crystal clock multiplier warmup period. CKRY = 0 indicates the startup delay is still counting. When the CKRY = 1, the counter has completed. This bit is cleared each time the CTM bit in the PMR register is changed from low to high to start the crystal multiplier. Once the CKRY is set, the lockout is removed on the CD1, CD0 bits to select the multiplied crystal clock as a system clock source. This status bit is also cleared each time the crystal oscillator is restarted when exiting stop mode.

RGMD
Bit 2

Ring mode status. This bit indicates the current clock source for the device. This bit is cleared to 0 after a power-on reset, and is unchanged by all other forms of reset.

0 = Device is operating from the external crystal or oscillator.

1 = Device is operating from the ring oscillator.

RGSL
Bit 1

Ring oscillator select. This bit selects the clock source following a resume from stop mode. Using the ring oscillator to resume from stop mode allows almost instantaneous startup. This bit is cleared to 0 after a power-on reset, and is unchanged by all other forms of reset. The state of this bit is undefined on devices that do not incorporate a ring oscillator.

0 = The device holds operation until the crystal oscillator has warmed up.

1 = The device begins operating from the ring oscillator and, when the crystal warmup is complete, it switches to the external clock source or oscillator.

BGS

Bandgap select. This bit enables/disables the bandgap reference during stop mode. Disabling the bandgap reference provides significant power savings in stop mode, but sacrifices the ability to perform a power-fail interrupt or power-fail reset while stopped. This bit can only be modified with a timed access procedure.

0 = The bandgap reference is disabled in stop mode, but functions during normal operation.

1 = The bandgap reference operates in stop mode.

Port 4 Control Register (P4CNT)

	7	6	5	4	3	2	1	0
SFR 92h	-	-	P4CNT.5	P4CNT.4	P4CNT.3	P4CNT.2	P4CNT.1	P4CNT.0
	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1

R = Unrestricted read, T = Timed-access write only, -n = Value after reset

P4CNT.7–0 **Port 4 control register.** P4CNT bits provide the configuration for the alternate addressing modes on port 4 and 6. These settings, in turn, establish the size of the external program memory that can be accessed. To prevent an unauthorized change in the external memory configuration, all writes to the P4CNT must use the timed-access function. Programming the bit combinations given in this section converts the designated port pins to I/O, address or chip enables. Once any bit combination containing a 1 is programmed into P4CNT.2-P4CNT.0, the corresponding port pins that are then assigned to chip enables are locked out from being programmed as I/O in the port 4 SFR. In a similar fashion, any bit combination containing a 1 programmed into P4CNT.5-P4CNT.3 locks out the corresponding port pins (P6.5-P6.4, P4.7–P4.4) assigned to addresses. This allows the normal use of the port SFR, without the concern that a byte write to the SFR alters any of the external chip enables or addresses. Following a reset, the P4CNT is set to FFh, which, in turn, assigns all of the port 4 and P6.5-4 pins to address bits and chip enables. This register should be programmed to reflect the actual system memory configuration.

Bit 7 **Reserved.**

Bit 6 **Reserved.**

P4CNT.5-P4CNT.3 **Port pin P6.5, P6.4, P4.7–4 configuration control bits for CEx.** Bits 5-0 configure the external memory control signals. P4CNT.5-3 determine whether specific P6 and P4 pins function as A21–A16 or I/O. The number of external address lines enabled establishes the range for each pro gram chip enable ($\overline{CE0\text{-}3}$) and data chip enable ($\overline{PCE0\text{-}3}$). When P4CNT.5-3 = 000b, $\overline{CE0}$–$\overline{CE3}$ are decoded on 32kB block boundaries.

PORT PIN FUNCTION							MAX MEMORY SIZE PER CEx
P4CNT.5-3	**P6.5**	**P6.4**	**P4.7**	**P4.6**	**P4.5**	**P4.4**	
000	I/O	I/O	I/O	I/O	I/O	I/O	32kB
001	I/O	I/O	I/O	I/O	I/O	A16	128kB
010	I/O	I/O	I/O	I/O	A17	A16	256kB
011	I/O	I/O	I/O	A18	A17	A16	512kB
100	I/O	I/O	A19	A18	A17	A16	1MB
101	I/O	A20	A19	A18	A17	A16	2MB
110 or 111	A21	A20	A19	A18	A17	A16	4MB

CE0–CE7 can be individually configured as program or program/data memory by the MCON and MCON1 SFRs. When CE0–CE7 are converted from program to program/data memory, PCE0–PCE3 are disabled if the corresponding data memory area is covered by CEx. The internally decoded range for each program chip enable (CE0–CE7) is established by the number of external address lines (A21–A16) enabled by the P4CNT.5–P4CNT.3 control bits. The following table outlines the assigned memory boundaries of each chip enable (CEx) as determined by the P4CNT.5-P4CNT.3 control bits. (The memory boundaries of each peripheral chip enable (PCEx) are determined by P6CNT.5-P6CNT.3.) Note that, when the external address bus is limited to A0–A15, the chip enables are internally decoded on a 32kB x 8 block boundary. This is to allow the use of the more common 32kB memories, as opposed to using a less common 64kB block size memory.

Program Memory Chip-Enable Boundaries

P4CNT.5-3	CE0	CE1	CE2	CE3	CE4	CE5	CE6	CE7
000	32K	32-64K	64-96K	96-128K	128-160K	160-192K	192-224K	224-256K
001	128K	128-256K	256-384K	384-512K	512-640K	640-768K	768-896K	896-1024K
010	256K	256-512K	512-768K	768-1024K	1024-1280K	1280-1536K	1536-1792K	1792-2048K
011	512K	.512-1M	1-1.5M	1.5-2M	2-2.5M	2.5-3M	3-3.5M	3.5-4M
100	1M	1-2M	2-3M	3-4M	4-5M	5-6M	6-7M	7-8M
101	2M	2-4M	4-6M	6-8M	8-10M	10-12M	12-14M	14-16M
110 or 111	4M	4-8M	8-12M	12-16M				

P4CNT.2-P4CNT.0

Port pin P4.3–P4.0 configuration control bits. P4CNT.2-0 determines whether specific P4 pins function as program chip-enable signals or I/O. The memory ranges for each $\overline{CEx}$ signal are determined by P4CNT.5-3. Note that, when the appropriate PDCEx bit (MCON.3–0) is set, the corresponding $\overline{CEx}$ pin functions as a combined program/peripheral chip enable, and the respective $\overline{PCE0}$–$\overline{PCE3}$ is disabled.

PORT 4 PIN FUNCTION

P4CNT.2-0	P4.3	P4.2	P4.1	P4.0
000	I/O	I/O	I/O	I/O
100	I/O	I/O	I/O	$\overline{CE0}$
101	I/O	I/O	$\overline{CE1}$	$\overline{CE0}$
110	I/O	$\overline{CE2}$	$\overline{CE1}$	$\overline{CE0}$
111	$\overline{CE3}$	$\overline{CE2}$	$\overline{CE1}$	$\overline{CE0}$

Data Pointer Extended Register 0 (DPX)

	7	6	5	4	3	2	1	0
SFR 93h	DPX.7	DPX.6	DPX.5	DPX.4	DPX.3	DPX.2	DPX.1	DPX.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPX.7–0
Bits 7–0

Data pointer extended register 0. This register contains the high-order byte of the extended 24-bit address for data pointer 0. This register is used only in the 24-bit paged and contiguous addressing modes. This register is not used for addressing the data memory in the 16-bit address ing ing mode and, therefore, can be utilized as a scratchpad SRAM register.

Data Pointer Extended Register 1 (DPX1)

	7	6	5	4	3	2	1	0
SFR 95h	DPX1.7	DPX1.6	DPX1.5	DPX1.4	DPX1.3	DPX1.2	DPX1.1	DPX1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPX1.7–0
Bits 7–0

Data pointer extended register 1. This register contains the high-order byte of the extended 24-bit address for auxiliary data pointer 1. This register is used only in the 24-bit paged and contiguous addressing modes. This register is not used for addressing the data memory in the 16-bit addressing mode and, therefore, can be utilized as a scratchpad SRAM register.

CAN 0 Receive Message Stored Register 0 (C0RMS0)

	7	6	5	4	3	2	1	0
SFR 96h	C0RMS0.7	C0RMS0.6	C0RMS0.5	C0RMS0.4	C0RMS0.3	C0RMS0.2	C0RMS0.1	C0RMS0.0
	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R = Unrestricted read, -n = Value after reset. The C0RMS0 is cleared to 00h on all forms of reset, including the reset established by the CRST bit.

C0RMS0.7–0 **CAN 0 receive message stored register 0.** The C0RMS0 bits indicate which message center (1–8) has successfully received and stored the last incoming message. The content of the C0RMS0 register is updated each time a new message is successfully received and stored. The contents of the C0RMS0 register are automatically cleared following each read of C0RMS0 by the microcon troller. A bit value 1 indicates that the assigned message center has successfully received and stored new data since the last read of the C0RMS0 register. A bit value 0 indicates that no new message has been successfully received and stored since the last read of the RMS0 register. No interrupts are asserted because of the C0RMS0 settings. This register works fully independent of the status bits in the CAN status register and the INTIN7–0 vector in the CAN interrupt register, as well as of the INTRQ bit in the CAN message control registers.

C0RMS0.7 **Message center 8, message received and stored.**
Bit 7

C0RMS0.6 **Message center 7, message received and stored.**
Bit 6

C0RMS0.5 **Message center 6, message received and stored.**
Bit 5

C0RMS0.4 **Message center 5, message received and stored.**
Bit 4

C0RMS0.3 **Message center 4, message received and stored.**
Bit 3

C0RMS0.2 **Message center 3, message received and stored.**
Bit 2

C0RMS0.1 **Message center 2, message received and stored.**
Bit 1

C0RMS0.0 **Message center 1, message received and stored.**
Bit 0

CAN 0 Receive Message Stored Register 1 (C0RMS1)

	7	6	5	4	3	2	1	0
SFR 97h	CORMS1.7	CORMS1.6	CORMS1.5	CORMS1.4	CORMS1.3	CORMS1.2	CORMS1.1	CORMS1.0
	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R = Unrestricted read, -n = Value after reset. The C0RMS0 is cleared to 00h on all forms of reset, including the reset established by the CRST bit.

C0RMS1.7–0 **CAN 0 receive message stored register 1.**

The C0RMS1 bits indicate which message center (9–15) has successfully received and stored the last incoming message. The content of the C0RMS1 register is updated each time a new message is successfully received and stored. The contents of the C0RMS1 register are automatically cleared following each read of C0RMS1 by the microcontroller. A bit value 1 indicates that the assigned message center has successfully received and stored new data since the last read of the C0RMS1 register. A bit value 0 indicates that no new message has been successfully received and stored since the last read of the RMS1 register. No interrupts are asserted because of the C0RMS1 set tings. This register works fully independent of the status bits in the CAN status register and the INTIN7–0 vector in the CAN interrupt register, as well as of the INTRQ bit in the CAN message con-trol registers.

C0RMS1.7 Bit 7 — **Reserved.**

C0RMS1.6 Bit 6 — **Message center 15, message received and stored.**

C0RMS1.5 Bit 5 — **Message center 14, message received and stored.**

C0RMS1.4 Bit 4 — **Message center 13, message received and stored.**

C0RMS1.3 Bit 3 — **Message center 12, message received and stored.**

C0RMS1.2 Bit 2 — **Message center 11, message received and stored.**

C0RMS1.1 Bit 1 — **Message center 10, message received and stored.**

C0RMS1.0 Bit 0 — **Message center 9, message received and stored.**

Serial Port 0 Control (SCON0)

	7	6	5	4	3	2	1	0
SFR 98h	SM0/FE_0	SM1_0	SM2_0	REN_0	TB8_0	RB8_0	TI_0	RI_0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SM0/FE-0 Bit 7 — **Serial port 0 mode bit 0.** (When SMOD0 is logic 0.) When SMOD0 is logic 1, it is the framing error flag that is set upon detection of an invalid stop bit and must be cleared by software. When SMOD0 is set, modification of this bit has no effect on the serial mode setting.

SM1-0 Bit 6 — **Serial port 0 mode bit 1.**

SM2_0 Bit 5 — **Serial port 0 mode bit 2.** Setting of this bit in mode 1 ignores reception if an invalid stop bit is detected. Setting this bit in mode 2 or 3 enables multiprocessor communications. This prevents the RI_0 bit from being set, and interrupt being asserted, if the 9th bit received is 0.

REN_0 Bit 4 — **Receiver enable.** This bit enable/disables the serial port 0 receiver shift register.

0 = Serial port 0 reception disabled.

1 = Serial port 0 receiver enabled (modes 1, 2, and 3). Initiate synchronous reception (mode 0).

TB8_0 Bit 3 — **9th transmission bit state.** This bit defines the state of the 9th transmission bit in serial port 0 modes 2 and 3.

RB8_0 Bit 2 — **9th received bit state.** This bit identifies that state of the 9th reception bit of received data in serial port 0 modes 2 and 3. When SM2_0 = 0, RB8_0 is the state of the stop bit in mode 1. RB8_0 is not used in mode 0.

TI_0 Bit 1 — **Transmitter interrupt flag.** This bit indicates that data in the serial port 0 buffer has been completely shifted out. In serial port mode 0, TI_0 is set at the end of the 8th data bit. In all other modes, this bit is set at the end of the last data bit. This bit must be cleared by software.

RI_0 Bit 0 — **Receiver interrupt flag.** This bit indicates that a byte of data has been received in the serial port 0 buffer. In serial port mode 0, RI_0 is set at the end of the 8th bit. In serial port mode 1, RI_0 is set after the last sample of the incoming stop bit subject to the state of SM2_0. In modes 2 and 3, RI_0 is set after the last sample of RB8_0. This bit must be cleared by software.

MODE	SM2	SM1	SM0	FUNCTION	LENGTH	PERIOD
0	0	0	0	Synchronous	8 bits	12 t_{CLK}
0	1	0	0	Synchronous	8 bits	4 t_{CLK}
1	x	1	0	Asynchronous	10 bits	Timer 1 or 2
2	0	0	1	Asynchronous	11 bits	64 t_{CLK} (SMOD_0 = 0)
2	0	0	1	Asynchronous	11 bits	32 t_{CLK} (SMOD_0 = 1)
2	1	0	1	Asynchronous (MP)	11 bits	64 t_{CLK} (SMOD_0 = 0)
2	1	0	1	Asynchronous (MP)	11 bits	34 t_{CLK} (SMOD_0 = 1)
3	0	1	1	Asynchronous	11 bits	Timer 1 or 2
3	1	1	1	Asynchronous (MP)	11 bits	Timer 1 or 2

Serial Data Buffer 0 (SBUF0)

	7	6	5	4	3	2	1	0
SFR 99h	SBUF0.7	SBUF0.6	SBUF0.5	SBUF0.4	SBUF0.3	SBUF0.2	SBUF0.1	SBUF0.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SBUF0.7–0 (Bits 7–0) **Serial data buffer 0.** Data for serial port 0 is read from or written to this location. The serial transmit and receive buffers are separate registers, but both are addressed at this location.

Bits 7–2 **Reserved.**

Extended Stack Pointer Register (ESP)

	7	6	5	4	3	2	1	0
SFR 9Bh	—	—	—	—	—	—	ESP.1	ESP.0
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

ESP.1-0

Bits 1-0 **Extended stack pointer.** These extended stack pointer bits are used with SP to form a 10-bit stack pointer to support the use of the 1kB of the internal data memory as stack memory. When SA = 1, any overflow of the SP from FFh to 00h increments the ESP by 1, and any under flow of the SP from 00h to FFh decrements the ESP by 1. The ESP register is not used as part of the stack pointer when the default stack memory location is selected (SA = 0), but is still read/write accessible. Relocating the internal 9kB SRAM through the IDM1: 0 bits does not alter the ability of the ESP and SP registers to properly access the internal memory. See MCON register for more detail.

Address Page Register (AP)

	7	6	5	4	3	2	1	0
SFR 9Ch	AP.7	AP.6	AP.5	AP.4	AP.3	AP.2	AP.1	AP.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

AP.7–0
Bits 7–0 **Address page register.** The address page register (AP) is a paging register, which is used with the 24-bit paged addressing mode to support extended 24-bit program and data addressing capabilities, and is fully compatible with the original 8052 16-bit addressing operation. The AP register and the higher-order byte of the program counter (PC23: 16) are cleared to 00 hex, following a system reset, to establish initial program execution in the first 64kB byte page (page 0). When the microcontroller is programmed to operate in the 24-bit paged addressing mode (AM1, AM0 = 01b), data programmed into the AP register is loaded into the program counter high-order byte when theprocessor executes an LJMP or LCALL instruction. Execution of any of these two instructions loads the AP into the high-order byte of the program counter (PC23: 16) to allow the program counter (PC) to drive address lines A23–A16 with the previous AP value at the same time as the lower 16 bits (A0–A15) of the PC are updated.

In this manner, software compiled using the standard 16-bit addressing scheme uses the contents of the AP to establish the page to which the program flow is to jump. The AP register can be loaded at any time prior to the execution of the above two instructions to establish the address vector, which is used when the LJMP or LCALL instruction is used to cross page boundaries. Note that the third byte of the program counter (PC23: 16) does not increment when the lower 16 bits in the lower two bytes of the PC roll over from FFFF hex to 0000 hex. PC23: 16 functions only as a holding register to issue high-order address (A23–A16) when the 24-paged addressing mode is enabled. All interrupts are handled by pushing the high byte of the program counter (PC23: 16) along with the standard push of the standard 16-bit program counter on to the stack before the hardware generated interrupt LCALL instruction is executed. The AP register is not altered during the interrupt and must be taken into consideration when doing another LJMP or LCALL instruction within the interrupt routine.

Typically, it is best to do a PUSH AP when entering the interrupt routine, and a POP AP when exit ing, if either LJMP or LCALL instructions are to be used inside the routine with a new page address assigned to the AP. The additional loading of PC23: 16 on to the stack results in one additional machine cycle during an interrupt and three bytes being stored on the stack. Following the execution of a RETI instruction, the processor automatically reloads the entire 24 value of the PC with the original address from the stack. Again, the RETI or RET requires one additional machine cycle when compared to the standard 16-bit address-only operation.

The address page register is not used with the PC when the AM0 and AM1 bits are programmed for either the 16-bit addressing or 24-bit contiguous addressing mode, but it is accessible as a general-purpose SFR register.

Address Control Register (ACON)

	7	6	5	4	3	2	1	0
SFR 9Dh	—	—	MROM	BPME	BROM	SA	AM1	AM0
	RT-1	RT-1	RT-0	RT-0	RT-X	RT-0	RT-0	RT-0

R = Unrestricted read, T = Timed access write only, -n = Value after reset. The address control register is cleared to 1100 x 000b on all forms of reset, but bit 3 is reset to 0 on power-on reset.

Bits 7-6 **Reserved.**

MROM
Bit 5

Merge ROM assignment. The MROM bit provides a software mechanism for mapping the lower 32kB internal ROM block to one of the two following address locations. The upper 32kB internal ROM block is always mapped to FF8000h–FFFFFFh of the program memory space.

MROM	LOWER 32kB ROM MEMORY LOCATION (HEX)
0	000000–007FFF (reset default)
1	FF0000–FF7FFF

BPME
Bit 4

Breakpoint mode enable. Setting this bit to 1 enables the software breakpoint mode. Once enabled, the processor can enter or exit the breakpoint mode by executing an A5h instruction. Clearing this bit to 0 disables the A5h instruction to the processor, and no breakpoint mode operation is allowed.

BROM
Bit 3

Bypass ROM. This bit determines whether the program flow is to start in the external user program or the internal ROM after a reset. A 0 forces the processor to start execution at location 000000h of internal ROM after a reset if the $\overline{EA}$ pin is connected high. A 1 forces the processor to start user program execution at location 000000h of the program memory after a reset if the $\overline{EA}$ pin is connected high. Connecting the $\overline{EA}$ pin to ground always forces the processor to start user program execution at location 000000h of the program memory after a reset, regardless of the logic state of the BROM bit. This bit is reset to 0 upon power-on. Changing this bit from a 0 to a 1 when the $\overline{EA}$ pin is connected high causes a reset immediately. Changing this bit from a 1 to a 0 has no immediate effect on the system function until a reset occurs.

SA
Bit 2

Extended stack address mode enable. Programming the SA bit to a 0 enables the standard 256 scratchpad SRAM bytes as the default stack. In this mode, the standard 8-bit stack pointer value is supplied by the SP register. ESP is not used in this mode. Programming the SA bit to a 1 enables the alternate use of 1kB of the internal data memory as the stack memory. In this mode, the 2 least significant bits of the ESP register are used as the two most significant bits of the 10-bit stack pointer.

AM1, AM0

Bits 1-0

Address mode control bits.

The AM0 and AM1 bits establish the addressing mode for the microcontroller.

AM1	AM0	ADDRESSING MODE
0	0	16-bit addressing mode (A23–A16 are locked to 00h)
0	1	24-bit paged addressing mode
1	x	24-bit contiguous addressing mode

Programming AM1 and AM0 to a 00 leaves the microcontroller in the traditional 8051 16-bit addressing mode. In this mode, the processor operates with a 16-bit address field with the higher-order program counter byte (PC23: 16) forced to 00h.

Programming AM1 and AM0 to a 01 enables the 24-bit paged addressing mode. In this mode, the processor operates with a 24-bit address field with the address page register (AP) functioning as the input source to load the high-order program counter byte (PC23: 16) during the execution of specific instructions.

Programming AM1 and AM0 to 10 or 11 enables the fully contiguous 24-bit program counter-addressing mode. In this mode, the processor addresses program memory with a full 24-bit program counter (A23–A0) and does not utilize the AP register as an input to the program counter. AP is converted into a general-purpose read/write SFR, and does not have any relationship to the program counter or address field. Note that AM1 and AM0 bits default to 00 on all resets, so the 24-bit contiguous address mode must be enabled before executing the following four instructions:

```
1) MOV DPTR, #data24
2) ACALL addr19
3) LCALL addr24
4) LJMP addr24
```

CAN 0 Transmit Message Acknowledgment Register 0 (C0TMA0)

	7	6	5	4	3	2	1	0
SFR 9Eh	C0TMA0.7	C0TMA0.7	C0TMA0.7	C0TMA0.7	C0TMA0.7	C0TMA0.7	C0TMA0.7	C0TMA0.7
	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R = Unrestricted read, -n = Value after reset. The C0TMA0 is cleared to 00h on all forms of reset, including the reset established by the CRST bit.

C0TMA0.7–0

CAN 0 transmit message acknowledgment register 0. The C0TMA0 bits indicate which message center (1–8) has successfully transmitted a message since the last read of the register. The contents of the C0TMA0 register are updated each time a new message is successfully transmit ted. The contents of the C0TMA0 register are automatically cleared following each read of C0TMA0 by the microcontroller. A bit value of 1 indicates that the assigned message center has been successfully transmitted since the last read of the C0TMA0 register. A bit value of 0 indicates that no new message has been successfully transmitted since the last read of the C0TMA0 register. Interrupts are not generated as a result of bits being set in the C0TM0 register. This register works fully independent of the status bits in the CAN status register, the INTIN7–0 vector in the CAN interrupt register, and the INTRQ bit in the CAN message control registers.

C0TMA0.7
Bit 7

Message center 8, message transmitted.

C0TMA0.6

Message center 7, message transmitted.

Bit 6

C0TMA0.5 **Message center 6, message transmitted.**
Bit 5

C0TMA0.4 **Message center 5, message transmitted.**
Bit 4

C0TMA0.3 **Message center 4, message transmitted.**
Bit 3

C0TMA0.2 **Message center 3, message transmitted.**
Bit 2

C0TMA0.1 **Message center 2, message transmitted.**
Bit 1

C0TMA0.0 **Message center 1, message transmitted.**
Bit 0

CAN 0 Transmit Message Acknowledgment Register 1 (C0TMA1)

	7	6	5	4	3	2	1	0
SFR 9Fh	—	C0TMA1.6	C0TMA1.5	C0TMA1.4	C0TMA1.3	C0TMA1.2	C0TMA1.1	C0TMA1.0
	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R = Unrestricted read, -n = Value after reset. The C0TMA0 is cleared to 00h on all forms of reset, including the reset established by the CRST bit.

C0TMA1.7–0 **CAN 0 transmit message acknowledgment register 1.** The C0TMA1 bits indicate which message center (1–8) has been successfully transmitted. The contents of the C0TMA1 register are updated each time a new message is successfully transmitted. The contents of the C0TMA1 register are automatically cleared following each read of C0TMA1 by the microcontroller. A bit value of 1 indicates that the assigned message center has been successfully transmitted since the last read of the C0TMA1 register. A bit value of 0 indicates that no new message has been successfully transmitted since the last read of the C0TMA1 register. The corresponding C0TMA1 bits are assigned to the following message centers. No interrupts are asserted because of the C0TMA1 settings. This register works fully independent of the status bits in the CAN status register and the INTIN7–0 vector in the CAN interrupt register, as well as of the INTRQ bit in the CAN message control registers (same as C0TMA0).

Bit 7 **Reserved.**

C0TMA1.6 **Message center 15, message transmitted.**
Bit 6

C0TMA1.5 **Message center 14, message transmitted.**
Bit 5

C0TMA1.4 **Message center 13, message transmitted.**

Bit 4

C0TMA1.3 **Message center 12, message transmitted.**
Bit 3

C0TMA1.2 **Message center 11, message transmitted.**
Bit 2

C0TMA1.1 **Message center 10, message transmitted.**
Bit 1

C0TMA1.0 **Message center 9, message transmitted.**
Bit 0

Port 2 (P2)

	7	6	5	4	3	2	1	0
SFR A0h	A15/P2.7	A14/P2.6	A13/P2.5	A12/P2.4	A11/P2.3	A10/P2.2	A9/P2.1	A8/P2.0
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

P2.7–0
Bits 7–0

Port 2. This port functions as an address bus during external memory access and as a general-purpose I/O port on devices that incorporate internal program memory. During external memory cycles, this port contains the MSB of the address. The only instructions to access the P2 SFR are MOVX A, @Ri and MOVX @Ri, A when port 2 is used as the MSB of an external address.

Port 5 (P5)

	7	6	5	4	3	2	1	0
SFR A1h	P5.7 $\overline{PCE3}$	P5.6 $\overline{PCE2}$	P5.5 $\overline{PCE1}$	P5.4 $\overline{PCE0}$	P5.3 —	P5.2 T3	P5.1 C0RX	P5.0 C0TX
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

P5.7–0
Bits 7–0

Port 5. This port can function as a programmable parallel I/O port, a CAN interface, timer 3 input, and/or peripheral enable signals. Data written to the port latch serves to set both logic level and direction of the data on the pin. A 1 written to a port latch, previously programmed low (0), activates a high-current, one-shot pullup on the corresponding pin. This is followed by a static, low-current pullup that remains on until the port is changed again. The final high state of the port pin is considered a pseudo-input mode, and can be easily overdriven from an external source. Port latches previously in a high-output state do not change, nor does the high-current one-shot fire when a 1 is loaded. Loading a 0 to a port latch results in a static, high-current pulldown on the corresponding pin. This mode is termed the I/O output state, since no weak devices are used to drive the pin.

Writes to P5.1–P5.0 are disabled when the P5CNT.3 bit in the port 5 control SFR is programmed to a 1. These bits read as a 1 when assigned to the CAN processor. The P5.2 latch bit must be set to 1 before the pin can be used for the alternate function of T3. The value of the port latch is not altered by a read operation, except the read-modify-write instructions that perform a read followed by a write. See P5CNT SFR (A2h) for more details.

$\overline{PCE3}$
Bit 7

Peripheral chip enable 3. When enabled by the P5CNT register, this pin asserts the fourth chip-enable signal.

$\overline{PCE2}$
Bit 6

Peripheral chip enable 2. When enabled by the P5CNT register, this pin asserts the third chip-enable signal.

$\overline{PCE1}$
Bit 5

Peripheral chip enable 1. When enabled by the P5CNT register, this pin asserts the second chip-enable signal.

$\overline{PCE0}$
Bit 4

Peripheral chip enable 0. When enabled by the P5CNT register, this pin asserts the first chip-enable signal.

Bit 3

—

T3
Bit 2

Timer/counter 3 external input. This pin functions as an external input to timer 3 when configured as such with the T3CM SFR. A 1-to-0 transition on this pin increments timer 3.

C0RX
Bit 1

CAN 0 receive. This pin is connected to the receive data-output pin of the CAN 0 transceiver device.

C0TX
Bit 0

CAN 0 transmit. This pin is connected to the transmit data-input pin of the CAN 0 transceiver device.

Bit 7 — **Reserved.** Read returns logic 1.

CAN0BA
Bit 6 — **CAN 0 bus active status.** The CAN0BA signal is a latched status bit that is set if the respective CAN 0 I/O-enabled (P5CNT.3) bit is set and bus activity is detected on the CAN 0 bus. Once activity is detected and the bit is set, it remains set until cleared by application software or a reset.

Bit 5 — **Reserved.** Read returns logic 0.

Bit 4 — **Reserved.** Read returns logic 0.

C0_I/O
Bit 3 — **CAN 0 I/O enable.** The P5CNT.3 bit configures P5.0 and P5.1 as either standard I/O or CAN receive input (P5.1–C0RX) and CAN transmit output (P5.0–C0TX). Programming P5CNT.3 to a 0 places P5.1 and P5.0 into the standard I/O mode. Programming P5CNT.3 to a 1 places P5.1 and P5.0 into the CAN transmit and receive mode. When P5CNT.3 is programmed to a 1, all I/O interaction through the port 5 SFR with P5.1 and P5.0 is disabled.

Port 5 Control Register (P5CNT)

	7	6	5	4	3	2	1	0
SFR A2h	—	CAN0BA	—	—	C0_I/O	P5CNT.2	P5CNT.1	P5CNT.0
	RW-1	RW-0	RW-0	RW-0	RW-0	RT-0	RT-0	RT-0

R = Unrestricted read, W = Unrestricted write, T = Timed Access Write Only, -n = Value after reset

P5CNT.2-P5CNT.0
Bits 2-0 — **Port pin P5.7–P5.4 configuration control bits.** Once any bit combination containing a 1 is programmed into P5CNT.2-P5CNT.0, the corresponding port pins that are then assigned to peripheral chip enable are locked out from being programmed as I/O in the port 5 SFR. The internally decoded range for each peripheral chip enable (PCE0–PCE3) is established by the number of external address lines (A19–A16), which are enabled by the P6CNT.5-P6CNT.3 control bits. This can be different than the program memory CE0–CE7 decoding. The following table outlines the assigned data memory boundaries of each chip enable as determined by the P6CNT.5-P6CNT.3 control bits. Note that, when the external address bus is limited to A0–A15, the chip enables are internally decoded on a 32kB x 8 block boundary. This is to allow the use of the more common memories, as opposed to using a less common 64kB block size memory.

PORT 5 PIN FUNCTION

P5CNT.2-0	P5.7	P5.6	P5.5	P5.4
000	I/O	I/O	I/O	I/O
100	I/O	I/O	I/O	$\overline{PCE0}$
101	I/O	I/O	$\overline{PCE1}$	$\overline{PCE0}$
110	I/O	$\overline{PCE2}$	$\overline{PCE1}$	$\overline{PCE0}$
111	$\overline{PCE3}$	$\overline{PCE2}$	$\overline{PCE1}$	$\overline{PCE0}$

The memory range addressable by each PCEx signal is a function of the total number of address lines (A19–A16) established by the P4CNT register. Note that the chip-enable range, when using A0–A15, is 32kB instead of the expected 64kB. This is to allow the use of more common 32kB memory devices rather than 64kB devices.

PORT 4 PIN FUNCTION

P4CNT.5-3	$\overline{PCE0}$	$\overline{PCE1}$	$\overline{PCE2}$	$\overline{PCE3}$
000	0–32kB	32–64kB	64–96kB	96–128kB
100	0–128kB	128–256kB	256–384kB	384–512kB
101	0–256kB	256–512kB	512–768kB	768kB–1MB
110	0–512kB	512–1MB	1–1.5MB	1.5–2MB
111	0–1MB	1–2MB	2–3MB	3–4MB

PCE0–PCE3 are internally decoded to data memory address block boundaries as determined by the P6CNT.5–P6CNT.3 control bits. When any of CE0–CE7 are converted from a program chip enable to program/data chip enables through the MCON and MCON1 registers, data memory areas assigned to PCE0–PCE3 are automatically disabled when the corresponding memory area is covered by CE0–CE7. Enabling merged program/data memory access under CE0–CE7 does not alter the port 5 control register bit states. Returning the CE0–CE7 enables back to the program memory automatically reenables the respective PCE0–PCE3 relationship.

CAN 0 Control Register (C0C)

	7	6	5	4	3	2	1	0
SFR A3h	ERIE	STIE	PDE	SIESTA	CRST	AUTOB	ERCS	SWINT
	RW-0	RW-0	RW-0	RW-0	RT-1	RW-0	RW-0	RT-1

R = Unrestricted read, W = Unrestricted write, T = Timed-access write only, -n = Value after reset

ERIE
Bit 7

CAN 0 error interrupt enable. Programming the ERIE bit to a 1 enables the CAN 0 status bus status (BSS) or error count greater than 96 bit (EC96) to issue an interrupt to the microcontroller, if the C0IE bit in the EIE SFR is also set. When ERIE is cleared to a 0, the error interrupt is disabled.

STIE
Bit 6

CAN 0 status interrupt enable. Programming the STIE bit to a 1 allows the CAN 0 status error bits (ER0-ER2), the transmit status bit (TXS), the receive status bit (RXS), or the wake-up status bit (WKS) to issue an interrupt to the microcontroller, if the C0IE bit in the EIE SFR is also set. When STIE is cleared to a 0, the status interrupt is disabled.

PDE
Bit 5

CAN 0 power-down enable. Programming the PDE bit to a 1 places the CAN 0 controller into a fully static power-down mode after completion of the last reception, transmission, or after the arbitration was lost or an error condition occurred. Note that the term 'after arbitration lost' denotes the fact the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. Programming PDE = 0 disables the power-down mode. The PDE mode forces all of the CAN 0 logic to a static state. The PDE mode can only be removed by either software reprogramming the PDE bit or through a system reset. A read of PDE establishes when the power-down mode has been enabled or removed as per the PDE bit. In all cases, the CAN controller begins operation after 11 recessive bits (a power-up sequence) on the CAN bus per the configuration settings for bit timing, which were programmed prior to entering the power-down mode. Since WKS reflects when the CAN has entered the low-power state, as per the SIESTA and/or PDE bit states, a read of the PDE bit establishes when the PDE bit is actually allowed to enable the low-power state. If the low-power state was previously enabled by setting the SIESTA bit, a read of PDE reflects the actual PDE bit value and not the low-power mode. If the low-power mode has not been previously enabled and the PDE bit is set to a 1 by software, a read of PDE returns a 0, until such time the PDE bit actually enables the low-power mode following an active transmit or receive operation. When the PDE and SIESTA bit are not used together, a read of the PDE bit, by default, also reflects the actual state of the low-power mode. Setting PDE does not alter any CAN block controls or error status relationships.

SIESTA
Bit 4

Low-power mode. Setting the SIESTA bit to a 1 places the CAN 0 controller into a low-power static state after completion of the last reception, transmission, or after the arbitration was lost or an error condition occurred. Note that the term 'after arbitration lost' denotes the fact the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. Programming SIESTA = 0 disables the low-power mode. The state of when the SIESTA mode is actually enabled or removed, as per the SIESTA bit programmed value, is reflected in the read of the SIESTA bit. The SIESTA mode is removed when the CAN 0 controller detects CAN 0 bus activity, by reprogramming the SIESTA bit to a 0, or by setting either CRST or SWINT to a 1. When the SIESTA bit is cleared by either a microcontroller write or activity on the CAN 0 bus, the CAN controller begins operation after 11 recessive bits on the CAN bus (after a power-up

sequence) using the configuration settings that were programmed prior to entering the power-down mode. Changing the SIESTA bit from a 0 to a 1 does not disrupt a currently active receive or transmit, but allows the completion of CAN 0 bus activity prior to entering into the static state. If the CAN 0 logic issues an interrupt as a result of an active CAN 0 receive or transmit while SIESTA is being set, the SIESTA bit is cleared, and the CAN 0 logic does not enter the low-power mode. Since WKS reflects when the CAN has entered the low-power state, as per the SIESTA and/or PDM bit states, a read of the SIESTA bit establishes when the SIESTA bit is actually allowed to enable the low-power state. If the low-power state was previously enabled by setting the PDM bit, a read of SIESTA reflects the actual SIESTA bit value, and not the low-power mode. If the low-power mode has not been previously enabled and the SIESTA bit is set to a 1 by soft ware, a read of SIESTA returns a 0 until such time that the SIESTA bit actually enables the low-power mode, following an active transmit or receive operation. When the PDE and SIESTA bit are not used together, a read of the SIESTA bit, by default, also reflects the actual state of the low-power mode. Setting SIESTA does not alter any CAN block controls or error status relationships. Note that the PDE and SIESTA bits act independent of each other. Setting both bits leaves the CAN processor in a low-power state until both bits have been cleared by their respective mechanisms.

CRST
Bit 3

CAN 0 reset. (Requires a timed-access write.) When CRST is set to a 1 and after completion of the last reception, transmission, or after arbitration was lost or an error condition occurred, all CAN registers located in the SFR memory map, with the exception of the CAN 0 control register are cleared to a 00 hex. The CAN 0 control register is set to 09 hex. Note that the term 'after arbitration lost' denotes the fact the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. In accordance with waiting until after the completion of the last reception, transmission, or after arbitration was lost or an error condition occurred, a read of the CRST bit, when previously programmed to a 1, returns a 0, until such time that the CRST = 1 state is actually allowed to place the CAN processor into the reset state. As such, a read of the CRST bit verifies when the CAN reset has been engaged or removed. CAN registers located in the MOVX memory map are left in the last state prior to setting CRST. Setting CRST also clears both the receive- and transmit-error counters in the CAN controllers and sets the SWINT bit to a 1. CRST must be cleared by software to remove the CAN reset and allow the CAN 0 processor to be initialized. When the CAN processor is not in a bus-off mode (BSS = 0) and the CAN processor exits either the software initialization mode (SWINT programmed from a 1 to a 0) or when the CAN reset is removed (CRST bit is cleared from a 1 to a 0 and the SWINT bit is cleared from 1 to 0), the CAN processor performs a power-up sequence of 11 consecutive recessive bits before the CAN controller enters into normal operation. If the CAN reset is removed and SWINT is left in the software initialization state, the microcontroller is allowed to immediately start programming the CAN registers and MOVX data memory prior to the completion of the power-up sequence. Exiting the software initialization mode (SWINT ≥ 0) requires a power-up sequence of 11 consecutive recessive bits before the CAN controller enters into normal operation. Clearing CRST to a 0 from a previous 0 state does not alter CAN processor operation. All writes to the CRST bit require a timed-access function.

AUTOB
Bit 2

Autobaud. When AUTOB is set to a 1, an internal loopback is enabled to AND the data from the external CAN bus with the transmitted data of the CAN 0 processor. The "ANDed' data is then connected to the internal input of the CAN 0 processor. At the same time, the transmitted data is disabled from reaching the external C0TX pin. The C0TX pin is placed into a recessive

state

when AUTOB = 1. The purpose of the internal loopback and the disabled C0TX pin is to allow the CAN processor to establish the proper CAN bus timing without disrupting the normal data flow between other nodes on the CAN bus. Disabling the C0TX pin and setting the C0TX pin to a recessive state prevents the CAN processor from driving nonsynchronized data onto the CAN bus (creating CAN bus errors to other nodes) when being programmed with various frequencies to synchronize the processor with the CAN bus. With AUTOB = 1, the microcontroller autobaud

algorithm makes use of the CAN 0 status register RXS and error status bits to determine when a message is successfully received (when AUTOB = 1, a successful receive, a store is not required). Each successive baud-rate attempt is proceeded by the microcontroller clearing the transmit- and receive-error counters by a write of 00 to the transmit-error SFR register and a read of the CAN 0 status register to clear the previous status-change interrupt. Note that a write to the transmit-error SFR register automatically resets the CAN fault confinement state machine to an initial (error-active) state if the error counters are cleared to 00 hex. If, however, the error counters are programmed to a value greater than 128, the CAN processor is in an error-passive state. Appropriate flags are set when the error counter is written with any value. A write of the status register is also used to remove the previous error value in the ER2–ER0 bits. Clearing the error counters also clears the EC96 bit, if set. When BSS = 1, the CAN processor locks out the ability for the microcontroller to write to the error counters by virtue of the fact that the SWINT bit is also forced to a 0 state during the period that the CAN processor performs a bus recovery and power-up sequence. Once the CAN processor has removed itself from the bus-off condition, it also clears BSS = 0, sets SWINT = 1, and clears both the transmit- and receive-error counters to 00 hex. Imagine a system with only two nodes on the CAN bus.

The following two situations are examples of how the autobaud function works on the CAN processor. In the first case, consider three nodes, A, B, and C, with nodes A and B operating in the normal CAN operational mode (nonautobaud) and node C (a DS80C400 CAN processor) attempting to establish a proper baud rate using the autobaud features. If node A transmits a message, node B acknowledges this message, and node C also receives the acknowledged message if it has the same baud rate. If node C does not have the same baud rate as nodes A and B, node C detects the mismatch by the respective error count. Node C then proceeds to adapt its baud rate and attempt to receive the following message.

In the second case, consider a system with only two nodes on the CAN bus. Consider node A in the autobaud mode and the second node on the bus in the normal CAN operational mode. Node B transmits a message and does not receive an acknowledgment, since there is no third node on the bus that is also properly synchronized with the bus and in the normal CAN operational mode. Once node B enters into an error-passive mode (after 16 repeated messages), it begins to send passive error flags. Note that, when node B is operating in an error-passive mode, it does not send any dominant errors flags to the bus. Once node A has established the proper baud rate, it receives the correct message. The internal autobaud loopback path also allows the passive acknowledgment error sent by node B to be "ANDed" with the dominant, internally transmitted acknowledgment bit from node A. As such, node A sees no errors, which establishes the fact that it is properly synchronized with the bus. Node A now exits out of the autobaud mode (AUTOB = 0) and enters into the normal CAN operational mode (with full transmit capability to the CAN bus). In this mode, node A then acknowledges the next message from node B.

ERCS
Bit 1

Error count select. The ERCS bit establishes in which level the error counters set or clear the EX96/128 bit in the CAN 0 status register. When ERCS = 0, the EC96/128 flag operates in an EC96 mode. In this mode, the EC96/128 bit is set to a 1 whenever the error count of either the transmit- or receive-error counters reach a level of 96 or greater. When ERCS = 1, the EC96/128 flag operates in an EC128 mode. In the EC128 mode, the EC96/128 flag is set to a 1 whenever the error count of either the transmit- or receive-error counters reach a level of 128 or greater.

SWINT
Bit 0

Software initialization. (Unrestricted read/write if BSS = 0, and read only if BSS = 1.) The SWINT bit establishes the initialization state for CAN 0, which disables CAN 0 bus activity to allow the processor to modify the MOVX SRAM assigned to the message centers without corrupt ing messages. When SWINT is set to 1 and after completion of the last reception or transmission, after arbitration was lost, or after an error condition occurred, all CAN 0 bus activity is disabled, allowing the processor to initialize any or all of the CAN 0 MOVX SRAM. Note that the term 'after arbitration lost' denotes the fact the arbitration was lost and the reception following this lost arbitration is completed. Recall that the CAN processor immediately becomes a receiver after it has lost its arbitration on the CAN bus. A read of the SWINT bit verifies when the CAN processor soft-

ware initialization mode has been engaged or removed. Although the transmit- and receive-error counters are not cleared when the SWINT bit is set, the CAN 0 transmit- and receive-error counters can be altered by software through the use of the CAN 0 transmit-error SFR register, as long as SWINT = 1. Setting SWINT to a 1 also clears the SIESTA bit independent of what is stored to the SIESTA bit location during or prior to the write of the C0C register. Clearing SWINT = 0 also hardware also disables the microcontroller from writing to the first 16 bytes of the CAN MOVX memory. These 16 locations make up the CAN 0 control/status/mask registers. When SWINT = 0, the microcontroller is allowed to write to any of the MOVX CAN register sites. All MOVX registers are readable at any time, independent of the SWINT bit. Also note that the SWINT bit does not alter the read or write access to any of the CAN 0 SFR registers or MOVX CAN message center registers. SWINT is programmed to a 0 when the processor has completed the MOVX SRAM initialization and CAN 0 bus activity has started. Software write access to the error counters is disabled when SWINT is cleared to a 0. A bus-off condition is caused by a high number of errors on the CAN bus. When a bus-off condition occurs, the CAN processor clears the SWINT bit to a 0 and immediately starts a bus recover and power-up sequence. During this time, the microcontroller is limited to only reading this bit. All microcontroller write access to SWINT is disabled when BSS = 1.

If the SWINT bit is set by a system reset, programming the CRST bit or setting the SWINT bit without the prior detection of a bus-off condition can cause an adverse condition. Clearing SWINT by software allows the CAN processor to synchronize itself to the CAN bus after the CAN processor executes a power-up sequence (11 recessive bits). The power-up sequence requires the CAN processor to detect 11 consecutive recessive bits. (In CAN protocol, this is termed a power-up sequence.) When SWINT = 0 by a bus-off condition, bus off forces the CAN processor to initiate a standard bus-off recovery sequence (128kB x 11 recessive bits). This is followed by entering into a reset state, requiring a power-up sequence (11 recessive bits), after which the CAN processor enters into the idle state (normal operation, BSS = 0) and sets the SWINT bit to a 1. This bit is not intended for use in changing data within the message centers after the CAN processor is placed into operation. Changes to the arbitration or data fields in the message centers should be done through the use of the MSRDY bit in the respective message (1–15) control registers. The SWINT bit is locked into the SWINT = 1 state until the bus timing registers are programmed to valid states. (The invalid states are 00 hex. See the CAN bus timing registers in the CAN control/status/mask registers.)

CAN 0 Status Register (C0S)

	7	6	5	4	3	2	1	0
SFR A3h	BSS	EC96/128	WKS	RXS	TXS	ER2	ER1	ER0
	R-0	R-0	R-0	RW-0	RW-0	RW-0	R-0	R-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

C0S.7–0

CAN 0 status register. The first three bits, BSS, EC96, and WKS, and the last 3 bits, ER2–ER0, in the CAN status register are read only by the microcontroller. The CAN processor sets or clears these flags (and interrupt sources) as defined by the system aspects associated with each bit. A CAN status register read clears the internal status-change interrupt flag. Unlike RXS and TXS, however, the individual mechanisms that set the ER2, ER0, BSS, EC96, and WKS bits do not reoccur without first being removed by the CAN processor. As a result, a new (0 ≥ 1) change by BSS, EC96, or (1 ≥ 0) change by WKS is required to set a new internal status change interrupt flag through these bits. In a similar fashion, a read of the CAN status register (which automatically sets ER2–ER0 to 111), followed by a new transmit or receive error, is required to set a new internal status change interrupt flag. If any one of these bits changes state from a previous 0 to a 1 (other than WKS, which changes from a 1 to a 0) and STIE is set to 1 with no other interrupt pending, the INTIN vector in the CAN interrupt register is set to 01 hex. If TXS or RXS is set to a 1 and a second message is successfully transmitted or received, and STIE is set to 1 while no other interrupt is pending, the INTIN vector in the CAN interrupt register is also set to 01 hex. If ER[2:0] changes from either a 000 or 111 binary state to any state other than 000 or 111, the INTIN vector in the CAN interrupt regis-

ter is also set to 01 hex. This issues a status change interrupt request if at least one of the following conditions is valid and no other interrupt is pending.

BSS
Bit 7

CAN 0 bus status. (Read only.)

The BSS bit reflects the current status of the CAN 0 bus. When BSS = 1, the CAN 0 bus is disabled (bus off) and is not capable of receiving or transmitting messages. This condition is the result of the transmit-error counter reaching a count of 256. When the CAN processor detects an error count of 256, the CAN processor automatically sets BSS = 1 and clears SWINT = 0. BSS is cleared to a 0 to enable CAN 0 bus activity when the CAN processor completes both the bus-off recovery (128kB x 11 consecutive recessive bits) and the power-up sequence (11 consecutive recessive bits). Once the CAN processor has completed this relationship, it sets SWINT = 1 and enters into the software initialization state. Once the microcontroller has cleared SWINT to a 0, the CAN processor is enabled to transmit and receive messages. BSS is set to a 1 whenever the transmit error counter for CAN 0 reaches the 256 limit. When BSS = 0, the CAN 0 bus is enabled to receive or transmit messages. A change in the state of BSS from a previous 0 to a 1 generates an interrupt if the ERIE, C0IE, and IE SFR register bits are set. All microcontroller writes to the SWINT bit are disabled when BSS = 1. Both the transmit- and receive-error counters are cleared to 00 hex when the bus-off condition is cleared by the CAN module and BSS is cleared to 0.

EC96/128
Bit 6

CAN 0 error count greater than 96/128 status. (Read only.) The EC96/128 bit operates in one of two modes. These two modes are determined by the state of the C0C.1 bit in the CAN 0 control register. Following a system or CAN reset, the C0C.1 bit is cleared to a 0, which in turn enables the EC96 mode.

C0C.1 = 0, EC96/128 = EC96. In this mode, when EC96/128 = 1, the interrupt flag indicates that either the CAN 0 transmit error counter or the CAN 0 receive error counter has reached an error count of 96, an exceptional high number of errors. EC96/128 = 0 indicates that the current transmit error counter and receive error counter both have an error count of less than 96. A change in the state of EC96/128 from a previous 0 to a 1 generates an interrupt if the ERIE, C0IE, and IE SFR register bits are set.

When C0C.1 is programmed to a 1, the EC96/128 bit is reconfigured into an EC128 bit flag mode.

C0C.1 = 1, EC96/128 = EC128. In this mode, when EC96/128 = 1, the interrupt flag indicates that either the CAN 0 transmit error counter or the CAN 0 receive error counter has reached an error count of 128, an exceptional number of errors. EC96/128 = 0 indicates that the current transmit error counter and receive error counter both have an error count of less than 128. A change in the state of EC96/128 from either a previous 0 to a 1 or from a previous 1 to a 0 generates an interrupt if the ERIE, C0IE and IE SFR register bits are set.

WKS
Bit 5

CAN 0 wake-up status. (Read only.) WKS = 0 indicates that the CAN 0 is not in a low-power mode. WKS = 1 indicates that CAN 0 is in a low-power mode, based on the setting of either the SIESTA bit or the power-down mode bit to a 1. Clearing both the SIESTA bit and power-down enable (PDE) bit forces the WKS bit to a 0. A change in the state of WKS from a previous 1 to a 0 generates an interrupt if the STIE, C0IE, and IE SFR register bits are set.

RXS

Receive status. The RXS bit functions in two modes. When the AUTOB bit is set to a Bit 4 1, RXS = 1 indicates that a message has been successfully received by CAN 0 since the last read of the CAN 0 status register. Note that this does not mean that the incoming message was or was not stored in a message center, but means that the message did not have any errors associated with it during the reception. Messages that are successfully received but are not stored do not pass the arbitration filtering tests required by the internal message centers. When the AUTOB bit is cleared to a 0, RXS = 1 indicates that a message has been both successfully received and stored in one of the message centers by CAN 0 since the last read of the CAN 0 status register.

RXS = 0 indicates that no message has been successfully received since the last read of the CAN 0 status register. RXS is set only by the CAN 0 logic and is not cleared by the CAN controller. It is cleared only by the microcontroller software, the CRST bit, or a system reset.

When the RXS bit (0 > 1) provides the interrupt source for an interrupt, the microcontroller is required to read the CAN status register to clear the internal status-change interrupt flag. This flag is seen externally by the presence of the 01 state in the CAN interrupt register. Once this flag is cleared, the 01 state in the CAN interrupt register is replaced with either the 00 state for no interrupts pending, or a lower-priority interrupt code related to one of the message centers. If a second successful reception is detected prior to or after the clearing of the RXS bit in the status register, a second status-change interrupt flag is set to allow a second interrupt to be issued. Each new successful reception generates an interrupt request independent of the previous state of the RXS bit, as long as the CAN status register has been read to clear the previous status-change interrupt flag. Note that if the microcontroller sets the RXS bit from a previous low, it generates an artificial status-change interrupt (STIE = 1).

Thus, if RXS is previously set to 0 and a reception was successful, RXS is set to 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to RXS when RXS was previously a 0, RXS is set to a 1 and an interrupt can be asserted if enabled. If RXS is previously set to 1 and a reception was successful, RXS stays set to a 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to RXS when RXS was previously a 1, RXS remains 1 and no interrupt is asserted.

TXS
Bit 3

Transmit status. TXS = 1 indicates that a message has been successfully transmitted by CAN 0 (error free and acknowledged) since the last read of the CAN 0 status register. TXS = 0 indicates that no message has been successfully transmitted since the last read of the CAN 0 status register. TXS is set only by the CAN 0 logic and is not cleared by the CAN controller, but is cleared only by the microcontroller software, the CRST bit, or a system reset.

When the TXS bit (0 > 1) provides the interrupt source for an interrupt, the microcontroller is required to read the CAN status register to clear the internal status-change interrupt flag (this flag is seen externally by the presence of the 01 state in the CAN interrupt register). Once this flag is cleared, the 01 state in the CAN interrupt register is replaced with either the 00 state for no interrupts pending or a lower-priority interrupt code related to one of the message centers. If a second successful transmission is detected prior to or after the clearing of the TXS bit in the status register, a second status-change interrupt flag is set to allow a second interrupt to be issued. Each new successful transmission generates an interrupt request independent of the previous state of the TXS bit, as long as the CAN status register has been read to clear the previous status-change interrupt flag. Note that, if the microcontroller sets the TXS bit from a previous low, it generates an artificial status-change interrupt (STIE = 1).

Thus, if TXS is previously set to 0 and a reception was successful, TXS is set to 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to TXS when TXS was previously a 0, TXS is set to a 1 and an interrupt can be asserted if enabled. If TXS is previously set to 1 and a reception was successful, TXS stays set to a 1 and an interrupt can be asserted if enabled. If the microcontroller writes a 1 to TXS when TXS was previously a 1, TXS remains 1 and no interrupt is asserted.

ER2-0
Bits 2-0

CAN 0 bus error status 2-0. The ER2–ER0 bits indicate the first type of error that is encountered within a CAN 0 bus frame. The following states outline the specific error type. The eighth state (111 binary) is automatically programmed into ER2–ER0, following a read of the CAN 0 status register to establish if there has been a change in an error condition when doing a future read of the CAN 0 status register. The status data (ER2–ER0) read by the processor must be analyzed or stored in a separate SRAM location, since the ER2–ER0 bits are automatically set to

the 111 state following a read. The 111 state remains in the register until a new frame is either transmitted or received, at which time the ER2–ER0 data is undated in relation to the associated transmit or receive message. The ER2–ER0 bits are read only. Any attempted write to these bits does not affect the bits or the interrupt relationship associated with their value.

The interrupt error represented by the ER2–ER0 bus error bits is updated following each reception or transmission. Since the stored error from one reception or transmission can be reproduced in the next attempted reception or transmission, a new interrupt is generated whenever a new error condition is detected. This occurs during a reception or transmission, as long as the previous error condition was removed by a read of the CAN 0 status register.

Thus, if ER2-ER0 is set to 000 or 111 and an error condition occurs, this error condition is stored in the ER2-ER0 bits. An interrupt request is made to the microcontroller whenever the ER2-ER0 values change from either a 000 or 111 binary state to any state other than 000 or 111. If a second error occurs prior to the microcontroller performing a read of the CAN status register, then the second error is not stored and the first error condition continues to reside in the ER2-ER0 bits. Once the CAN status register is read by the microcontroller, the error status bits are set to 111. If another error occurs after the microcontroller read of the CAN status register, the ER2-ER0 bits are updated with the new error condition.

If two errors come up at the same time, only the one with the higher priority (as in the following table) is shown. Priority 1 is the highest and 6 is the lowest priority. The format error is higher than the bit 1 error, since the format error is always a bit 1 error, but a bit 1 error is not necessarily a format error. The error value displayed is selected according to relevance, if the two errors occur at the same time. This is based on which error is the main error and which one is an accompanying error.

ER2	ER1	ER0	PRIORITY	ERROR CONDITIONS
0	0	0	N/A	No error in last frame
0	0	1	2	Bit stuff error
0	1	0	5	Format error
0	1	1	4	Transmit not acknowledged error
1	0	0	6 (lowest)	Bit 1 error
1	0	1	1 (highest)	Bit 0 error
1	1	0	3	CRC error
1	1	1	N/A	No change since last C0S read

The following is a description of error types:

Bit stuff error: The CAN controller detects more than five consecutive bits of an identical state in an incoming message.

Format error: A received message has the wrong format.

Transmit not acknowledged error: A data frame was sent and the requested node did not acknowledge the message.

Bit 1 error: The CAN attempted to transmit a message and that, when a recessive bit was transmitted, the CAN bus was found to have a dominant bit level. This error is not generated when the bit is a part of the arbitration field (identifier and remote retransmission request).

Bit 0 error: The CAN attempted to transmit a message and that, when a dominant bit was transmitted, the CAN bus was found to have a recessive bit level. This error is not generated when the bit is a part of the arbitration field. The bit 0 error is set each time a recessive bit is received during the period that the CAN processor is recovering from a bus-off recovery period.

CRC error: The calculated CRC of a received message does not match the CRC embedded in the message.

CAN 0 Interrupt Register (C0IR)

	7	6	5	4	3	2	1	0
SFR A5h	INTIN7	INTIN6	INTIN5	INTIN4	INTIN3	INTIN2	INTIN1	INTIN0
	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

INTIN7–0
Bits 7–0

CAN 0 interrupt indicator 7–0. The C0IR register indicates the status of the interrupt sources in the CAN 0 processor. The contents of C0IR indicate that no interrupt is pending (00 hex), if an interrupt is due to a change in the CAN 0 status register (01 hex), or if an interrupt has been generated from the successful reception or transmission of one of the 15 message centers (02–10 hex). The C0IR register is cleared to 00 hex following a reset.

To properly reflect the value of each interrupt source in the C0IR register, each source must be enabled by the respective interrupt enable. These include ERIE and/or STIE enable in the case of status-change-related interrupt (01) sources, and either the ETI or ERI enable for each message center interrupt (02–10 hex) source. The status values of the interrupt sources in C0IR do not, however, require setting either the EA or C0IE bits in the IE and EIE SFR registers.

There are two methods for verifying message center interrupts. One method uses the ETI/ERI interrupt enable in the CAN status register, and the other method uses the STIE interrupt enables within each CAN message control register.

STIE = 1. When a transmission or a reception by the corresponding message center was successfully completed, the status-change interrupt and the RXS/TXS bit are asserted. To understand how each bit in the status register acts as an interrupt source, review the descriptions of each bit in the status register. Note that a successful receive in relation to the RXS bit is dependent on the AUTOB bit (AUTOB = 1 is successful receive only, and AUTOB = 0 is successful receive and store). This is not the case with the following ERI relationship, in which a receive is considered successful only if the data was stored in the respective message center. The STIE interrupt method requires the microcontroller to poll each message center to establish the respective interrupt source following each status-change interrupt.

ETI = 1 and/or ERI = 1. When a successful transmission or a successful reception and store by the corresponding message center are completed, the interrupt is asserted according to its priority. This method relies on the hardwired priority of the message centers. Minimal microcontroller intervention is required.

Terms used in the following description:

Value A is the value that was indicated before and is not zero.

MCV (message center's value) is the interrupt indicator value, which corresponds to the message center that received or transmitted a message (i.e., 02 for MC15, 03 for MC1, etc.)

Description:

1A. STIE = 1 only (polling method: ETI = ERI 0) with no prior interrupt active:

CASE	STIE	CHANGE DETECTED IN BIT 5-0 OF C0S SFR?	INTIN VECTOR	INTRQ	CAN 0 INT
A	0	No	Value A or 0	Not affected	Inactive
B	0	Yes	Value A or 0	Not affected	Inactive
C	1	No	Value A or 0	Not affected	Inactive

It is important to note that additional changes in bits 4–0 (RXS, TXS) of the CAN 0 status register can be detected even if these bits have not been cleared by the microcontroller. The only requirement for the second status-change interrupt is for the microcontroller to read the CAN 0 status register in order to clear the previous interrupt. Multiple changes in the CAN 0 status register, which are read from the CAN 0 status register and occur without the microcontroller clearing the status-change interrupt, appear as one interrupt. The WKS bit is a read-only bit and is not altered by a write from the microcontroller, and the ER2-ER0 bits are automatically set to 111 following a read of the CAN status register.

Although not related to a successful transmission or reception, ERIE = 1 also enables a similar interrupt relationship when bits 6 or 7 are changed in the CAN status register, with ERIE = 1.

1B. ERIE = 1 with no prior interrupt active:

CASE	ERIE	CHANGE DETECTED IN BIT 7 OR 6 OF C0S SFR?	INTIN VECTOR	INTRQ	CAN 0 INT
A	0	No	Value A or 0	Not affected	Inactive
B	0	Yes	Value A or 0	Not affected	Inactive
C	1	No	Value A or 0	Not affected	Inactive
D	1	Yes	Value A or 0 > 1	Not affected	Active

2. ERI = 1 and/or ETI = 1 only (hardwired method: STIE = 0) with no prior interrupt active:

CASE	ERI	RECEPTION SUCCESSFUL?	INTIN VECTOR	INTRQ	CAN 0 INT
A	0	No	Value A or 0	0	Inactive
B	0	Yes	Value A or 0	0	Inactive
C	1	No	Value A or 0	0	Inactive
D	1	Yes	Value A or (MCV > INTIN)	1	Active

General Issues:

The INTIN vector value does not change when a new interrupt source becomes active and the previous one has not yet been acknowledged and removed (i.e., microcontroller read of CAN 0 status register or microcontroller clear of the appropriate INTRQ bit in the respective CAN 0 message control register), regardless of the fact that the new interrupt has a higher priority or not.

If two properly enabled interrupt sources become active at the same time, the interrupt of highest priority is indicated. For example, if a message center completes a successful transmission or reception and both STIE and ERI, ETI are set, the interrupt indicated by the INTIN7–0 vector is that of the status-change interrupt (i.e., INTIN07 = 01 hex and not the message center interrupt; i.e., INTIN7–0 = MCV).

RXS and TXS are always activated when a transmission or reception is successfully completed. These bits are reset by the microcontroller writing a 0 to them. Reading the CAN 0 status register removes only the INTIN7–0 = 01 hex vector, but does not clear these bits. These bits (RXS and TXS) can be set by either the CAN processor or microcontroller, but are never reset by the CAN controller.

The CAN 0 interrupt is active when an active-interrupt source is indicated in the interrupt vector INTIN7–0. Changes in the INTIN7–0 value from a previous 00 hex state indicate the interrupt source first detected by the CAN processor following the nonactive-interrupt state. The INTIN7–0 interrupt values displayed in C0IR remain in place until the respective interrupt source is removed, independent of other higher- or lower-priority interrupts that become active prior to clearing the currently displayed interrupt source. The CAN 0 interrupt to the microcontroller is not active when INTIN7–0 = 00 hex. In all the other cases, the interrupt line is asserted and the INTIN7–0 vector must be read to determine the current interrupt source.

When the current (INTIN7–0) interrupt source is cleared, INTIN7–0 is changed to reflect the next active interrupt with the highest priority. The status-change interrupt is asserted if there has been a change in the CAN 0 status register (if enabled by the appropriate ERIE and/or STIE bit) and the CAN status interrupt state is set. A message center interrupt is indicated if the INTRQ bit in the respective CAN message control register is set.

The priority of the next interrupt displayed is fixed. As an example, consider the case in which the current INTIN7–0 value is that of a message center interrupt. The current INTIN7–0 interrupt source is cleared (INTRQ = 0), and the status-change interrupt and another message center interrupt are both active. The next interrupt indicated by INTIN7–0 should be the status-change interrupt, which has a higher priority than that of the message center interrupt.

When the current INTIN7–0 interrupt indicated is a status interrupt, and the status register is read, the INTIN7–0 vector is changed to the next lowest INTIN7–0 value (which is the next highest priority) of the corresponding message center whose INTRQ bit is set to 1. During this time, the interrupt line to the microcontroller remains active. The microcontroller either does an RETI and then is forced back into the same interrupt routine by the active-interrupt line, or remains in the interrupt routine until the microcontroller has cleared all active-interrupt sources (INTIN7–0 = 00 hex).

An active message center interrupt is cleared by writing a 0 to the INTRQ bit in the respective CAN message control register. The interrupt line to the microcontroller goes to an inactive state, and the INTIN7–0 vector is reset to 00 hex, if no other interrupts are active and enabled.

Example case:

t<i>: moment in time

STIE = 1, ERI = 1, ETI = 1

t1: INTRQ[1] = 1, RXS = 1	INTIN = 1, interrupt line = active
t2: INTRQ[15] = 1, TXS = 1	INTIN = 1, interrupt line = active
t3: ERR[2:0] = 3'b101	INTIN = 1, interrupt line = active
t4: Begin processing interrupts by micro	INTIN = 1, interrupt line = active
t5: TXS = 1 ≥ 0	INTIN = 1, interrupt line = active
t6: RXS = 1 ≥ 0	INTIN = 1, interrupt line = active
t7: ERR[2:0] = 101 ≥ 111	INTIN = 2, interrupt line = active
t8: INTRQ[15] = 1 ≥ 0	INTIN = 3, interrupt line = active
t9: INTRQ[1] = 1 ≥ 0	INTIN = 0, interrupt line = inactive

The following are the values of the INTIN7–0 bits for each interrupt source along with the respective priority of each.

INTERRUPT SOURCE	INTIN7–0 HEX VALUE	INTERRUPT PRIORITY
No pending interrupt	00	N/A
CAN 0 status register	01	Highest = 1
Message 15	02	2
Message 1	03	3
Message 2	04	4
Message 3	05	5
Message 4	06	6
Message 5	07	7
Message 6	08	8
Message 7	09	9
Message 8	0A	10
Message 9	0B	11
Message 10	0C	12
Message 11	0D	13
Message 12	0E	14
Message 13	0F	15
Message 14	10	Lowest = 16

CAN 0 Receive-Error Register (C0RE)

	7	6	5	4	3	2	1	0
SFR A7h	CORE.7	CORE.6	CORE.5	CORE.4	CORE.3	CORE.2	CORE.1	CORE.0
	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

C0RE.7–0
Bits 7–0

CAN 0 receive-error register. The CAN 0 receive-error register provides a means of reading the CAN 0 receive-error counter. New values can be loaded into the receive error counter through the CAN 0 transmit error register. CORE is cleared to a 00 hex following all hardware resets and software resets enabled by the CRST bit in the CAN 0 control register.

Interrupt Enable (IE)

	7	6	5	4	3	2	1	0
SFR A8h	EA	ES1	ET2	ES0	ET1	EX1	ET0	EX0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

EA
Bit 7
Global interrupt enable. This bit controls the global masking of all interrupts except power-fail interrupt, which is enabled by the EPFI bit (WDCON.5).

0 = Disable all interrupt sources. This bit overrides individual interrupt mask settings.

1 = Enable all individual interrupt masks. Individual interrupts occur if enabled.

ES1
Bit 6
Enable serial port 1 interrupt. This bit controls the masking of the serial port 1 interrupt.
0 = Disable all serial port 1 interrupts.

1 = Enable interrupt requests generated by the RI_1 (SCON1.0) or TI_1 (SCON1.1) flags.

ET2
Bit 5
Enable timer 2 interrupt. This bit controls the masking of the timer 2 interrupt.
0 = Disable all timer 2 interrupts.

1 = Enable interrupt requests generated by the TF2 flag (T2CON.7).

ES0
Bit 4
Enable serial port 0 interrupt. This bit controls the masking of the serial port 0 interrupt.
0 = Disable all serial port 0 interrupts.

1 = Enable interrupt requests generated by the RI_0 (SCON0.0) or TI_0 (SCON0.1) flags.

ET1
Bit 3
Enable timer 1 interrupt. This bit controls the masking of the timer 1 interrupt.
0 = Disable all timer 1 interrupts.

1 = Enable all interrupt requests generated by the TF1 flag (TCON.7).

EX1
Bit 2
Enable external interrupt 1. This bit controls the masking of external interrupt 1.
0 = Disable external interrupt 1.

1 = Enable all interrupt requests generated by the $\overline{\text{INT1}}$ pin.

ET0
Bit 1
Enable timer 0 interrupt. This bit controls the masking of the timer 0 interrupt.
0 = Disable all timer 0 interrupts.

1 = Enable all interrupt requests generated by the TF0 flag (TCON.5).

EX0
Bit 0
Enable external interrupt 0. This bit controls the masking of external interrupt 0.
0 = Disable external interrupt 0.

1 = Enable all interrupt requests generated by the $\overline{\text{INT0}}$ pin.

Slave Address Register 0 (SADDR0)

	7	6	5	4	3	2	1	0
SFR A9h	SADDR0.7	SADDR0.6	SADDR0.5	SADDR0.4	SADDR0.3	SADDR0.2	SADDR0.1	SADDR0.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SADDR0.7–0
Bits 7–0
Slave address register 0. This register is programmed with the given or broadcast address assigned to serial port 0.

Slave Address Register 1 (SADDR1)

	7	6	5	4	3	2	1	0
SFR AAh	SADDR1.7	SADDR1.6	SADDR1.5	SADDR1.4	SADDR1.3	SADDR1.2	SADDR1.1	SADDR1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SADDR1.7–0
Bits 7–0

Slave address register 1. This register is programmed with the given or broadcast address assigned to serial port 1.

CAN 0 Message Center 1 Control Register (C0M1C)

	7	6	5	4	3	2	1	0
SFR ABh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

COM1C

Read/write access. MSRDY, ETI, ERI, and INTRQ are unrestricted read/write bits. EXTRQ is read/clear only. When T/R = 0, ROW is read only. When T/R = 1, TIH is unrestricted read/write. MTRQ is unrestricted read and can only be set to a 1 when written to by the microcontroller or by the CAN controller, in case of a remote frame reception in a transmit message center. A write of a 0 to MTRQ leaves the MTRQ bit unchanged. DTUP is unrestricted read. When T/R = 0, DTUP can only be cleared to a 0 when written to by the microcontroller. A write of a 1 to DTUP with T/R = 0 leaves the DTUP bit unchanged. DTUP is unrestricted read/write when T/R = 1.

MSRDY
Bit 7

CAN 0 message center 1 ready. (MSRDY is unrestricted read/write.) MSRDY is programmed by the microcontroller to notify the CAN 0 logic when the associated message is ready for communication on the CAN 0 bus. When MSRDY = 0, the CAN 0 processor does not access this mes sage center for transmissions or to receive data or remote frame requests. MSRDY = 1 indicates the message is ready for communication, and MSDRY = 0 indicates either that the associated mes sage is not configured for use or that it is not required at the present time. This bit is used by the microcontroller to prevent the CAN 0 logic from accessing a message while the microcontroller is updating message attributes. These include as identifiers: arbitration registers 0–3, data byte reg isters 0–7, data byte count (DTBYC3, DTBYC0), direction control (T/R), the extended or standard mode bit (EX/ST), and the mask enables (MEME and MDME) associated with message 1. MSRDY is cleared to a 0 following a microcontroller hardware reset or a reset generated by the CRST bit in the CAN 0 control register, and must also remain in a cleared mode until all the CAN 0 initialization has been completed. Individual message MSRDY controls can be changed after initialization to reconfigure specific messages, without interrupting the communication of other messages on the CAN 0 bus.

ETI
Bit 6

CAN 0 message center 1 enable transmit interrupt. (ETI is unrestricted read/write.) When ETI is cleared to 0, a successful transmission does not set INTRQ and, as such, does not generate an interrupt. Setting ETI to a 1 enables a successful CAN 0 transmission to set the INTRQ bit, which in turn issues an interrupt to the microcontroller. Note that the ETI bit located in message center 15 is ignored by the CAN processor, since the message center 15 is a receive-only message center.

ERI
Bit 5

CAN 0 message center 1 enable receive interrupt. (ERI is unrestricted read/write.) When ERI is cleared to 0, a successful reception does not set the INTRQ and, as such, does not generate an interrupt. When the ERI is set to a 1, the INTRQ bit is only set when the CAN processor successfully receives and stores the incoming message into one of the message centers. Setting INTRQ, in turn, issues an interrupt request to the microcontroller.

INTRQ
Bit 4

Interrupt request. (INTRQ is unrestricted read/write.) INTRQ is automatically set to a 1 by the CAN 0 logic when the ERI is set and the CAN 0 logic completes a successful reception and store. The INTRQ bit is also set to a 1 when the ETI is set and the CAN 0 logic completes a successful transmission. The INTRQ interrupt request must also be enabled by the EA global mask in the IE SFR register if the interrupt is to be acknowledged by the microcontroller interrupt logic.

EXTRQ
Bit 3

External transmit request. (Read/clear only.) When EXTRQ is cleared to a 0, there are no pending requests by external CAN nodes for this message. When EXTRQ is set to a 1, a request has been made for this message by an external CAN node, but the service request has not been completed by the CAN 0 controller at the time of the read of EXTRQ. Following the completion of a requested transmission by a message center programmed for transmission (T/R = 1), the EXTRQ bit is cleared by the CAN 0 controller. A remote request is only answered by a message center programmed for transmission (T/R = 1) when DTUP = 1 and TIH = 0 (i.e., when new data was loaded and is not being currently modified by the micro). Note that a message center programmed for a receive mode (T/R = 0) also detects a remove frame request and sets the EXTRQ bit in a similar manner, but does not automatically transmit a data frame and, as such, does not automatically clear the EXTRQ bit.

MTRQ
Bit 2

Microcontroller transmit request. MTRQ is unrestricted read and can only be set to a 1 when written to by the microcontroller. A write of a 0 to MTRQ leaves the MTRQ bit unchanged. MTRQ can only be cleared as a result of a successful transmission by the respective message center, or when the CRST bit is set or the CAN processor experiences a system reset from the reset sources outlined in the functional description in the reset option and reset timing section of this user's guide supplement.

The MTRQ is a read-limited write bit, and is designed to allow the microcontroller to request a message to be transmitted. MTRQ is programmed to a 1 when the microcontroller is requesting the respective message to be transmitted. MTRQ remains set until such time that the message transmission is successfully completed, at which time the CAN 0 controller clears the MTRQ bit. Setting MTRQ with T/R = 1 (directional = transmit) results in the sending of a data frame for the transmitted message, and setting MTRQ with T/R = 0 (directional = receive) results in the sending of a remote frame request. When the associated message is programmed for transmit (T/R = 1), the MTRQ bit is also set by the CAN 0 controller at the same time that the EXTRQ bit is set by a message request from an external node. MTRQ is cleared by the CAN 0 controller at the same time as the EXTRQ bit, once a successful transmission of the message is completed. Note that the MTRQ bit located in message center 15 is ignored by the CAN processor, since the message center 15 is a receive-only message center.

ROW/TIH
Bit 1

Receive overwrite/transmit inhibit. The receive overwrite (ROW) and transmit inhibit (TIH) bits share the same bit 1 location in the CAN 0 message control register. The ROW function is only support ed when the associated message is programmed by the T/R = 0 bit in the message format register to function in the receive mode. Similarly, the TIH function is only supported when the associated message is programmed by the T/R = 1 bit in the message format register to function in the transmit mode.

Receive overwrite. (T/R = 0, ROW is read only.) The ROW is automatically set to a 1 by the CAN 0 controller if a new message is received and stored while the DTUP bit is still set. When set, ROW indicates that the previous message was potentially lost and may not have been read, since the microcontroller had not cleared the DTUP bit prior to the new load. When ROW = 0, no new message has been received and stored while DTUP was set to 1 since this bit was last cleared. Note that the ROW bit is not set when the WTOE bit is cleared to a 0, since all overwrites are disabled. Thus, if the incoming message matches the respective message center and DTUP = 1 in the respective message center, the combination of WTOE = 0 and DTUP = 1 forces the CAN processor to ignore the respective message center when the CAN is processing the incoming data.

ROW is cleared by the CAN processor when the microcontroller clears the DTUP bit associated with the same message center. It must be pointed out that the ROW bit for message center 15 is related to the overwrite of the buffer associated with message center 15, as opposed to the actual message center 15. ROW reflects the actual message center relationships for message centers 1–14. The ROW bit for the message center 15 shadow buffer is cleared, once the shadow buffer is loaded into the message center 15 and the shadow buffer is cleared to allow a new message to be loaded. The shadow buffer is automatically loaded

into message center 15 when the microcontroller clears the DTUP and EXTRQ bits in message center 15.

Transmit inhibit. (T/R = 1, TIH is unrestricted read/write.) The TIH allows the microcontroller to disable the transmission of the message when the data contents of the message are being updated. TIH = 1 directs the CAN 0 controller not to transmit the associated message. TIH = 0 enables the CAN 0 controller to transmit the message. If TIH = 1, EXTRQ is set to a 1 when a remote frame request is received by the message center. Following the remote frame request and after the microcontroller has established the proper data to be sent, the microcontroller clears the TIH bit to a 0, which allows the CAN processor to send the data requested by the previous remote frame request. Note that the TIH bit located in message center 15 is ignored by the CAN processor, since the message center 15 is a receive-only message center.

DTUP
Bit 0

Data updated. (DTUP is unrestricted read.) When T/R = 0, DTUP can only be cleared to a 0 when written by the microcontroller. A write of a 1 to DTUP with T/R = 0 leaves the DTUP bit unchanged. A write of a 1 to DTUP with T/R = 1 leaves the MTRQ bit unchanged. DTUP is unrestricted read/write when T/R = 1.

The DTUP bit has a dual function depending on whether a message is configured for transmit or receive by the T/R bit in the CAN 0 message format register. The DTUP bit is set to a 1 by either the microcontroller (when in transmit), or by the CAN 0 controller (when in receive), to signify that new data has been loaded into the data portion of the message.

Transmission mode (T/R = 1). The microcontroller sets TIH = 1 and clears DTUP = 0 prior to doing an update of the associated message center. This prevents the CAN processor from transmitting the data while the microcontroller is updating it. Once the microcontroller has finished configuring the message center, the microcontroller clears TIH = 0 and sets MSRDY = 1, MTRQ = 1 and DTUP = 1 to enable the CAN processor to transmit the data.The CAN processor does not clear the DTUP after the transmission, but the microcontroller is able to determine that the transmission has been completed by checking the MTRQ bit, which is cleared (MTRQ = 0) after the transmission has been successfully completed.

Receive mode (T/R = 0). The CAN processor sets the DTUP bit when it has completed a successful reception and storage of the incoming message to the respective message center. The CAN processor does not clear the DTUP after the microcontroller has read the associated data. That function is left to the microcontroller.

When operating in the receive mode (T/R = 0), the DTUP = 1 signal notifies the microcontroller that the respective message center has new data to be read by the microcontroller. The DTUP bit is used in two ways when doing the read of the message center, as determined by the WTOE bit in the CAN 0 message 1 arbitration register 3 (C0M1AR3).

When WTOE = 1 and the CAN processor is allowed to perform overwrites of respective message centers, the microcontroller uses the DTUP bit to establish the validity of each message read. Clearing DTUP = 0 before a read of a receive message center and then reading the DTUP bit after finishing the message center read, the microcontroller can determine if new data was loaded (DTUP = 1) or not (DTUP = 0) into the message center during the microcontroller read of the message center.

If DTUP = 1, then there was new data stored to the message center while the microcontroller was performing the message center read. This status condition requires the microcontroller to again clear the DTUP bit and perform a second read of the message center to verify that the data it reads is completely updated.

If DTUP = 0, the message center data read by the microcontroller had not been updated while it was being read by the microcontroller, and the data is complete.

When WTOE = 0 and the CAN processor is not allowed to perform overwrites of respective message centers, the microcontroller only needs to clear DTUP = 0 after performing the read of the message center. The CAN processor is not allowed to write into a message center where the DTUP = 1 state exists.

The DTUP bit is never cleared by the CAN processor, but is set as per the above discussion. The only mechanism used to clear the DTUP bit is the microcontroller or a system reset or the setting of the CRST bit.

When T/R = 1, all message center transmissions are automatically disabled until both DTUP = 1 and TIH = 0. This mechanism prevents the CAN from sending incomplete data.

Remote frame transmissions are not affected by the TIH bit in the receive mode (T/R = 0), since this function does not exist in this mode. In a similar fashion, the state of the DTUP bit does not inhibit remote frame request transmissions in the receive mode. The only gating item for remote frame transmissions in the receive mode (T/R = 0) is the setting of both the MSRDY = 1 and MTRQ = 1 bits.

CAN 0 Message Center 2 Control Register (C0M2C)

	7	6	5	4	3	2	1	0
SFR ACh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M2C Bits 7–0 — Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh) SFR. Consult the description of that register for more information.

CAN 0 Message Center 3 Control Register (C0M3C)

	7	6	5	4	3	2	1	0
SFR ADh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M3C Bits 7–0 — Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

CAN 0 Message Center 4 Control Register (C0M4C)

	7	6	5	4	3	2	1	0
SFR AEh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M4C Bits 7–0 — Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

CAN 0 Message Center 5 Control Register (C0M5C)

	7	6	5	4	3	2	1	0
SFR AFh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M5C Bits 7–0 — Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

Port 3 (P3)

	7	6	5	4	3	2	1	0
SFR B0h	P3.7 $\overline{RD}$	P3.6 $\overline{WR}$	P3.5 T1	P3.4 T0	P3.3 $\overline{INT1}$	P3.2 $\overline{INT0}$	P3.1 TXD0	P3.0 RXD0
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

P3.7–0
Bits 7–0
Purpose I/O port 3. This register functions as a general-purpose I/O port. In addition, all the pins have an alternative function listed as follows. Several other SFRs control each of the functions. The associated port 1 latch bit must contain a logic 1 before the pin can be used in its alternate function capacity.

$\overline{RD}$
Bit 7
External data memory read strobe. This pin provides an active-low read strobe to an external memory device.

$\overline{WR}$
Bit 6
External data memory write strobe. This pin provides an active-low write strobe to an external memory device

T1
Bit 5
Timer/counter external input. A 1-to-0 transition on this pin increments timer 1.

T0
Bit 4
Counter external input. A 1-to-0 transition on this pin increments timer 0.

$\overline{INT1}$
Bit 3
External interrupt 1. A falling edge/low level on this pin causes an external interrupt 1 if enabled.

$\overline{INT0}$
Bit 2
External interrupt 0. A falling edge/low level on this pin causes an external interrupt 0 if enabled.

TXD0
Bit 1
Serial port 0 transmit. This pin transmits the serial port 0 data in serial port modes 1, 2, and 3 and emits the synchronizing clock in serial port mode 0.

RXD0
Bit 0
Serial port 0 receive. This pin receives the serial port 0 data in serial port modes 1, 2, and 3 and is a bidirectional data transfer pin in serial port mode 0.

Port 6 (P6)

	7	6	5	4	3	2	1	0
SFR B1h	P6.7 TXD2	P6.6 RXD2	P6.5 A21	P6.4 A20	P6.3 $\overline{CE7}$	P6.2 $\overline{CE6}$	P6.1 $\overline{CE5}$	P6.0 $\overline{CE4}$
	R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

P6.7–0
Bits 7–0
Parallel I/O port 6. Any port 6 pin assigned to function as an external memory interface through the port 6 control register cannot be altered by a write to the port 6 SFR. All bits assigned a standard I/O are programmed as per the data value. In addition, all pins have an alternate function listed as follows. A read of a bit assigned to function as an external memory interface reads back a 1 when read by the port 6 SFR. A read of a bit assigned to standard I/O produces the value of the respective port 6 pin when that port pin was previously programmed with a 1 pseudo-input state or previously programmed as a 0 output state. Note that the use of read-modify-write instructions on ports 1, 2, 3, 4, 5, 6, and 7 on the DS80C400 read the state of the port latch, as opposed to the port pin data. These instructions are outlined in the *High-Speed Microcontroller User's Guide*.

TXD2
Bit 7
Serial port 2 transmit. This pin transmits the serial port 2 data in serial port modes 1, 2, and 3 and emits the synchronizing clock in serial port mode 0.

RXD2
Serial port 2 receive. This pin receives the serial port 2 data in serial port modes 1, 2, and 3 and is a

Bit 6 bidirectional data transfer pin in serial port mode 0.

A21 Bit 5 **Program/data memory address 21.** When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the A21 memory signal.

A20 Bit 4 **Program/data memory address 20.** When this bit is set to logic 1 and the P4CNT register is configured correctly, the corresponding device pin represents the A20 memory signal.

$\overline{CE7}$ Bit 3 **Program memory chip enable 7.** When this bit is set to logic 1 and the P6CNT register is configured correctly, the corresponding device pin represents the CE7 memory signal.

$\overline{CE6}$ Bit 2 **Program memory chip enable 6.** When this bit is set to logic 1 and the P6CNT register is configured correctly, the corresponding device pin represents the CE6 memory signal.

$\overline{CE5}$ Bit 1 **Program memory chip enable 5.** When this bit is set to logic 1 and the P6CNT register is configured correctly, the corresponding device pin represents the CE5 memory signal.

$\overline{CE4}$ Bit 0 **Program memory chip enable 4.** When this bit is set to logic 1 and the P6CNT register is configured correctly, the corresponding device pin represents the CE4 memory signal.

Port 6 Control Register (P6CNT)

	7	6	5	4	3	2	1	0
SFR B2h	—	—	P6CNT.5	P6CNT.4	P6CNT.3	P6CNT.2	P6CNT.1	P6CNT.0
	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1

R = Unrestricted read, T = Timed-access write only, -n = Value after reset

P6.7–0 **Port 6 control register.** P6CNT bits provide the configuration for the alternate addressing modes on port 6. These settings, in turn, establish the size of the external program memory that can be accessed. Programming the bit combinations given in this section converts the designated port 6 pins to I/O, address, or chip enables. Once any bit combination containing a 1 is programmed into P6CNT.2–P6CNT.0, the corresponding port pins that are then assigned to peripheral chip enables are locked out from being programmed as I/O in the port 6 SFR. In a similar fashion, any bit combination containing a 1 programmed into P6CNT.5–P6CNT.3 locks out the corresponding port pins assigned to addresses for respective peripheral chip enables. This allows the normal use of the port 6 SFR, without the concern that a byte write to the SFR would alter either any of the external chip enables or addresses.

Bit 7 **Reserved.**

Bit 6 **Reserved.**

P6CNT.5–P6CNT.3 **Port pin P4.7–P4.4 configuration control bit for PCEx.** Note that setting these bits to values other than those listed in the following table causes them to be treated as value of 000b and specifies peripheral memory chip size to 32kB. The peripheral chip enables are configured by P5CNT.2–0, and are alternate function of P5.7–4.

PORT PIN FUNCTION

P6CNT.5–3	P4.7	P4.6	P4.5	P4.4	MAX. MEMORY SIZE PER PCEx
000	I/O	I/O	I/O	I/O	32kB
001	I/O	I/O	I/O	A16	128kB
010	I/O	I/O	A17	A16	256kB
011	I/O	A18	A17	A16	512kB
100	A19	A18	A17	A16	1MB

CE0–CE7 can be individually configured as program or program/data memory through the MCON and MCON1 SFR. When CE0–CE7 are converted from program to program/data memory, PCE0–PCE3 is disabled if the corresponding data memory area is covered by CEx. The internally decoded range for each program chip enable (CE0–CE7), is established by the number of external address lines (A21–A16) enabled by the P4CNT.5-P4CNT.3 control bits. The following table outlines the assigned memory boundaries of each peripheral chip enable (PCEx) as determined by the P6CNT.5-P6CNT.3 control bits. Note that, when the external address bus is limited to A0–A15, the chip enables are internally decoded on a 32kB x 8-block boundary. This is to allow the use of the more common 32kB memories, as opposed to using a less common 64kB block size memory.

PERIPHERAL CHIP-ENABLE BOUNDARIES

P6CNT.5–3	$\overline{PCE0}$	$\overline{PCE1}$	$\overline{PCE2}$	$\overline{PCE3}$
000	32kB	32–64kB	64–96kB	96–128kB
001	128kB	128–256kB	256–384kB	384–512kB
010	256kB	256–512kB	512–768kB	768–1024kB
011	512kB	512-1MB	1-1.5MB	1.5-2MB
100	1MB	1-2MB	2-3MB	3-4MB

P6CNT.2–P6CNT.0

Port pin P6.3–P6.0 configuration control bits. P6CNT.2-0 determine whether specific P6 pins function as program chip-enable signals or as I/O.

PORT PIN 4 FUNCTION

P6CNT.2–0	P6.3	P6.2	P6.1	P6.0
000	I/O	I/O	I/O	I/O
100	I/O	I/O	I/O	$\overline{CE4}$
101	I/O	I/O	$\overline{CE5}$	$\overline{CE4}$
110	I/O	$\overline{CE6}$	$\overline{CE5}$	$\overline{CE4}$
111	$\overline{CE7}$	$\overline{CE6}$	$\overline{CE5}$	$\overline{CE4}$

CAN 0 Message Center 6 Control Register (C0M6C)

	7	6	5	4	3	2	1	0
SFR B3h	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M6C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

CAN 0 Message Center 7 Control Register (C0M7C)

	7	6	5	4	3	2	1	0
SFR B4h	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M7C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

CAN 0 Message Center 8 Control Register (C0M8C)

	7	6	5	4	3	2	1	0
SFR B4h	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M8C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

CAN 0 Message Center 9 Control Register (C0M9C)

	7	6	5	4	3	2	1	0
SFR B5h	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M9C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

CAN 0 Message Center 10 Control Register (C0M10C)

	7	6	5	4	3	2	1	0
SFR B6h	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M10C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Consult the description of that register for more information.

Interrupt Priority (IP)

	7	6	5	4	3	2	1	0
SFR B8h	—	PS1	PT2	PS0	PT1	PX1	PTO	PXO
	—	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset, IP is set to 80h on all forms of reset.

Bit 7 **Reserved.** Read data is indeterminate.

PS1
Bit 6 **Serial port 1 interrupt.** This bit controls the priority of the serial port 1 interrupt.
0 = Serial port 1 is a low priority.
1 = Serial port 1 is a high-priority interrupt.

PT2
Bit 5 **Timer 2 interrupt.** This bit controls the priority of timer 2 interrupt.
0 = Timer 2 is a low priority.
1 = Timer 2 is a high-priority interrupt.

PS0
Bit 4 **Serial port 0 interrupt.** This bit controls the priority of the serial port 0 interrupt.
0 = Serial port 0 is a low priority.
1 = Serial port 0 is a high-priority interrupt.

PT1
Bit 3 **Timer 1 interrupt.** This bit controls the priority of timer 1 interrupt.
0 = Timer 1 is a low priority.
1 = Timer 1 is a high-priority interrupt.

PX1
Bit 2 **External interrupt 1.** This bit controls the priority of external interrupt 1.
0 = External interrupt 1 is a low priority.
1 = External interrupt 1 is a high-priority interrupt.

PT0
Bit 1 **Timer 0 interrupt.** This bit controls the priority of timer 0 interrupt.
0 = Timer 0 is a low priority.
1 = Timer 0 is a high-priority interrupt.

PX0
Bit 0 **External interrupt 0.** This bit controls the priority of external interrupt 0.
0 = External interrupt 0 is a low priority.
1 = External interrupt 0 is a high-priority interrupt.

Slave Address Mask Enable Register 0 (SADEN0)

	7	6	5	4	3	2	1	0
SFR B9h	SADEN0.7	SADEN0.6	SADEN0.5	SADEN0.4	SADEN0.3	SADEN0.2	SADEN0.1	SADEN0.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SADEN0.7–0
Bits 7–0 **Slave address mask enable register 0.** This register is a mask enable when comparing serial port 0 addresses for automatic address recognition. When a bit is set in this register, the correspond ing bit location in the SADDR0 register is exactly compared with the incoming serial port 0 data to determine if a receive interrupt should be generated. When a bit in this register is cleared, the corresponding bit in the SADDR0 register becomes a "don't care" and is not compared against the incoming data. All incoming data generates a receive interrupt when this register is cleared.

Slave Address Mask Enable Register 1 (SADEN1)

	7	6	5	4	3	2	1	0
SFR BAh	SADEN1.7	SADEN1.6	SADEN1.5	SADEN1.4	SADEN1.3	SADEN1.2	SADEN1.1	SADEN1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SADEN1.7–0
Bits 7–0

Slave address mask enable register 1. This register is a mask enable when comparing serial port 1 addresses for automatic address recognition. When a bit is set in this register, the corresponding bit location in the SADDR1 register is exactly compared with the incoming serial port 1 data to determine if a receive interrupt should be generated. When a bit in this register is cleared, the cor responding bit in the SADDR1 register becomes a "don't care" and is not compared against the incoming data. All incoming data generates a receive interrupt when this register is cleared.

CAN 0 Message Center 11 Control Register (C0M11C)

	7	6	5	4	3	2	1	0
SFR BBh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M11C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Please consult the description of that register for more information.

CAN 0 Message Center 12 Control Register (C0M12C)

	7	6	5	4	3	2	1	0
SFR BCh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M12C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Please consult the description of that register for more information.

CAN 0 Message Center 13 Control Register (C0M13C)

	7	6	5	4	3	2	1	0
SFR BDh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M13C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Please consult the description of that register for more information.

CAN 0 Message Center 14 Control Register (C0M14C)

	7	6	5	4	3	2	1	0
SFR BEh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M14C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Please consult the description of that register for more information.

CAN 0 Message Center 15 Control Register (C0M15C)

	7	6	5	4	3	2	1	0
SFR BFh	MSRDY	ETI	ERI	INTRQ	EXTRQ	MTRQ	ROW/TIH	DTUP
	RW-0	RW-0	RW-0	RW-0	RC-0	R*-0	R*-0	R*-0

*R = Unrestricted read, C = Clear only, * = See description, -n = Value after reset*

C0M15C
Bits 7–0

Operation of the bits in this register are identical to those found in the CAN 0 message 1 control register (C0M1C: ABh). Please consult the description of that register for more information.

Serial Port Control (SCON1)

	7	6	5	4	3	2	1	0
SFR C0h	SM0/FE_1	SM1_1	SM2_1	REN_1	TB8_1	RB8_1	TI_1	RI_1
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SM0-2
Bits 7-5

Serial port 1 mode. These bits control the mode of serial port 1 as follows.

SM0	SM1	SM2	MODE	FUNCTION	LENGTH	PERIOD
0	0	0	0	Synchronous	8 bits	12 t_{CLK}
0	0	1	0	Synchronous	8 bits	4 t_{CLK}
0	1	X	1	Asynchronous	10 bits	Timer 1
1	0	0	2	Asynchronous	11 bits	64 t_{CLK} (SMOD=0) 32 t_{CLK} (SMOD=1)
1	0	1	2	Asynchronous w/ multiprocessor communication	11 bits	64 t_{CLK} (SMOD=0) 32 t_{CLK} (SMOD=1)
1	1	0	3	Asynchronous	11 bits	Timer 1
1	1	1	3	Asynchronous w/ multiprocessor communication	11 bits	Timer 1

SM0/FE_1
Bit 7

Serial port 1 mode bit 0. When SMOD0 is set to 1, it is the framing error flag that is set upon detection of an invalid stop bit and must be cleared by software. Modification of this bit when SMOD0 is set has no effect on the serial mode setting.

SM1_1
Bit 6

Serial port 1 mode bit 1.

SM2_1
Bit 5

Serial port 1 mode bit 2. Setting of this bit in mode 1 ignores reception if an invalid stop bit is detected. Setting this bit in mode 2 or 3 enables multiprocessor communications. This prevents the RI_1 bit from being set and interrupt being asserted, if the 9th bit received is 0.

REN_1
Bit 4

Receive enable.
REN_0 = 0: serial port 1 reception disabled.

REN_0 = 1: serial port 1 receiver enabled for modes 1, 2, and 3.

Initiate synchronous reception for mode 0.

TB8_1
Bit 3

9th transmission bit state. This bit defines the state of the 9th transmission bit in serial port 1, modes 2 and 3.

RB8_1
Bit 2

9th received bit state. This bit identifies the state of the 9th bit of received data in serial port 1, modes 2 and 3. When SM2_1 is 0, it is the state of the stop bit in mode 1. This bit has no meaning in mode 0.

TI_1
Bit 1

Transmitter interrupt flag. This bit indicates that the data in the serial port 1 buffer has been completely shifted out. It is set at the end of the last data bit for all modes of operation and must be cleared by software.

9th received bit state. This bit identifies the state of the 9th bit of received data in serial port 1, modes 2 and 3. When SM2_1 is 0, it is the state of the stop bit in mode 1. This bit has no meaning in mode 0.

RI_1
Bit 0

Receive interrupt flag. This bit indicates that a data byte has been received in the serial port 1 buffer. It is set at the end of the 8th bit for mode 0, after the last sample of the incoming stop bit for mode 1 subject to the value of the SM2_1 bit, or after the last sample of RB8_1 for modes 2 and 3. This bit must be cleared by software.

Serial Data Buffer 1 (SBUF1)

	7	6	5	4	3	2	1	0
SFR C1h	SBUF1.7	SBUF1.6	SBUF1.5	SBUF1.4	SBUF1.3	SBUF1.2	SBUF1.1	SBUF1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SBUF1.7–0
Bits 7–0

Serial data buffer 1. Data for serial port 1 is read from or written to this location. The serial transmit and receive buffers are separate registers, but both are addressed at this location.

Power-Management Register (PMR)

	7	6	5	4	3	2	1	0
SFR C4h	CD1	CD0	SWB	CTM	4X/$\overline{2X}$	ALEOFF	—	—
	R*-1	R*-0	RW-0	R*-0	R*-0	RW-0	R-1	R-0

*R = Unrestricted read, W = Unrestricted write, * = See description, -n = Value after reset*

CD1, CD0
Bits 7–6

Clock divide control bit 0.

Clock divide control bit 1. These bits select the number of crystal oscillator clocks required to generate one machine cycle. Switching between modes requires a transition through the divide-by-4 mode (CD1, CD0 = 01). For example, to go from 1 to 1024 clocks-per-machine cycle, the device must first go from 1 to 4 clocks per cycle, and then from 4 to 1024 clocks per cycle. Attempts to perform an invalid transition are ignored. The setting of these bits affects the timers and serial ports, as shown in the following table:

CD1:0	4X/$\overline{2X}$	OSCILLATOR CYCLES PER MACHINE CYCLE	OSC CYCLES PER TIMER 0/1/2 CLOCK		OSC CYCLES PER TIMER 2 CLK, BAUD RATE GEN	OSC CYCLES PER SERIAL PORT CLK, MODE 0		OSC CYCLES PER SERIAL PORT CLK, MODE 2	
			TXM = 0	TXM = 1		SM2 = 0	SM2 = 1	SMOD = 0	SMOD = 1
00	1	1	12	1	2	3	1	64	32
00	0	2	12	2	2	6	2	64	32
01	x	Reserved							
10	x	4	12	4	2	12	4	64	32
11	x	1024	3072	1024	512	3072	1024	64	32

Attempts to change these bits to the frequency multiplier setting (one or two clocks per cycle) fail when running from the internal ring oscillator. In addition, it is not possible to change these bits to the 1024 clocks-per-machine cycle setting while the switchback enable bit (SWB) is set and any of the switchback sources (external interrupts or serial port transmit or receive activity) are active.

SWB
Bit 5

Switchback enable. When set to 1, SWB allows mask-enabled external interrupts, as well as enabled serial port receive functions, to force the clock divide control (CD1 and CD0) bits from 11b (1024 oscillator cycles per machine cycle) to 10b (four oscillator cycles per machine cycle). When SWB is cleared to 0, switchback mode is disabled. Switchback is supported only from the divide-by-1024 mode. The first switchback condition is initiated by the detection of a low on INT0, INT1, INT3 or INT5, or high on INT2 or INT4, when the respective pin has been program-enabled to issue an interrupt. Note that the switchback interrupt relationship requires that the respective external interrupt source be allowed to generate an interrupt as defined by the priority of the interrupt and the state of nested interrupts, before the switchback actually occurs. The second switchback condition occurs when the serial port is enabled to receive data and is found to have an active-low start bit on the receive input pin. Serial port transmit activity also forces a switchback if the SWB is set. Note that the serial port activity, as related to the switchback, is independent of the serial port interrupt relationship. The automatic switchback is only enabled when the clock divide control bits has established a divide-by-1024 mode and the SWB is set to 1.

CTM
Bit 4

Crystal multiplier enable. The CTM bit is used to enable the crystal clock multiplier. The CTM bit can be changed only when the CD1 and CD0 bits are set to divide-by-4 mode and the RGMD is cleared to 0. When programmed to 0, the CTM bit disables the crystal clock multiplier to save energy and, when programmed to 1, the CTM bit enables the crystal clock multiplier. The crystal clock multiplier requires a startup stabilization period. Setting CTM to 1 from a previous 0 automatically clears the CKRY bit in the EXIF register and starts the crystal clock warmup period. During the startup count, the CKRY bit remains cleared and the CD1: 0 bits should not be changed to select the crystal clock multiplier until the CKRY has indicated the startup time has elapsed (CKRY = 1). CTM cannot be changed from a 1 to a 0 while the crystal clock multiplier option is selected by the CD1 and CD0 clock control bits. Setting the CTM bit enables the crystal clock multiplier to run at the programmed 2X or 4X multiply rate established by the 4X/$\overline{2X}$ bit. The 4X/$\overline{2X}$ bit cannot be altered unless the CTM bit is cleared. The CTM is also automatically cleared to logic 0 when the processor enters into a stop mode.

4X/$\overline{2X}$
Bit 3

System clock multiplier. The 4X/$\overline{2X}$ bit establishes the multiplication factor associated with the internal crystal oscillator multiplier. Clearing this bit to a logic 0 sets the multiply function as a frequency doubler (2X crystal frequency). Setting this bit to a logic 1 adjusts the multiply function to operate as a frequency quadrupler (4X crystal frequency). This bit must be established for the preferred multiplication factor before setting the crystal multiplier (CTM) bit. The 4X/$\overline{2X}$ bit can only be altered when the CTM bit is cleared. This prevents the system from changing the multiplication factor while the clock multiplier is enabled, and forces such a change to be made from the divide-by-4 mode.

ALEOFF
Bit 2

ALE disable. When set to 1, this bit disables ALE (set high externally) during all on-board program and data memory access times. External multiplexed address/data (off-chip) memory access (MUX = 0) automatically enables ALE, independent of ALEOFF. External demultiplexed address/data (off-chip) memory access (MUX = 1) automatically disables ALE if ALEOFF = 1, or leaves ALE toggling if ALEOFF = 0. When ALEOFF is cleared to 0, ALE toggles normally at all times.

Bits 1-0

Reserved.

Status Register (STATUS)

	7	6	5	4	3	2	1	0
SFR C5	PIP	HIP	LIP	—	SPTA1	SPRA1	SPTA0	SPRA0
	R-0	R-0	R-0	—	R-0	R-0	R-0	R-0

R = Unrestricted read, -n = Value after reset

PIP
Bit 7

Power-fail priority interrupt status. When set, this bit indicates that software is currently servicing a power-fail interrupt. It is cleared when the program executes the corresponding RETI instruction.

HP
Bit 6

High-priority interrupt status. When set, this bit indicates that software is currently servicing a high-priority interrupt. It is cleared when the program executes the corresponding RETI instruc tion.

LIP
Bit 5

Low-priority interrupt status. When set, this bit indicates that software is currently servicing a low-priority interrupt. It is cleared when the program executes the corresponding RETI instruction.

Bit 4

Reserved. Read value is indeterminate.

SPTA1
Bit 3

Serial port 1 transmit activity monitor. When set, this bit indicates that data is currently being transmitted by serial port 1. It is cleared when the internal hardware sets the TI_1 bit.

SPRA1
Bit 2

Serial port 1 receive activity monitor. When set, this bit indicates that data is currently being received by serial port 1. It is cleared when the internal hardware sets the RI_1 bit.

SPTA0
Bit 1

Serial port 0 transmit activity monitor. When set, this bit indicates that data is currently being transmitted by serial port 0. It is cleared when the internal hardware sets the TI_0 bit.

SPRA0
Bit 0

Serial port 0 receive activity monitor. When set, this bit indicates that data is currently being received by serial port 0. It is cleared when the internal hardware sets the RI_0 bit.

Memory Control Register (MCON)

	7	6	5	4	3	2	1	0
SFR C6h	IDM1	IDM0	CMA	—	PDCE3	PDCE2	PDCE1	PDCE0
	RT-0	RT-0	RT-0	R-1	RT-0	RT-0	RT-0	RT-0

R = Unrestricted read, T = Timed-access write only, -n = Value after reset

IDM1, IDM0
Bits 7-6

Internal data memory configuration and memory. The IDM1 and IDM0 bits establish the address location of the internal 9kB SRAM memory. Use of the SRAM for extended stack memory (SA = 1 in the ACON SFR) is not disrupted by the memory relocation assignment.

IDM1	IDM0	INTERNAL SRAM MEMORY LOCATION (HEX)
0	0	00DC00–00FFFF (1kB = 00DC00–00DFFF, 8kB = 00E000–00FFFF)
0	1	000000–0023FF (8kB = 000000–001FFF, 1kB = 002000–0023FF)
1	0	FFDC00–FFFFFF (1kB = FFDC00–FFDFFF, 8kB = FFE000–FFFFFF)
1	1	Reserved, trying to write 11b does not change the previous setting.

CMA
Bit 5

CAN data memory assignment. The CMA bit provides a software mechanism for moving the data memory blocks associated with the CAN controller. The 256 bytes of data memory can be located at one of the two following address locations.

CMA	CAN MEMORY LOCATION (HEX)
0	00DB00–00DBFF - Reset default
1	FFDB00–FFDBFF

Bit 4

Reserved.

PDCE3
Bit 3

Program/data chip enable 3. PDCE3 provides the software selection for CE3 to be used with either program or program and data memory when CE3 is enabled by the port 4 control register (P4CNT). PDCE3 becomes a "don't care" when CE3 is not enabled. The port 4 control register SFR establishes the specific address range for CE3. Write access to the memory block, which is connected to CE3 as data memory (PDCE3 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE3 as program and data memory (PDCE3 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE3 = 0 enables CE3 as a program memory chip enable.

PDCE3 = 1 enables CE3 as a merged program and data memory chip enable.

PDCE2
Bit 2

Program/data chip enable 2. PDCE2 provides the software selection for CE2 to be used with either program or program and data memory when CE2 is enabled by the port 4 control register (P4CNT). PDCE2 becomes a "don't care" when CE2 is not enabled. The port 4 control register SFR establishes the specific address range for CE2. Write access to the memory block, which is connected to CE2 as data memory (PDCE2 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE2 as program and data memory (PDCE2 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE2 = 0 enables CE2 as a program memory chip enable.

PDCE2 = 1 enables CE2 as a merged program and data memory chip enable.

PDCE1
Bit 1

Program/data chip enable 1. PDCE1 provides the software selection for CE1 to be used with either program or program and data memory when CE1 is enabled by the port 4 control register

(P4CNT). PDCE1 becomes a "don't care" when CE1 is not enabled. The port 4 control register SFR establishes the specific address range for CE1. Write access to the memory block, which is connected to CE1 as data memory (PDCE1 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE1 as program and data memory (PDCE1 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE1 = 0 enables CE1 as a program memory chip enable.

PDCE1 = 1 enables CE1 as a merged program and data memory chip enable.

PDCE0
Bit 0

Program/data chip enable 0. PDCE0 provides the software selection for CE0 to be used with either program or program and data memory when CE0 is enabled by the port 4 control register (P4CNT). PDCE0 becomes a "don't care" when CE0 is not enabled. The port 4 control register SFR establishes the specific address range for CE0. Write access to the memory block, which is connected to CE0 as data memory (PDCE0 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE0 as program and data memory (PDCE0 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE0 = 0 enables CE0 as a program memory chip enable.

PDCE0 = 1 enables CE0 as a merged program and data memory chip enable.

Timed-Access Register (TA)

	7	6	5	4	3	2	1	0
SFR C7h	TA.7	TA.6	TA.5	TA.4	TA.3	TA.2	TA.1	TA.0
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

W = Unrestricted write, -n = Value after reset

TA.7–0
Bits 7–0

This register provides a timed-control sequence for software writes to some special register bits, in order to protect against inadvertent changes to configuration and to the program memory in the event of a loss of software control.

Timer 2 Control (T2CON)

	7	6	5	4	3	2	1	0
SFR C8h	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TF2
Bit 7

Timer 2 overflow flag. This bit is set when timer 2 overflows from FFFFh, or the count is equal to the capture register in down-count mode. It must be cleared by software. This bit can only be set if RCLK and TCLK are both cleared to 0.

EXF2
Bit 6

Timer 2 external flag. A negative transition on the T2EX (P1.1) causes this flag to be set if (CP/PL2 = EXEN2 = 1) or (CP/PL2 = DCEN = 0 and EXEN2 = 1). When CP/PL2 = 0 and DCEN = 1, this bit toggles whenever timer 2 underflows or overflows. In this mode, EXF2 can be used as the 17th timer bit and does not cause an interrupt. If set by a negative transition, this flag must be cleared by software. Setting this bit forces a timer interrupt, if enabled.

RCLK
Bit 5

Receive clock flag. This bit determines the serial port 0 time base when receiving data in serial modes 1 or 3. Setting this bit to 1 causes timer 2 overflow to be used to determine receive baud rate and forces timer 2 into baud-rate generation mode, which operates from divide-by-2 of the external clock. Clearing this bit to 0 causes timer 1 overflow to be used.

TCLK
Bit 4

Transmit clock flag. This bit determines the serial port 0 time base when transmitting data in serial modes 1 or 3. Setting this bit to 1 causes timer 2 overflow to be used to determine transmit baud rate and forces timer 2 into baud-rate generation mode, which operates from divide-by-2 of the external clock. Clearing this bit to 0 causes timer 1 overflow to be used.

EXEN2

Timer 2 external enable. Setting this bit to 1 enables the capture/reload function on the T2EX

Bit 3	(P1.1) pin for a negative transition, if timer 2 is not generating baud rates for the serial port. Clearing this bit to 0 causes timer 2 to ignore all external events on T2EX pin.
TR2 Bit 2	**Timer 2 run control.** This bit enables timer 2 operation when set to 1. Clearing this bit to 0 halts timer 2 operation and preserves the current count in TH2 and TL2.
C/$\overline{T2}$ Bit 1	**Counter/timer select.** This bit determines whether timer 2 functions as a timer or counter. Setting this bit to 1 causes timer 2 to count negative transitions on the T2 (P1.0) pin. Clearing this bit to 0 causes timer 2 to function as a timer. The speed of timer 2 is determined by the T2M (CKCON.5) bit. Timer 2 operates from divide-by-2 external clock when used in either baud-rate generator or clock output mode.
CP/$\overline{RL2}$ Bit 0	**Capture/reload select.** This bit determines whether the capture or reload function is used for timer 2. If either RCLK or TCLK is set, timer 2 functions in an autoreload mode following each overflow. Setting this bit to 1 causes a timer 2 capture to occur when a falling edge is detected on T2EX if EXEN2 is 1. Clearing this bit to 0 causes an autoreload to occur when timer 2 overflow, or a falling edge, is detected on T2EX if EXEN2 is 1.

Timer 2 Mode (T2MOD)

	7	6	5	4	3	2	1	0
SFR C9h	—	—	—	D13T1	D13T2	—	T2OE	DCEN
	RW-1	RW-1	RW-1	RW-0	RW-0	RW-1	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

Bits 7-5	**Reserved.**
D13T1 Bit 4	**Divide-by-13 clock option for timer 1.** The D13T1 bit provides an alternate clock source to the timer 1 in place of the normal external T1 input pin. When D13T1 is cleared to 0, the clock source for timer 1 is supplied through the standard T1 external input pin, the divide-by-12 of the oscillator (T1M = 0), or the divide-by-4 of the oscillator (T1M = 1), as controlled by T1M and C/T. When D13T1 is set to a 1, the clock source for timer 1 is supplied through a separate divide-by-13 of the system clock, independent of T1M. The C/T bit must also be programmed to a 1 to select the divide-by-13 counter.
D13T2 Bit 3	**Divide-by-13 clock option for timer 2.** The D13T2 bit provides an alternate clock source to the timer 2 in place of the normal external T2 input pin. When D13T2 is cleared to 0, the clock source for timer 2 is supplied through the standard T2 external input pin, the divide-by-12 of the oscillator (T2M = 0), or the divide-by-4 of the oscillator (T2M = 1), as controlled by T2M and C/$\overline{T2}$. When D13T2 is set to a 1, the clock source for timer 2 is supplied through a separate divide-by-13 of the system clock independent of T2M. The C/$\overline{T2}$ bit must also be programmed to a 1 to select the divide-by-13 counter.
Bit 2	**Reserved.**
T2OE Bit 1	**Timer 2 output enable.** Setting this bit to 1 enables the clock output function of T2 (P1.0) pin if C/T2 = 0. Timer 2 rollovers do not cause interrupts. Clearing this bit to 0 causes the T2 pin to function either as a standard port pin or a counter input for timer 2.
DCEN Bit 0	**Down-count enable.** This bit, in conjunction with the T2EX pin, controls the direction that timer 2 counts in 16-bit autoreload mode. Clearing this bit to 0 causes timer 2 to count up. Setting this bit to 1 causes timer 2 to count up if the T2EX pin is 1, and timer 2 to count down if the T2EX pin is 0.

Timer 2 Capture LSB (RCAP2L)

	7	6	5	4	3	2	1	0
SFR CAh	RCAP2L.7	RCAP2L.6	RCAP2L.5	RCAP2L.4	RCAP2L.3	RCAP2L.2	RCAP2L.1	RCAP2L.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

RCAP2L.7–0
Bits 7–0

Timer 2 capture LSB. This register is used to capture the TL2 value when timer 2 is configured in capture mode. RCAP2L is also used as the LSB of a 16-bit reload value when timer 2 is configured in autoreload mode.

Timer 2 Capture MSB (RCAP2H)

	7	6	5	4	3	2	1	0
SFR CBh	RCAP2H.7	RCAP2H.6	RCAP2H.5	RCAP2H.4	RCAP2H.3	RCAP2H.2	RCAP2H.1	RCAP2H.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

RCAP2H.7–0
Bits 7–0

Timer 2 capture MSB. This register is used to capture the TH2 value when timer 2 is configured in capture mode. RCAP2H is also used as the MSB of a 16-bit reload value when timer 2 is configured in autoreload mode.

Timer 2 LSB (TL2)

	7	6	5	4	3	2	1	0
SFR CCh	TL2.7	TL2.6	TL2.5	TL2.4	TL2.3	TL2.2	TL2.1	TL2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TL2.7–0
Bits 7–0

Timer 2 LSB. This register contains the least significant byte of timer 2.

Timer 2 MSB (TH2)

	7	6	5	4	3	2	1	0
SFR CDh	TH2.7	TH2.6	TH2.5	TH2.4	TH2.3	TH2.2	TH2.1	TH2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TH2.7–0
Bits 7–0

Timer 2 MSB. This register contains the most significant byte of timer 2.

Clock Output Register (COR)

	7	6	5	4	3	2	1	0
SFR CEh	IRDACK	—	—	C0BPR7	C0BPR6	COD1	COD0	XCLKOE
	RT-0	RT-1	RT-1	RT-0	RT-0	RT-0	RT-0	RT-0

R = Unrestricted read, T = Timed-access write only, -n = Value after reset

IRDACK
Bit 7

IRDA clock output enable. When XCLKOE = 0, IRDACK bit assumes a "don't care" condition. When XCLKOE = 1 and IRDACK = 1, the clock output pad issues a clock that is 16 times the baud rate of the programmed baud rate associated with serial port 0. When XCLKOE = 1 and IRDACK = 0, the clock output pad is controlled by the clock output divide select bits, COD1 and COD0. Note that the appropriate baud rate must be established by use of timer 1, programmed for the baud-rate generator mode 2.

Bits 6-5 **Reserved.**

C0BPR7, C0BPR6
Bits 4-3

CAN 0 baud-rate prescaler bits. The C0BPR7 and C0BPR6 bits establish the two high-order bits associated with the 8-bit baud-rate prescaler in the CAN 0 controller. Note that the C0BPR7 and C0BPR6 bits cannot be written when the SWINT bit in the CAN 0 control register is cleared to 0.

COD1, COD0
Bits 2-1

Clock output divide select bits. The clock output divide bits are used to establish the output clock frequency from the CLKO function on port pin P3.5, when enabled by the COR.0 (XCLKOE) bit. Consult the description of the XCLKOE bit for more information.

COD1	COD0	P3.5 OUTPUT FREQUENCY
0	0	System clock divided by 2
0	1	System clock divided by 4
1	0	System clock divided by 6
1	1	System clock divided by 8

XCLKOE
Bit 0

External clock output enable. XCLKOE = 1 enables a clock defined by COD1-COD0 and IRDACK to be driven from the port pin P3.5. XCLKOE = 1 provides a full push-pull driver on P3.5. COD1 and COD0 are in "don't care" states when XCLKOE and IRDACK are set to logic 1, causing the serial port baud rate to be multiplied by 16. XCLKOE = 0 disables the clock output and leaves the P3.5 pin to function as a general-purpose I/O port (GPIO), or as the T1 alternate function.

Program Status Word (PSW)

	7	6	5	4	3	2	1	0
SFR D0h	CY	AC	F0	RS1	RS0	0V	F1	P
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

CY
Bit 7

Carry flag. This bit is set when if the last arithmetic operation resulted in a carry (during addition) or a borrow (during subtraction). Otherwise, it is cleared to 0 by all arithmetic operations.

AC
Bit 6

Auxiliary carry flag. This bit is set to 1 if the last arithmetic operation resulted in a carry into (during addition), or a borrow from (during subtraction) the high-order nibble. Otherwise, it is cleared to 0 by all arithmetic operations.

F0

Bit 5

User flag 0. This is a bit-addressable, general-purpose flag for software control.

RS1, RS0
Bits 4-3

Register bank select 1-0. These bits select which register bank is addressed during register accesses.

RS1	RS0	REGISTER BANK	ADDRESS
0	0	0	00h–07h
0	1	1	08h–0Fh
1	0	2	10h–17h
1	1	3	18h–1Fh

OV
Bit 2

Overflow flag. This bit is set to 1 if there is a carry-out of bit 6, but not out of bit 7, or if there is a carry-out of bit 7, but not out of bit 6 for addition. When adding signed integers, OV indicates a negative number resulted as the sum of two positive operands, or a positive sum from two negative operands. OV is set if a borrow is needed into bit 6, but not into bit 7, or is needed into bit 7, but not into bit 6 for subtraction. When subtracting signed integers, OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive value is subtracted from a negative value. OV is also set if the product is greater than 0FFh for multiplication. This bit is always cleared for division operations.

F1
Bit 1

User flag 1. This is a bit-addressable, general-purpose flag for software control.

P
Bit 0

Parity flag. This bit is set to 1 if there is an odd number of 1's in the accumulator, and is cleared to 0 if there is an even number of 1's in the accumulator.

Multiplier Control Register 0 (MCNT0)

	7	6	5	4	3	2	1	0
SFR D1h	LRSFT	CSE	SCE	MAS4	MAS3	MAS2	MAS1	MAS0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

LRSFT
Bit 7

Left/right shift. The LRSFT bit is cleared to 0 following either a system reset or the initialization of the accelerator. The LRSFT bit is programmed to 0 when a left shift is required, and is pro grammed to 1 for a right shift. LRSFT does not alter any other type of calculation other than the shift function.

CSE
Bit 6

Circular shift enable. The CSE bit is cleared to 0 following a system reset. When CSE and SCE are cleared to 0's, all left or right shifts performed on the MA register, as per the programming of LRST and MAS4–MAS0, shift clear bit values into the most significant bit for a right shift and the least significant bit for a left shift. The least significant bit is also lost when doing a right shift, and the most significant bit is lost when doing a left shift. When CSE is set to 1 and SCE is cleared to 0, the most significant bit of the MA register is shifted into the least most significant bit for a left shift. Similarly, the least significant bit of the MA register is shifted into the most significant bit for a right shift. When CSE is cleared to 0 and SCE is set to 1, the shift carry bit is shifted into the most significant bit for the right shift and the least significant bit for a left shift. The least significant bit is also lost when doing a right shift, and the most significant bit is lost when doing a left shift. When CSE and SCE are set to 1, the most significant MA bit is shifted into the shift carry bit. The shift carry bit is shifted into the least significant MA bit when doing a left shift. The least significant MA bit is shifted into the shift carry bit, while the shift carry bit is shifted into the most significant MA bit when doing a right shift.

SCE
Bit 5

Shift carry enable. The SCE bit is cleared to 0 following a system reset. When SCE is cleared to a 0, all left or right shifts performed on the MA register, as per the programming of CSE, LRST, and MAS4–MAS0, do not incorporate the shift carry bit SCB (MCNT1.5) as a part of the shifting process. When SCE is set to a 1, the shift carry bit is shifted into the least significant bit for a left shift and into the most significant bit for a right shift. If CSE is cleared to a 0, the shift carry bit remains unchanged during the shift process. If CSE is set to a 1, the most significant MA bit is shifted into the shift carry bit when doing a left shift, and least most significant MA bit is shifted into the shift carry bit when doing a right shift.

MAS4–0
Bits 4–0

Multiplier register shift bits. These bits determine the number of shifts performed when a shift operation is performed with the arithmetic accelerator, and are also used to indicate how many shifts were performed during a previous normalization operation. These bits are cleared to 00000b following a system reset or the initialization of the arithmetic accelerator.

When these bits are cleared to 00000b after loading the arithmetic accelerator, the device normalizes the 32-bit value loaded into the arithmetic accelerator accumulator, rather than shifting it. Following the normalization operation, the MAS4–0 bits are modified to indicate how many shifts were performed.

MAS4	MAS3	MAS2	MAS1	MAS0	NUMBER OF SHIFTS OF ARITHMETIC ACCELERATOR ACCUMULATOR
0	0	0	0	0	Normalization
0	0	0	0	1	Shift by 1
0	0	0	1	0	Shift by 2
0	0	0	1	1	Shift by 3
—	—	—	—	—	—
1	1	1	1	0	Shift by 30
1	1	1	1	1	Shift by 31

Multiplier Control Register 1 (MCNT1)

	7	6	5	4	3	2	1	0
SFR D2h	MST	MOF	SCB	CLM	—	—	—	—
	RW-0	R-0	RW-0	RW-0	R-1	R-1	R-1	R-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

MST
Bit 7

Multiply/accumulate status flag. The MST bit indicates the current status of the multiplier. When MST is set to a 1 by the multiplier/accumulator hardware, it indicates that the multiplier/accumulator has not completed an assigned task. Immediately after the processor begins loading data into the MA or MB register, MST is automatically set and remains set until the assigned task is completed. There are no restrictions on how quickly data is entered into the MA or MB registers. The only requirement to do a calculation is to perform the load of MA, MB, and/or MCNT1 within the specified sequential relationship associated with the requested task. MST is automatically cleared by the multiplier/accumulator hardware once an assigned task is completed and the results are ready for the processor to read. A cleared value in MST (0) also indicates that the accelerator is in an initialized state and can be loaded with new values. Any data previously stored in MA or MB as the result or remainder of a previous calculation is lost once new data is loaded into MA or MB. Data in the message center register is continually updated by the accumulation function and is preserved from one calculation to another. The processor software can also clear the MST bit from a previous high state when the processor needs to initialize the multiplier prior to the completion of a current operation. This action initializes the state machine action within the accelerator, which allows the processor to immediately begin loading new data into MA and/or MB to perform a new calculation. An additional initialization can be achieved if the MA register or MB register is loaded prior to the completion of a current calculation. All previous calculation results are lost as the accelerator resets the registers to begin accepting new data. In either of these forced clearing methods, stored data in the message center accumulator register can become invalid.

-

MOF
Bit 6

Multiply overflow flag. The MOF flag bit is cleared to 0, following either a system reset or the initialization of the accelerator. The MOF bit is automatically set when the accelerator detects a divide-by-0, or when the result of the calculation is larger than FFFF hex.

SCB
Bit 5

Shift carry bit. The SCB is used as a carry bit for shift operation when SCE (MCNT0.5) bit is set to 1. Note that the SCB is not cleared at the beginning of a new operation and must be cleared by a write to this bit or a system reset.

CLM
Bit 4

Clearing the MA, MB, and MC (accumulator) register. Setting the CLM bit clears the MA, MB, and MC registers, and CLM is automatically cleared to a zero state following the clear operation. Writing a 0 to this bit results no operation.

Bits 3-0

Reserved.

Multiplier A Register (MA)

	7	6	5	4	3	2	1	0
SFR D3h	MA.7	MA.6	MA.5	MA.4	MA.3	MA.2	MA.1	MA.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

Bits 7–0

Multiplier A register. The multiplier A (MA) register is used to load the 32-bit or 16-bit numerator when the math accelerator is configured in a 32-bit by 16-bit or 16-bit by 16-bit divide mode. The multiplier A register is also used to load the second value associated with a 16-bit by 16-bit calculation when the accelerator is used in the multiply mode. A read of the MA register, following a completed function, provides the 32-bit result of a 32-bit by 16-bit divide, the 16-bit result of a 16-bit by 16-bit divide, the result of a 16-bit by 16-bit multiplication calculation, the result of a normalized 32-bit calculation, or the result of a shifted 32-bit calculation.

A read pointer and a write pointer keep track of which of the four bytes is read or written to when

accessing or loading the 32 or 16 bits of the MA register. The pointer is set to the most significant byte for reads and the least significant byte for writes following a system reset, the completion of a calculation, the setting of the CLM bit, or the setting of the MST bit in the MCNT1 SFR. Following each read of MA, the read pointer is moved to the next most significant byte until the entire contents of MA are read. Similarly, each write moves the write pointer to the next least significant byte until the entire 32 bits of the MA register is written. Neither of the pointers wrap around, but rather lock at the extreme end of associated read or write 32-bit word size. Note that, in loading or reading a 16-bit value, only two reads or writes are required. In loading a 16-bit value, ensure that the remaining 16 bits of the 32-bit value are completely cleared.

When accessing data from the MA register, the most significant byte is the first byte read from MA when downloading the contents of a completed multiply or divide, as determined by the MST bit in the MCNT1 SFR. All subsequent reads of MA, after completing the appropriate reads to secure the respective results of the above calculations, produce a 00 hex value. MA is also cleared to 00 hex following either a system reset, the setting of CLM, or the setting of the MST bit in the MCNT1 SFR. When loading the MA register, data must be written with the least significant byte first and most significant byte last.

Multiplier B Register (MB)

	7	6	5	4	3	2	1	0
SFR D4h	MB.7	MB.6	MB.5	MB.4	MB.3	MB.2	MB.1	MB.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

Bits 7–0

Multiplier B register. The multiplier B register is used to load the 16-bit denominator when the math accelerator is configured in a 32-bit by 16-bit or 16-bit by 16-bit divide mode. The multiplier B register is also used to load the first value associated with a 16-bit by 16-bit calculation when the accelerator is used in the multiply mode. A read of the MB register following a completed function provides the 16-bit remainder of a 32-bit by 16-bit divide or the 16-bit remainder of a 16-bit by 16-bit divide.

A read pointer and a write pointer keep track of which of the two bytes is read or written to when accessing or loading the 16 bits of the MB register. The pointer is set to the most significant byte for reads and the least significant byte for writes following a system reset, the completion of a calculation, the setting of the CLM bit, or the setting of the MST bit in the MCNT1 SFR. Following each read of MB, the read pointer is moved to the least significant byte. Similarly, a write moves the write pointer to the most significant byte of the MB register. Neither of the pointers wrap around, but rather lock at the extreme end of associated read or write 16-bit word size.

When accessing data from the MB register, the most significant byte is the first byte read from MB when downloading the contents of a completed multiply or divide, as determined by the MST bit in the MCNT1 SFR. The next read of MB produces the least significant byte, and any subsequent read produces a 00 hex value. MB also reads as 00 hex, following either a system reset or the initialization of the accelerator. When loading the MB register, data must be written with the least significant byte first and most significant byte last.

Multiplier C Register (MC)

	7	6	5	4	3	2	1	0
SFR D5h	MC.7	MC.6	MC.5	MC.4	MC.3	MC.2	MC.1	MC.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

Bits 7–0

Multiplier C register. The multiplier C register is also termed the accumulator register within the math accelerator. Each time a multiply or divide is performed with the MA and MB registers, the resulting value is added to the previous value of the accumulator, and is then stored back into the accumulator. All shift or normalization tasks are not added to the accumulator. The MC register is a full read/write register that provides direct access to the accumulator. The accumulator is 40-bits long, and is accessed by five reads or writes of the MC register. The accumulator is cleared following a system reset, setting the CLMC bit in the MCNT1 SFR, or by loading five consecutive 00 hex values into the MC register.

A read pointer and a write pointer keep track of which of the five bytes is read or written to when accessing or loading the 40 bits of the accumulator. The pointer is set to the most significant byte for reads and the least significant byte for writes following a system reset, the completion of a calculation, the setting of the CLM bit, or the setting of the MST bit in the MCNT1 SFR. Following each read of MC, the read pointer is moved to the next less significant byte until the entire contents of MC are read. Similarly, each write moves the write pointer to the next more sig nificant byte until the entire 40 bits of the MC register is written. Neither of the pointers wrap around, but rather lock at the extreme end of associated read or write 40-bit word size. Note that, in loading or reading a 16-bit (or 32-bit) value, only two (or four) reads or writes are required. In loading a 16-bit (or 32-bit) value, it is important to make sure that the remaining 16 bits (or 8 bits) of the 40-bit value are all cleared.

The most significant byte is the first byte read from MC when downloading the contents of a completed addition of the accumulator, as determined by the MST bit in the MCNT1 SFR. Unlike the MA and MB registers, data in the accumulator is not cleared during a read. All subsequent reads of MB, after completing the appropriate reads to secure the respective results of the above calculations, produce the contents of the fifth byte of data associated with the 40-bit accumulator value. When loading the MC register, data must be written with the least significant byte first and most significant byte last.

Memory Control Register 1 (MCON1)

	7	6	5	4	3	2	1	0
SFR D6h	—	—	—	—	PDCE7	PDCE6	PDCE5	PDCE4
	RT-1	RT-1	RT-1	RT-1	RT-0	RT-0	RT-0	RT-0

R = Unrestricted read, T = Timed-access write, -n = Value after reset

Bits 7–4

Reserved.

PDCE7
Bit 3

Program/data chip enable 7. PDCE7 provides the software selection for CE7 to be used with either program or program and data memory when CE7 is enabled by the port 6 control register (P6CNT). PDCE7 becomes a "don't care" when CE7 is not enabled. The port 4 control register SFR establishes the specific address range for CE7. Write access to the memory block, which is connected to CE7 as data memory (PDCE7 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE7 as program and data memory (PDCE7 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE7 = 0 enables CE7 as a program memory chip enable.

PDCE7 = 1 enables CE7 as a merged program and data memory chip enable.

PDCE6
Bit 2

Program/data chip enable 6. PDCE6 provides the software selection for CE6 to be used with either program or program and data memory when CE6 is enabled by the port 6 control register

(P6CNT). PDCE2 becomes a "don't care" when CE6 is not enabled. The port 4 control register SFR establishes the specific address range for CE6. Write access to the memory block, which is connected to CE6 as data memory (PDCE6 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE6 as program and data memory (PDCE6 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE6 = 0 enables CE6 as a program memory chip enable.

PDCE6 = 1 enables CE6 as a merged program and data memory chip enable.

PDCE5
Bit 1

Program/data chip enable 5. PDCE5 provides the software selection for CE5 to be used with either program or program and data memory when CE5 is enabled by the port 6 control register (P6CNT). PDCE1 becomes a "don't care" when CE5 is not enabled. The port 4 control register SFR establishes the specific address range for CE5. Write access to the memory block, which is connected to CE5 as data memory (PDCE5 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE5 as program and data memory (PDCE5 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE5 = 0 enables CE5 as a program memory chip enable.

PDCE5 = 1 enables CE5 as a merged program and data memory chip enable.

PDCE4
Bit 0

Program/data chip enable 4. PDCE4 provides the software selection for CE4 to be used with either program or program and data memory when CE04 is enabled by the port 6 control register (P6CNT). PDCE4 becomes a "don't care" when CE4 is not enabled. The port 4 control register SFR establishes the specific address range for CE4. Write access to the memory block, which is connected to CE4 as data memory (PDCE4 = 1), comes from the P3.6 WR signal. A read of the memory block connected to CE4 as program and data memory (PDCE4 = 1) comes from the PSEN signal, as opposed to the normal P3.7 RD signal when doing data memory reads.

PDCE4 = 0 enables CE4 as a program memory chip enable.

PDCE4 = 1 enables CE4 as a merged program and data memory chip enable.

Memory Control Register 2 (MCON2)

	7	6	5	4	3	2	1	0
SFR D7h	WPIF	WPR2	WPR1	WPR0	WPE3	WPE2	WPE1	WPE0
	RT-0	RT-0	RT-0	RT-0	RT-0	RT-0	RT-0	RT-0

R = Unrestricted read, T = Timed-access write, -n = Value after reset

WPIF
Bit 7

Write-protected interrupt flag. This flag is set by hardware when an MOVX instruction attempts to write to a write-protected memory area. Once set, this bit must be cleared by software.

WPR2-0
Bits 6-4

Write-protected range bits 2-0. These bits specify the write-protection range when any write-protected enable bits are set:

WPR2	WPR1	WPR0	PROTECTION RANGE(kB)
0	0	0	0–2
0	0	1	0–4
0	1	0	0–6
0	1	1	0–8
1	0	0	0–10
1	0	1	0–12
1	1	0	0–14
1	1	1	0–16

WPE3
Bit 3

Write-protected enable 3. Setting the WPE3 to 1 enables write protection on the lower memory locations controlled by CE3 when PDCE3 (MCON.3) is set. Any MOVX write attempt to these

locations sets the WPIF bit and leaves the data unaltered. Clearing this bit to 0 disables the write protection. The protection range is specified by the WPR2-0 bits.

WPE2
Bit 2

Write-protected enable 2. Setting the WPE2 to 1 enables write protection on the lower memory locations controlled by CE2 if PDCE2 in the MCON register is also set. Any MOVX write to these locations sets the WPIF bit and no data is altered. Clearing this bit to 0 disables the write protection. The protection range is specified by the WPR2-0 bits.

WPE1
Bit 1

Write-protected enable 1. Setting the WPE1 to 1 enables write protection on the lower memory locations controlled by CE1 if PDCE1 in the MCON register is also set. Any MOVX write to these locations sets the WPIF bit and no data is altered. Clearing this bit to 0 disables the write protection. The protection range is specified by the WPR2-0 bits.

WPE0
Bit 0

Write-protected enable 0. Setting the WPE0 to 1 enables write protection on the lower memory locations controlled by CE0 if PDCE0 in the MCON register is also set. Any MOVX write to these locations sets the WPIF bit and no data is altered. Clearing this bit to 0 disables the write protection. The protection range is specified by the WPR2-0 bits.

Watchdog Control (WDCON)

	7	6	5	4	3	2	1	0
SFR D8h	SMOD_1	POR	EPFI	PFI	WDIF	WTRF	EWT	RWT
	RW-0	RT-*	RW-0	RW-*	RT-0	RW-*	RT-*	RT-0

*R = Unrestricted read, W = Unrestricted write, T = Timed-access write only, -n = Value after reset, * = See description*

SMOD_1
Bit 7

Serial modification. Setting this bit to 1 causes the baud rate for serial port 1 to be doubled in modes 1, 2, and 3. Clearing this bit disables the doubler.

POR
Bit 6

Power-on reset flag. This bit indicates whether the last reset was a power-on reset. This bit is typically interrogated following a reset. It must be cleared before the next reset of any kind for software to work correctly. This bit is set following a power-on reset and is unaffected by all other resets.

EPFI
Bit 5

Enable power-fail interrupt. Setting this bit to 1 enables the internal bandgap reference to generate a power-fail interrupt when V_{CC} falls below minimum V_{CC} in normal operation. In stop mode, BGS (EXIF.0) bit has to be set to enable a power-fail interrupt. Clearing this bit to 0 disables the power-fail interrupt.

PFI
Bit 4

Power-fail interrupt flag. This bit is set to a logic 1 when Vcc3 power-fail (V3PF) or Vcc1 power-fail (V1PF) flags are set. Setting of PFI generates a power-fail interrupt request if enabled (EPFI = 1). The V3PF (STATUS1.2) is set when $V_{CC}3$ falls below Vpfw3, and the V1PF (STATUS1.3) is set when Vcc1 falls below Vpfw1. The PFI bit must be cleared in software before exiting the interrupt service routine, or another interrupt is generated. Clearing the PFI bit also clears the V3PF and V1PF flags. Setting this bit by software generates a power-fail interrupt, if enabled. This bit is cleared by software or a power-fail reset if Vcc3 is greater than Vpfw3 and Vcc1 is greater than Vpfw1 following the crystal startup time.

WDIF
Bit 3

Watchdog interrupt flag. This bit is set to 1 by a watchdog timeout, which indicates a watchdog timer event has occurred. When set, EWT (WDCON.1) and EWDI (EIE.4) determine the action to be taken. This bit can only be modified using a timed-access procedure. Setting this bit in software generates a watchdog interrupt, if enabled. This bit must be cleared in software before exiting the interrupt service routine, or another interrupt, is generated.

EWT	EWDI	ACTIONS
0	0	No interrupt has occurred.
0	1	Watchdog interrupt has occurred.
1	0	No interrupt has been generated. Watchdog reset occurs in 512 cycles if RWT is not set.
1	1	Watchdog interrupt has occurred. Watchdog reset occurs in 512 cycles if RWT is not set.

WTRF
Bit 2

Watchdog timer reset flag. When set, this bit indicates that a watchdog timer reset has occurred. It is typically interrogated to determine if a reset was caused by the watchdog timer. It is cleared by power-on reset, but otherwise, it must be cleared by software before the next reset of any kind to allow software to work correctly. Setting this bit by software does not generate a watchdog timer reset. If the EWT bit is cleared, the watchdog timer has no effect on this bit.

EWT
Bit 1

Enable watchdog timer reset. Setting this bit to 1 enables the watchdog timer to reset the device; clearing this bit to 0 disables the watchdog timer reset. It has no effect on the timer itself and its ability to generate a watchdog interrupt. This bit can only be modified using timed-access procedure. The EWT bit is cleared to a logic 0 on power-on reset, and is unchanged by all other resets.

RWT
Bit 0

Reset watchdog timer. Setting this bit resets the watchdog timer count. This bit must be set using a timed-access procedure before the watchdog timer expires, or a watchdog timer reset and/or interrupt is generated if enabled. The timeout period is defined by CKCON.7-6 watchdog timer mode select bits. When read, this bit is always 0.

Slave Address Register 2 (SADDR2)

	7	6	5	4	3	2	1	0
SFR D9h	SADDR2.7	SADDR2.6	SADDR2.5	SADDR2.4	SADDR2.3	SADDR2.2	SADDR2.1	SADDR2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SADDR2.7–0
Bits 7–0

Slave address register 2. This register is programmed with the given or broadcast address assigned to serial port 2.

Breakpoint Address Register 1 (BPA1)

	7	6	5	4	3	2	1	0
SFR DAh	BPA1.7	BPA1.6	BPA1.5	BPA1.4	BPA1.3	BPA1.2	BPA1.1	BPA1.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset, unrestricted read/write in the emulation mode.

BPA1.7–0
Bits 7–0

Breakpoint LSB address register. This register is intended to be used only by the internal breakpoint hardware to store the least significant address byte of the return address when an A5h software breakpoint is issued. Modification of this register is allowed during the breakpoint routine.

Breakpoint Address Register 2 (BPA2)

	7	6	5	4	3	2	1	0
SFR DBh	BPA2.7	BPA2.6	BPA2.5	BPA2.4	BPA2.3	BPA2.2	BPA2.1	BPA2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset, unrestricted read/write in the emulation mode.

BPA2.7–0
Bits 7–0

Breakpoint MSB address register. This register is intended to be used only by the internal breakpoint hardware to store the most significant address byte of the return address when an A5h software breakpoint is issued. Modification of this register is allowed during the breakpoint routine.

Breakpoint Address Register 3 (BPA3)

	7	6	5	4	3	2	1	0
SFR DCh	BPA3.7	BPA3.6	BPA3.5	BPA3.4	BPA3.3	BPA3.2	BPA3.1	BPA3.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset, unrestricted read/write in the emulation mode.

BPA3.7–0
Bits 7–0

Breakpoint XSB address register. This register is intended to be used only by the internal breakpoint hardware to store the extended address byte of the return address when an A5h software breakpoint is issued. Modification of this register is allowed during the breakpoint routine.

Accumulator (ACC)

	7	6	5	4	3	2	1	0
SFR E0h	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

ACC.7–0
Bits 7–0

Accumulator. This register serves as the accumulator for arithmetic operations. It is functionally identical to the accumulator found in the 80C32.

One's Complement Adder Data (OCAD)

	7	6	5	4	3	2	1	0
SFR E1h	OCAD.7	OCAD.6	OCAD.5	OCAD.4	OCAD.3	OCAD.2	OCAD.1	OCAD.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

OCAD.7–0
Bits 7–0

One's complement adder data. This register serves as the data register for the one's complement adder. Two writes to the OCAD initiate a summation by the one's complement adder. When loading the OCAD, data must be written with the least significant byte first and then the most significant byte. When accessing data from the accumulator, the most significant byte is the first byte read from the OCAD. Four reads are required to fully download the contents of the accumulator.

CSR Data (CSRD)

	7	6	5	4	3	2	1	0
SFR E3h	CSRD.7	CSRD.6	CSRD.5	CSRD.4	CSRD.3	CSRD.2	CSRD.1	CSRD.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

CSRD.7–0
Bits 7–0

CSR data. This register serves as the CSR data register for accessing CSR registers inside the MAC core. For a CSR write operation, data to be written to a CSR register is loaded to the 32-bit CSR platform register through the CSRD. Data is accessed by the CSRD after a CSR read operation.

CSR Address (CSRA)

	7	6	5	4	3	2	1	0
SFR E4h	CSRA.7	CSRA.6	CSRA.5	CSRA.4	CSRA.3	CSRA.2	CSRA.1	CSRA.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

CSRA.7–0
Bits 7–0

CSR address. This register serves as the CSR address register for accessing CSR registers inside the MAC core. The lower 8-bit address of the desired CSR register is input through the CSRA to the BCU.

Ethernet Buffer Size (EBS)

	7	6	5	4	3	2	1	0
SFR E5h	FPE	RBF	—	BS4	BS3	BS2	BS1	BS0
	RT-0	R-1	RT-0	RT-0	RT-0	RT-0	RT-0	RT-0

R = Unrestricted read, T = Timed-access write only, -n = Value after reset

FPE
Bit 7

Flush filter failed-packet enable. Setting this bit to 1 enables the BCU to abort the current receive operation and flush data received for the current frame from the receive buffer if the packet fails the address filtering. The receive interrupt flag is not set and no receive interrupt is generated. Clearing this bit to 0 allows the BCU to receive all packets, regardless of its address-filtering result. This bit is overridden if the receive-all bit in the MAC control register is set.

RBF
Bit 6

Receive buffer full. This bit is a read-only bit and is set by hardware when there is no open page in the receive buffer. When RBF is set, the BCU ignores any new frame received by the MAC. Under this condition, the BCU has to acknowledge the receive status word, but the receive data buffer and receive FIFO are not updated, the receive interrupt flag is not set, and no receive interrupt is generated. The RBF bit is set when the BCU aborts an incoming frame that overflows the receive buffer. The RBF bit is cleared by hardware when there are enough open pages (>4) in the receive buffer.

Bit 5

Reserved.

BS4–0
Bit 4–0

Buffer size bits. The BS4:0 bits can be programmed to any value n between 0 and 31, inclusive. The receive buffer occupies the first n pages of the 8kB memory, while the transmit buffer occupies the remaining (32 - n) pages. Changing the BS4:0 bits automatically flushes the con tents of the receive buffer and receive FIFO. Note that, when BS4:0 = 00000b (default value), there are no receive buffers. When BS4:0 = 00001b, this is the first receive buffer and the rest are transmit buffers.

Buffer Control Unit Data (BCUD)

	7	6	5	4	3	2	1	0
SFR E6h	BCUD.7	BCUD.6	BCUD.5	BCUD.4	BCUD.3	BCUD.2	BCUD.1	BCUD.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

BCUD.7–0
Bits 7–0

BCU data. This register serves as the BCU data register for packet transmit and receive operation. For transmit operation, the 11-bit byte count and the starting page address of the transmit packet are loaded to the BCU through the BCUD register. For receive operation, the page information of the current packet can be read by the BCUD register.

Buffer Control Unit Control (BCUC)

	7	6	5	4	3	2	1	0
SFR E7h	BUSY	EPMF	TIF	RIF	BC3	BC2	BC1	BC0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Read returns page information of the first packet in the receive FIFO, W = Unrestricted write, -n = Value after reset

BUSY
Bit 7

Busy. This read-only busy indicator is set by the hardware when the BCU is in the process of executing a CSR read/write operation. It is cleared by the BCU when it is done. Data writes to this bit are ignored.

EPMF
Bit 6

Ethernet power mode interrupt flag. This flag is set when the power-management block detects a wake-up frame or a magic packet. Setting this flag causes an Ethernet power mode interrupt to be generated. This flag must be cleared by software once set. If this flag is set by software, an Ethernet power mode interrupt is generated if enabled.

TIF
Bit 5

Transmit interrupt flag. This flag is set when the BCU has stored a transmit status word in the transmit buffer after a packet transmission. Setting this flag causes an Ethernet activity interrupt to be generated if enabled. This flag must be cleared by software once set. If this flag is set by software, an Ethernet activity interrupt is generated if enabled.

RIF
Bit 4

Receive interrupt flag. This flag is set by hardware when the BCU has stored a receive status word to the receive buffer, and updated the receive FIFO after it has received a packet from the MAC. This flag is also set by an invalidate current frame command whenever the receive FIFO is not empty and the flag is not currently set. Setting RIF causes an Ethernet activity interrupt to be generated if enabled. This flag is cleared by hardware whenever: 1) the receive FIFO becomes empty, 2) a BCU command empties the receive buffer (invalidating the only packet or flushing buffer), 3) the EBS register is updated to change the size of the buffers, or 4) a reset condition occurs. Otherwise, this flag must be cleared by software once set. If this flag is set by software, an Ethernet activity interrupt is generated if enabled. Note that there is potential of missing an interrupt when RIF is cleared immediately following an invalidate current frame command if there is another frame in the receive buffer. It is recommended to clear the flag before invalidation.

BC3-0
Bits 3-0

BCUC control bits.
These bits are used as the BCU command bits to provide communication between the BCU and CPU. The following BCU commands are supported. All other BC3:0 command values are reserved and are ignored by the BCU.

BCUC3:BCUC0	COMMANDS
0000	No operation (default)
0010	Invalidate current receive packet
0011	Flush receive buffer
0100	Transmit request—Normal
0101	Transmit request—Disable padding
0101	Transmit request—Add CRC disabled
1000	Write CSR
1001	Read CSR
1100	Enable sleep mode
1101	Disable sleep mode

Extended Interrupt Enable (EIE)

	7	6	5	4	3	2	1	0
SFR E8h	EPMIE	C0IE	EAIE	EWDI	EWPI	ES2	ET3	EX2-5
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

EPMIE
Bit 7

Ethernet power mode interrupt enable.
EPMIE = 1 enables the Ethernet power mode interrupt.

EPMIE = 0 disables the Ethernet power mode interrupt.

C0IE
Bit 6

CAN 0 interrupt enable. C0IE = 1 enables a change in the CAN 0 status register to initiate an interrupt if the corresponding ERIE or STIE bit in the CAN 0 control register is set. C0IE = 0 disables a change in the CAN 0 status register from generating an interrupt.

EAIE
Bit 5

Ethernet activity interrupt enable. EAIE = 1 enables the Ethernet activity interrupt if the RIF or TIF bit in the BCUC register is set. EAIE = 0 disables generating an interrupt.

EWDI
Bit 4

Watchdog interrupt enable. Setting this bit to 1 enables interrupt requests generated by the watchdog timer. Clearing this bit to 0 disables the interrupt requests by the watchdog timer.

EWPI
Bit 3

Write-protected interrupt enable. Setting this bit to 1 enables interrupt requests generated by the WPIF flag in the MCON2. Clearing this bit to 0 disables the write-protected interrupt request.

ES2
Bit 2

Serial port 2 interrupt enable. Setting this bit to 1 enables interrupt requests generated by the RI_2 or TI_2 flags in SCON2. Clearing this bit to 0 disables serial port 2 interrupts.

ET3
Bit 1

Timer 3 interrupt enable. Setting this bit to 1 enables interrupts from timer 3 TF3 flag in T3CM. Clearing this bit to 0 disables all timer 3 interrupts.

EX2-5
Bit 0

External interrupt 2-5 enable. Setting this bit to 1 enables interrupt requests generated by the IE2, IE3, IE4, or IE5 flag in EXIF. Clearing this bit to 0 disables the external interrupt 2 to 5.

MOVX Address Extended Register (MXAX)

	7	6	5	4	3	2	1	0
SFR EAh	MXAX.7	MXAX.6	MXAX.5	MXAX.4	MXAX.3	MXAX.2	MXAX.1	MXAX.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

MXAX.7–0
Bits 7–0

MOVX address extended register. This register is used to provide the extended address byte to complement the low byte address provided by the indirect addressing of the Ri register. Using the address values in the P2 and MXAX and the address value indirectly specified by the Ri register allows the processor to access the full 24-bit data address range when executing a MOVX @Ri, A or MOVX A, @Ri instruction. The DPTR-related MOVX instructions do not utilize the P2 and MXAX register. Note that the MXAX register is only used for 24-bit addressing when the processor is operating in either the 24-bit paged or 24-bit contiguous modes. It can be utilized as a scratchpad SRAM register in 16-bit address mode.

Data Pointer Extended Register 2 (DPX2)

	7	6	5	4	3	2	1	0
SFR EBh	DPX2.7	DPX2.6	DPX2.5	DPX2.4	DPX2.3	DPX2.2	DPX2.1	DPX2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPX2.7–0
Bits 7–0

Data pointer extended byte 2. This register contains the high-order byte of the extended 24-bit address for auxiliary data pointer 2. This register is used only in the 24-bit paged and contiguous addressing modes. This register is not used for addressing the data memory in the 16-bit addressing mode and, therefore, can be utilized as a scratchpad SRAM register.

Data Pointer Extended Register 3 (DPX3)

	7	6	5	4	3	2	1	0
SFR EDh	DPX3.7	DPX3.6	DPX3.5	DPX3.4	DPX3.3	DPX3.2	DPX3.1	DPX3.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPX3.7–0
Bits 7–0

Data pointer extended byte 3. This register contains the high-order byte of the extended 24-bit address for auxiliary data pointer 3. This register is used only in the 24-bit paged and contiguous addressing modes. This register is not used for addressing the data memory in the 16-bit addressing mode and, therefore, can be utilized as a scratchpad SRAM register.

1-Wire Master Address Register (OWMAD)

	7	6	5	4	3	2	1	0
SFR EEh	—	—	—	—	—	OWMAD.2	OWMAD.1	OWMAD.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

Bits 7–3

Reserved. (Read returns all zeros.)

OWMAD.2-0
Bits 2-0

1-Wire master address select bits 2-0. These bits are used to select one of the five 1-Wire master registers to be accessed by the OWMDR SFR. Prior to accessing any of the 1-Wire

master's registers, the address of the target register must be specified as following:

A2	A1	A0	REGISTER	ACCESS MODE
0	0	0	Command register	Read/write
0	0	1	Receive/transmit buffer	Read (receive)/write (transmit)
0	1	0	Interrupt flag register	Read
0	1	1	Interrupt enable register	Read/write
1	0	0	Clock divisor register	Read/write
1	0	1	Control register	Read/write

The 1-Wire master supports only the above address values. When these bits are set to states other than those listed above, read data in the OWMDR is invalid, and write data to the OWMDR does not change the logic state of any of the five registers. Note that the default values for these bits are set to 111b.

1-Wire Master Data Register (OWMDR)

	7	6	5	4	3	2	1	0
SFR EFh	OWMDR.7	OWMDR.6	OWMDR.5	OWMDR.4	OWMDR.3	OWMDR.2	OWMDR.1	OWMDR.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

OWMDR.7–0
Bits 7–0

1-wire master data register. This register contains the data value of the target register as selected by the A2:A0 bits in the OWMAD SFR, when read to the OWMDR. A write to the OWMDR causes a write access to the target register as selected by the A2:A0 bit in the OWMAD SFR, and updates the target register with the new data.

B Register (B)

	7	6	5	4	3	2	1	0
SFR F0h	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

B.7–0
Bits 7–0

B register. This register serves as a second accumulator for certain arithmetic operations. It is functionally identical to the B register found in the 80C32.

Slave Address Mask Enable Register 2 (SADEN2)

	7	6	5	4	3	2	1	0
SFR F1h	SADEN2.7	SADEN2.6	SADEN2.5	SADEN2.4	SADEN2.3	SADEN2.2	SADEN2.1	SADEN2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SADEN2.7–0
Bits 7–0

Slave address mask enable register 2. This register is a mask enable when comparing serial port 2 addresses for automatic address recognition. When a bit is set in this register, the corresponding bit location in the SADDR2 register is exactly compared with the incoming serial port 2 data to determine if a receive interrupt should be generated. When a bit in this register is cleared, the corresponding bit in the SADDR2 register becomes a "don't care" and is not compared against the incoming data. All incoming data generates a receive interrupt when this register is cleared.

Data Pointer Low Register 2 (DPL2)

	7	6	5	4	3	2	1	0
SFR F2h	DPL2.7	DPL2.6	DPL2.5	DPL2.4	DPL2.3	DPL2.2	DPL2.1	DPL2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPL2.7–0
Bits 7–0

Data pointer low byte 2. This register is the low byte of the auxiliary data pointer, and contains the low-order byte of the 24-bit data address.

Data Pointer High Register 2 (DPH2)

	7	6	5	4	3	2	1	0
SFR F3h	DPH2.7	DPH2.6	DPH2.5	DPH2.4	DPH2.3	DPH2.2	DPH2.1	DPH2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPH2.7–0
Bits 7–0

Data pointer high byte 2. This register is the high byte of auxiliary data pointer 2, and contains the middle-order byte of the 24-bit data address.

Data Pointer Low Register 3 (DPL3)

	7	6	5	4	3	2	1	0
SFR F4h	DPL3.7	DPL3.6	DPL3.5	DPL3.4	DPL3.3	DPL3.2	DPL3.1	DPL3.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPL3.7–0
Bits 7–0

Data pointer low byte 3. This register is the low byte of the auxiliary data pointer 3, and contains the low-order byte of the 24-bit data address.

Data Pointer High Register 3 (DPH3)

	7	6	5	4	3	2	1	0
SFR F5h	DPH3.7	DPH3.6	DPH3.5	DPH3.4	DPH3.3	DPH3.2	DPH3.1	DPH3.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

DPH3.7–0
Bits 7–0

Data pointer high byte 3. This register is the high byte of auxiliary data pointer 3, and contains the middle-order byte of the 24-bit data address.

Data Pointer Select Register 1 (DPS1)

	7	6	5	4	3	2	1	0
SFR F6h	ID3	ID2	—	—	—	—	—	—
	RW-0	RW-0	R-1	R-1	R-1	R-1	R-1	R-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

ID3
Bit 7

Increment/decrement data pointer 3. This bit defines how the INC DPTR instruction functions in relation to data pointer 3 when it is selected by SEL1 and SEL bits (SEL1, SEL = 11). When ID3 is set to logic 1, the INC DPTR instruction actually decrements the content of data pointer 3 by 1.

When ID3 is cleared to 0, the INC DPTR instruction increments the content of data pointer 3 by 1.

ID2 (Bit 6) — **Increment/decrement data pointer 2.** This bit defines how the INC DPTR instruction functions in relation to data pointer 2, when it is selected by SEL1 and SEL bits (SEL1, SEL = 10). When ID2 is set to logic 1, the INC DPTR instruction actually decrements the content of data pointer 2 by 1. When ID2 is cleared to 0, the INC DPTR instruction increments the content of data pointer 2 by 1.

Bits 5–0 — **Reserved.** (Read returns all one's.)

Status Register 1 (STATUS1)

	7	6	5	4	3	2	1	0
SFR F7	—	—	—	—	V1PF	V3PF	SPTA2	SPRA2
	R-1	R-1	R-1	R-1	R-0	R-0	R-0	R-0

R = Unrestricted read, -n = Value after reset

Bits 7–0 — **Reserved.**

V1PF (Bit 3) — **Vcc1 power-fail.** When set, this bit indicates that the voltage level of Vcc1 has fallen below Vpfw1. Hardware setting of this bit forces PFI bit (WDCON.4) to 1. V1PF is cleared when PFI bit is cleared.

V3PF (Bit 2) — **Vcc3 power-fail.** When set, this bit indicates that the voltage level of Vcc3 has fallen below Vpfw3. Hardware setting of this bit forces PFI bit (WDCON.4) to 1. V3PF is cleared when PFI bit is cleared.

SPTA2 (Bit 1) — **Serial port 2 transmit activity monitor.** When set, this bit indicates that data is currently being transmitted by serial port 2. It is cleared when TI_2 is set.

SPRA2 (Bit 0) — **Serial port 2 receive activity monitor.** When set, this bit indicates that data is currently being received by serial port 2. It is cleared when RI_2 bit is set.

Extended Interrupt Priority (EIP)

	7	6	5	4	3	2	1	0
SFR F8h	EPMIP	C0IP	EAIP	PWDI	PWPI	PS2	PT3	PX2-5
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

EPMIP (Bit 7) — **Ethernet power mode interrupt priority.**
EPMIP = 1 selects the Ethernet power mode interrupt source as a high priority.
EPMIP = 0 selects the Ethernet power mode interrupt source as a low priority.

C0IP (Bit 6) — **CAN 0 interrupt LSB priority control.**
C0IP = 1 selects the CAN 0 status register source as a high priority.
C0IP = 0 selects the CAN 0 status register source as a low priority.

EAIP (Bit 5) — **Ethernet activity interrupt priority.**
EAIP = 1 selects the Ethernet activity interrupt as a high priority.
EAIP = 0 selects the Ethernet activity interrupt as a low priority.

PWDI (Bit 4) — **Watchdog interrupt priority.** Watchdog interrupt is a high priority when this bit is set to 1, and is a low priority when this bit is cleared to 0.

PWPI (Bit 3) — **Write-protected interrupt priority.** Write-protected interrupt is a high priority when this bit is set to 1, and is a low priority when this bit is cleared to 0.

PS2
Bit 2
Serial port 2 interrupt priority. Setting this bit to 1 selects serial port 0 interrupt source as a high priority, a 0 selects it as a low priority.

PT3
Bit 1
Timer 3 interrupt priority. Setting this bit to 1 selects timer 3 interrupt source as a high priority, a 0 selects it as a low priority.

PX2-5
Bit 0
External interrupt 2–5 priority. External interrupts 2–5 and 1-Wire bus master interrupt (if EOWMI = 1) are high priority when this bit is set to 1, and are low priority when this bit is cleared to 0.

Parallel I/O Port 7 (P7)

	7	6	5	4	3	2	1	0
SFR F9h	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0
	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

R = Unrestricted read, W = Unrestricted write, -n = Value after reset (P7._ above)

P7.7–0
Bits 7–0
Port 7 bits 7–0. This port is a programmable parallel I/O port. Data written to the port latch serves to set both logic level and direction of the data on the pin. A 1 written to a port latch, previously programmed to a 0, activates a high-current, one-shot pullup on the corresponding pin. This is followed by a static, low-current pullup, which remains on until the port is changed again. The final high state of the port pin is considered a pseudo-input mode, and can be easily over driven from an external source. Port latches previously in a high-output state do not change, nor does the high-current one-shot fire when a 1 is loaded. Loading a 0 to a port latch results in a static, high-current pulldown on the corresponding pin. This mode is termed the I/O output state, since no weak devices are used to drive the pin. Port 7 functions as the nonmultiplexed external address output port for addresses A0–A7 when MUX = 1.

Timer 3 LSB (TL3)

	7	6	5	4	3	2	1	0
SFR FBh	TL3.7	TL3.6	TL3.5	TL3.4	TL3.3	TL3.2	TL3.1	TL3.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TL3.7–0
Bits 7–0
Timer 3 LSB. This register is used to load and read the least significant 8-bit value in timer 3.

Timer 3 MSB (TH3)

	7	6	5	4	3	2	1	0
SFR FCh	TH3.7	TH3.6	TH3.5	TH3.4	TH3.3	TH3.2	TH3.1	TH3.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TH3.7–0
Bits 7–0
Timer 3 MSB. This register is used to load and read the most significant 8-bit value in timer 3.

Timer 3 Control/Mode Register (T3CM)

	7	6	5	4	3	2	1	0
SFR FDh	TF3	TR3	T3M	SMOD_2	GATE	C/$\overline{T3}$	M1	M0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

TF3
Bit 7

Timer 3 overflow flag. This bit is set to 1 when timer 3 overflows its maximum count, as defined by the current mode. It is cleared either by software, or by the start of the timer 3 interrupt service routine. A zero on this bit indicates that no timer 3 overflow has been detected.

TR3
Bit 6

Timer 3 run control. Setting this bit enables timer 3. Clearing this bit halts the timer 3.

T3M
Bit 5

Timer 3 clock select. This bit controls the division of the system clock that drives timer 3. This bit has no effect on instruction cycle timing.

0 = Timer 3 uses a divide-by-12 of the crystal frequency.
1 = Timer 3 uses a divide-by-4 of the system clock frequency.

SMOD_2
Bit 4

Serial port 2 baud-rate doubler enable. Setting this bit enables the serial baud-rate doubling function in mode 1, 2 and 3 for serial port 2. A 0 disables the doubler.

GATE
Bit 3

Timer 3 gate control.

GATE = 0: Timer 3 clocks when TR3 is 1, regardless of INT3.
GATE = 1: Timer 3 clocks only when TR1 and INT3 are 1.

C/$\overline{T3}$
Bit 2

Counter/timer 1 select.

C/$\overline{T3}$ = 0: Selects timer function with internal clock for timer 3.
C/$\overline{T3}$ = 1: Selects counter function with input from T3 when TR3 is 1.

M1-0
Bits 1-0

Timer 3 mode-select bits 1 and 0.

M1	M0	TIMER MODE
0	0	Mode 0: 8-bit with 5-bit prescale
0	1	Mode 1: 16-bit with no prescale
1	0	Mode 2: 8-bit with autoreload
1	1	Mode 3: Timer 3 is halted, but its count is held.

Serial Port 2 Control Register (SCON2)

	7	6	5	4	3	2	1	0
SFR FEh	SM0/FE_2	SM1_2	SM2_2	REN_2	TB8_2	RB8_2	TI_2	RI_2
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

SM0/FE_2
Bit 7

Serial port 2 mode bit 0. When SMOD0 is set to 1, it is the framing error flag that is set upon detection of an invalid stop bit and must be cleared by software. Modification of this bit when SMOD0 is set has no effect on the serial mode setting.

MODE	SM2	SM1	SM0	FUNCTION	LENGTH	PERIOD
0	0	0	0	Synchronous	8 bits	12 t_{CLK}
0	1	0	0	Synchronous	8 bits	4 t_{CLK}
1	—	1	0	Asynchronous	10 bits	Timer 3
2	0	0	1	Asynchronous	11 bits	64 t_{CLK} (SMOD_1 = 0) 32 t_{CLK} (SMOD_1 = 1)
2	1	0	1	Asynchronous (MP)	11 bits	64 t_{CLK} (SMOD_1 = 0) 32 t_{CLK} (SMOD_1 = 1)
3	0	1	1	Asynchronous	11 bits	Timer 3
3	1	1	1	Asynchronous (MP)	11 bits	Timer 3

SM1_2
Bit 6
Serial port 2 mode bit 1.

SM2_2
Bit 5
Serial port 2 mode bit 2. Setting of this bit in mode 1 ignores reception if an invalid stop bit is detected. Setting this bit in mode 2 or 3 enables multiprocessor communications. This prevents the RI_2 bit from being set and an interrupt being asserted, if the 9th bit received is 0.

REN_2
Bit 4
Receive enable.
REN_0 = 0: Serial port 2 reception disabled.
REN_0 = 1: Serial port 2 receiver enabled for modes 1, 2, and 3. Initiate synchronous reception for mode 0.

TB8_2
Bit 3
9th transmission bit state. This bit defines the state of the 9th transmission bit in serial port 2, modes 2 and 3.

RB8_2
Bit 2
9th received bit state. This bit identifies the state of the 9th bit of received data in serial port 2, modes 2 and 3. When SM2_2 is 0, it is the state of the stop bit in mode 1. This bit has no meaning in mode 0.

TI_2
Bit 1
Transmit interrupt flag. This bit indicates that the data in the serial port 2 buffer has been completely shifted out. It is set at the end of the last data bit for all modes of operation and must be cleared by software.

RI_2
Bit 0
Receive interrupt flag. This bit indicates that a data byte has been received in the serial port 2 buffer. It is set at the end of the 8th bit for mode 0, after the last sample of the incoming stop bit for mode 1 subject to the value of the SM2_2 bit, or after the last sample of RB8_2 for modes 2 and 3. This bit must be cleared by software.

Serial Data Buffer 2 (SBUF2)

R = Unrestricted read, W = Unrestricted write, -n = Value after reset

	7	6	5	4	3	2	1	0
SFR FFh	SBUF2.7	SBUF2.6	SBUF2.5	SBUF2.4	SBUF2.3	SBUF2.2	SBUF2.1	SBUF2.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

SBUF2.7–0
Bits 7–0
Serial data buffer 2. Data for serial port 2 is read from or written to this location. The serial transmit and receive buffers are separate registers, but both are addressed at this location.

ADDENDUM TO SECTION 5: CPU TIMING

EXTERNAL CLOCK SOURCE

The DS80C400 supports a maximum operating frequency of 75MHz. However, when using an external crystal, the frequency must not exceed 40MHz in order for the internal oscillator circuitry to work properly. Thus, the maximum operating frequency can be achieved in one of two ways: 1) use of a stand-alone clock oscillator or clock source (up to 75MHz) to directly drive the XTAL1 pin or 2) use of the on-chip clock multiplier circuitry (described later) to 4X/$\overline{2X}$ multiply the external crystal frequency.

SYSTEM CLOCK SELECTION

The internal clocking options of the DS80C400 differ slightly from that described in the *High-Speed Microcontroller User's Guide*. Most members of the family offer the option of 4, 256, or 1024 clocks per machine cycle. The DS80C400 can operate at 1, 2, 4, or 1024 oscillator clocks per machine cycle. The system clock divide control function is shown in Figure 5-1. A 3:1 multiplexer, controlled by CD1, CD0 (PMR.7-6), selects one of three sources for the internal system clock:

- Crystal oscillator or external clock source
- Crystal oscillator or external clock source divided by 256
- Crystal oscillator or external clock source frequency multiplied by 2 or 4

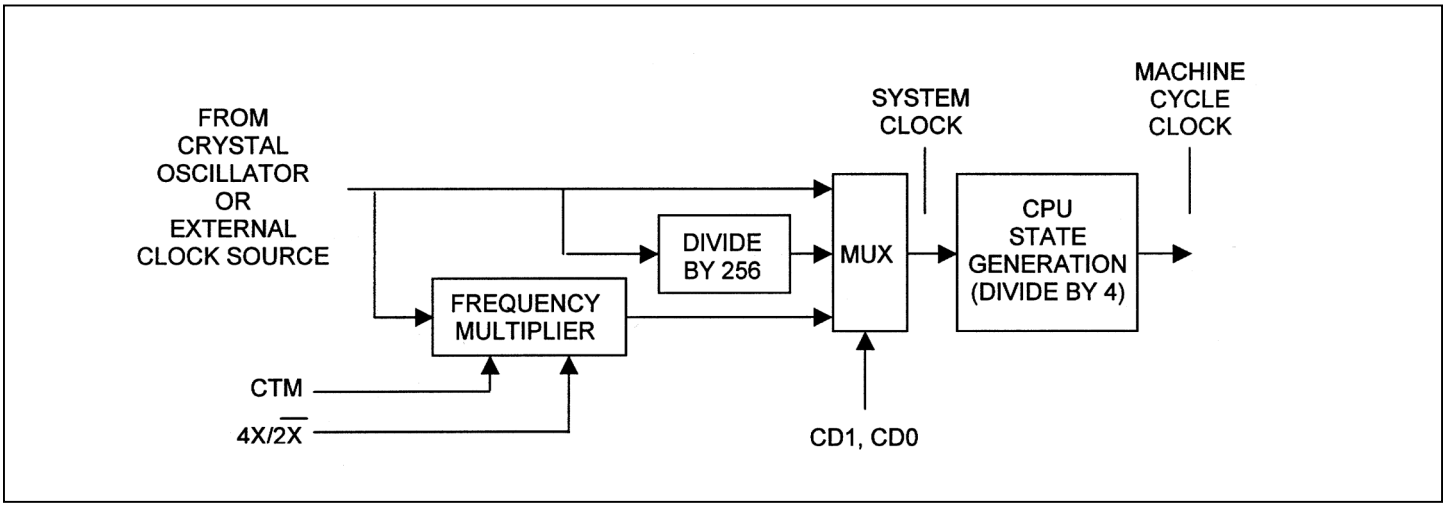


Figure 5-1. System Clock Control Diagram

The system clock control circuitry generates two clock signals that are used by the microcontroller. The internal system clock provides the time base for timers and internal peripherals. The system clock is run through a divide-by-4 circuit to generate the machine cycle clock that provides the time base for CPU operations. All instructions execute in one to six machine cycles. It is important to note the distinction between these two clock signals as they are sometimes confused, creating errors in timing calculations.

Setting CD1:0 to 00b enables the frequency multiplier, either doubling or quadrupling the frequency of the crystal oscillator or external clock source. The 4X/$\overline{2X}$ bit controls the multiplying factor, selecting two or four times the frequency when set to 0 or 1, respectively. Enabling the frequency multiplier results in apparent instruction execution speeds of 2 or 1 oscillator clocks. Regardless of the configuration of the frequency multiplier, the system clock of the microcontroller can never be operated faster than 75MHz. This means that the maximum crystal oscillator or external clock source is 18.75MHz when using the 4X setting, and 37.5MHz when using the 2X setting.

The primary advantage of the clock multiplier is that it allows the microcontroller to use slower crystals to achieve the same performance level. This reduces EMI and cost, as slower crystals are generally more available and, therefore, less expensive.

Table 5-1. System Clock Configuration

CD1	CD0	4X/2$\overline{X}$	NAME	CLOCKS/MC	MAX EXTERNAL FREQUENCY (MHZ)
0	0	1	Frequency Multiplier (4x)	1	18.75
0	0	0	Frequency Multiplier (2x)	2	37.5
0	1	N/A	Reserved	—	—
1	0	N/A	Divide-by-four (default)	4	75
1	1	N/A	Power Management Mode	1024	75

The system clock and machine cycle rate changes one machine cycle after the instruction that changes the control bits. Note that the change affects all aspects of system operation, including timers and baud rates. The use of the switchback feature, described later, can eliminate many of the issues associated with the power-management mode's affect on peripherals such as the serial port. Table 5-2 illustrates the effect of the clock modes on the operation of the timers:

Table 5-2. Effect of Clock Modes on Timer Operation

CD1	CD0	4X/2$\overline{X}$	OSC CYCLES PER MACHINE CYCLE	OSC CYCLES PER TIMER 0/1/2/3 CLOCK		OSC CYCLES PER TIMER 2 CLOCK, BAUD RATE GEN		OSC CYCLES PER SERIAL PORT CLOCK MODE 0		OSC CYCLES PER SERIAL PORT CLOCK MODE 2	
				TXM = 0	TXM = 1	TXM=0	TXM = 1	SM2=0	SM2 = 1	SMOD = 0	SMOD = 1
0	0	1	1	12	1	2	2	3	1	64	32
0	0	0	2	12	2	2	2	6	2	64	32
0	1	N/A	4 (reserved)	12	4	2	2	12	4	64	32
1	0	N/A	4 (default)	12	4	2	2	12	4	64	32
1	1	N/A	1024	3072	1024	512	512	3072	1024	16,384	8192

CHANGING THE SYSTEM CLOCK/MACHINE CYCLE CLOCK FREQUENCY

The microcontroller incorporates a special locking sequence to ensure "glitch-free" switching of the internal clock signals. All changes to the CD1, CD0 bits must pass through the 10 (divide-by-4) state. For example, to change from 00 (frequency multiplier) to 11 (PMM), the software must change the bits in the following sequence: 00 ⇒ 10 ⇒ 11. Attempts to switch between invalid states fail, leaving the CD1, CD0 bits unchanged.

The following sequence must be followed when switching to the frequency multiplier as the internal time source. This sequence can only be performed when the device is in divide-by-4 operation. The steps must be followed in this order, although it is possible to have other instructions between them. Any deviation from this order causes the CD1, CD0 bits to remain unchanged. Switching from frequency multiplier to nonmultiplier mode requires no steps other than the changing of the CD1, CD0 bits.

1) Ensure that the CD1, CD0 bits are set to 10, and the RGMD (EXIF.2) bit = 0.
2) Clear the Crystal Multiplier Enable (CTM) bit.
3) Set the 4X/$\overline{2X}$ bit to the appropriate state.
4) Set the CTM bit.
5) Poll the CKRDY bit (EXIF.3), waiting until it is set to 1. This takes approximately 65536 cycles of the external crystal or clock source.
6) Set CD1, CD0 to 00. The frequency multiplier is engaged on the machine cycle following the write to these bits.

ADDENDUM TO SECTION 6: MEMORY ACCESS

INTERNAL PROGRAM MEMORY

The DS80C400 incorporates 64kB of on-chip ROM program memory. The 64kB block of memory is logically divided into two 32kB blocks. The upper 32kB block, which is reserved for internal use, is always mapped to the very top of the 16MB program memory space (FF8000h–FFFFFFh). The logical address location for the lower 32kB block, the TINI®400 (Tiny InterNet Interfaces) ROM, is controlled by the merge ROM (MROM) bit of the address control (ACON: 9Dh) register. The functionality implemented by the TINI400 ROM is covered in a separate section of this supplement. The reset default location for the 32kB TINI400 ROM, when MROM = 0, is 000000h–007FFFh. When MROM is set (MROM = 1), the 32kB block is then logically mapped to the range FF0000h–FF7FFFh.

Two control mechanisms, the $\overline{EA}$ pin and the bypass ROM (BROM) SFR bit, dictate whether the ROM is executed, or even included in the memory map. No matter the state of the BROM bit, if the $\overline{EA}$ pin is held at a logic low level, the TINI400 ROM code is not entered and is not accessible as program memory. If the $\overline{EA}$ pin is at a logic high level, the BROM bit is then examined to determine whether the internal TINI400 firmware should be executed or bypassed. If BROM = 0, the TINI400 code is executed. Otherwise, (BROM = 1), the TINI400 code is bypassed, and execution is transferred to external user code at address 000000h. The BROM bit defaults to 0 on a power-on reset, but is unaffected by other reset sources. Figure 6-1 shows the possible program memory map alternatives.

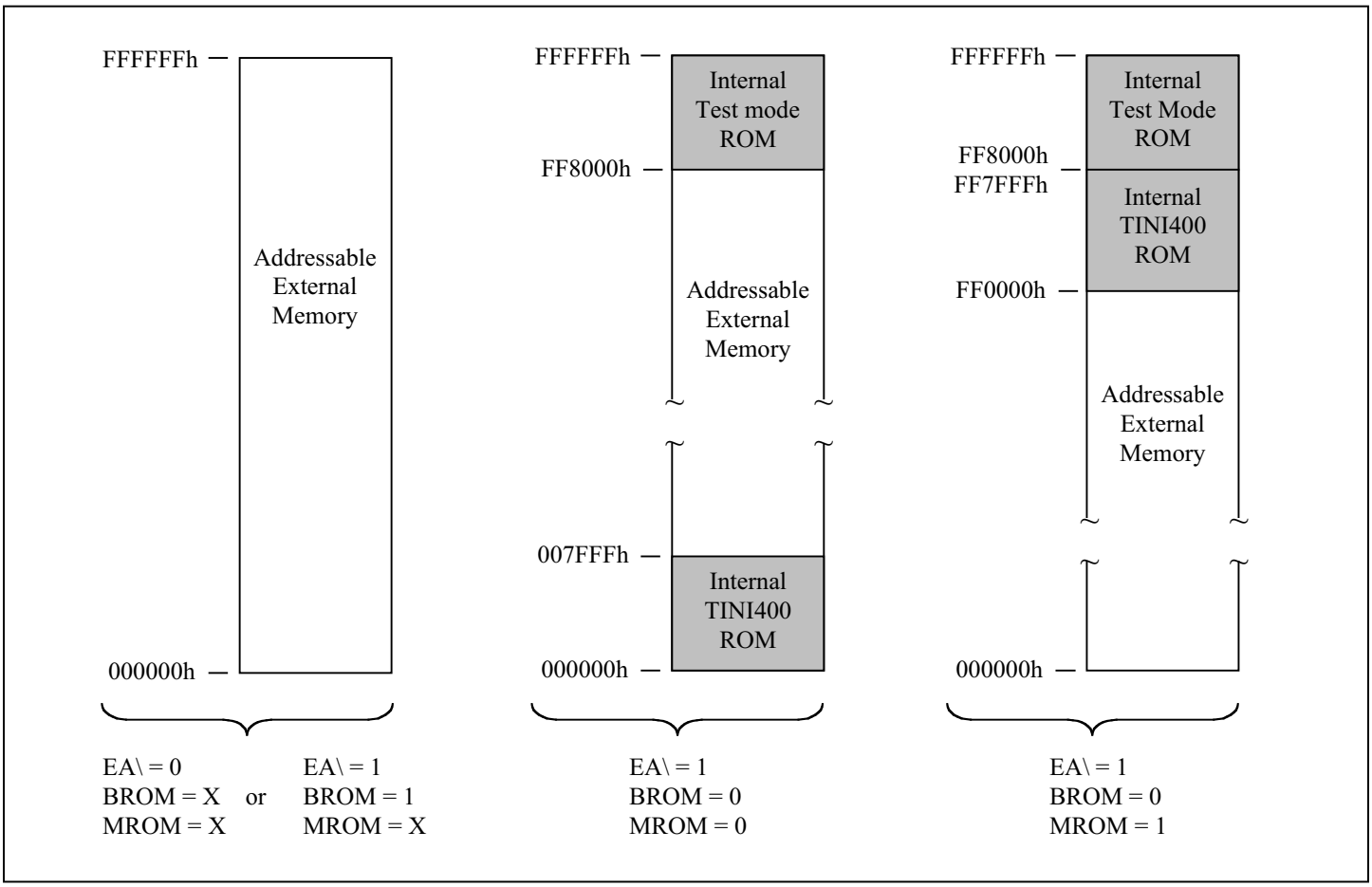


Figure 6-1. Program Memory Map Options

TINI is a registered trademark of Dallas Semiconductor.

INTERNAL DATA MEMORY

The DS80C400 incorporates 9472 bytes of internal SRAM memory, in addition to the standard 256-byte scratchpad memory. This additional on-chip SRAM is logically divided into three memory blocks: a 1kB block usable as data memory and extended stack memory, an 8kB block usable as data memory and Ethernet transmit/receive buffer memory, and a 256-byte block usable as data memory and CAN controller memory. In order for the 1kB internal SRAM to be used as extended stack memory, the stack address mode (SA) bit, contained in the ACON register, must be set to 1. The logical address location for each block is determined by the settings of the IDM1, IDM0 bits and the CMA bit, all contained in the MCON (C6h) register. The following table summarizes the six possible configurations for the three internal SRAMs, while Figure 6-2 illustrates three data memory map possibilities.

Table 6-1. Internal Data Memory Address Locations

IDM1:0	CMA	1kB SRAM (OPTIONAL STACK)	8kB SRAM (ETHERNET BCU)	256–BYTE SRAM (CAN)
00	0	00DC00h–00DFFFh	00E000h–00FFFFh	00DB00h–00DBFFh
00	1	00DC00h–00DFFFh	00E000h–00FFFFh	FFDB00h–FFDBFFh
01	0	002000h–0023FFh	000000h–001FFFh	00DB00h–00DBFFh
01	1	002000h–0023FFh	000000h–001FFFh	FFDB00h–FFDBFFh
10	0	FFDC00h–FFDFFFh	FFE000h–FFFFFFh	00DB00h–00DBFFh
10	1	FFDC00h–FFDFFFh	FFE000h–FFFFFFh	FFDB00h–FFDBFFh

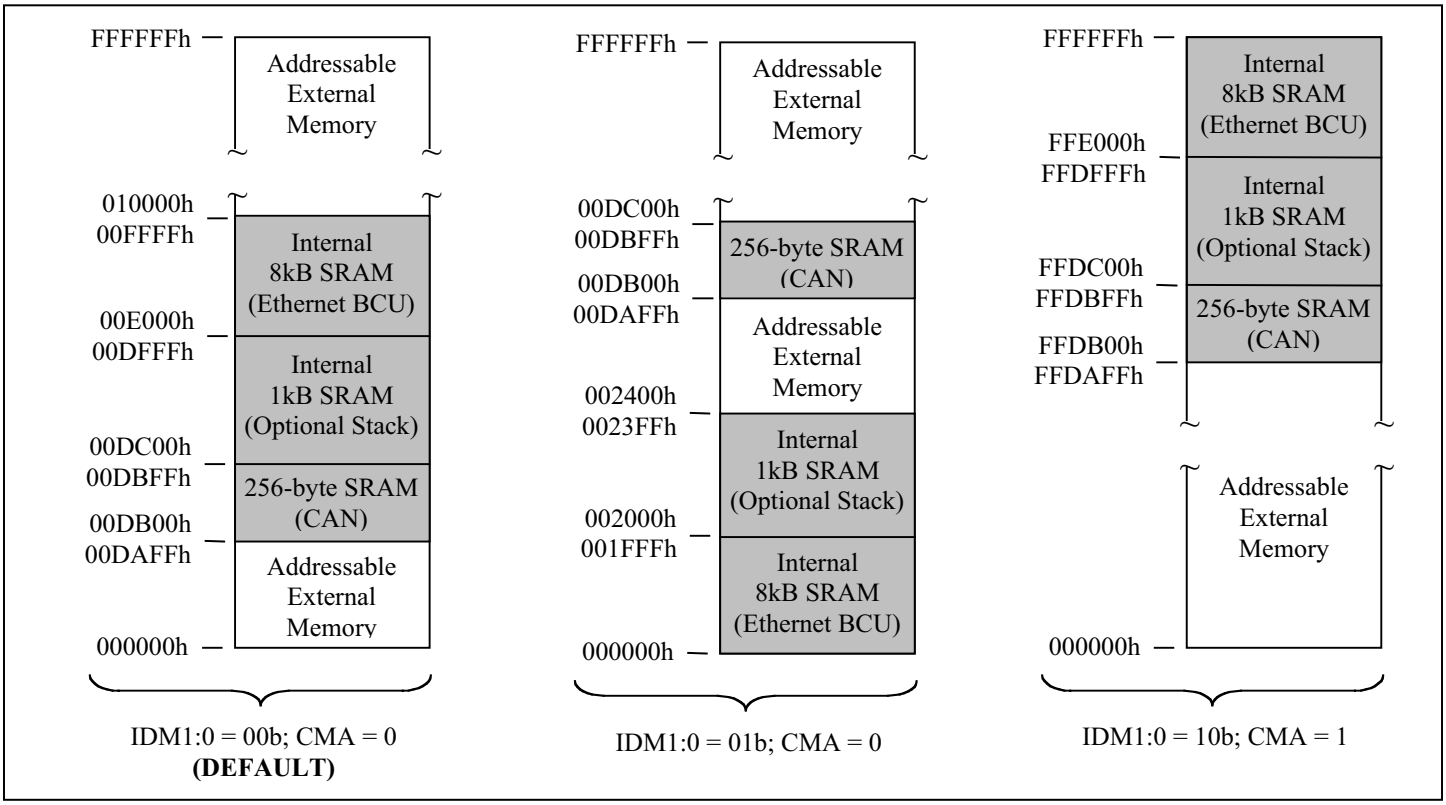


Figure 6-2. Example Data Memory Map Configurations

EXTERNAL MEMORY ACCESS

The DS80C400 follows the memory interface convention established by the industry standard 80C32/80C52, but with many added improvements. Most notably, the device incorporates a 24-bit addressing capability that directly supports up to 16MB of program memory and 4MB of data memory. Externally, the memory is accessed through a multiplexed or demultiplexed 22-bit address bus/8-bit data bus and eight chip-enable signals (active during program memory access) or four peripheral-enable signals (active during data mem-

ory access). Multiplexed addressing mode mimics the traditional 8051 memory interface, with the address MSB presented on port 2 and the address LSB and data multiplexed on port 0. The multiplexed mode requires an external latch to demultiplex the address LSB and data. When the MUX pin is pulled high, the address LSB and data are demultiplexed, with the address MSB presented on port 2, the address LSB on port 7, and the data on port 0. The elimination of the demultiplexing latch removes a delay element in the memory timing and can, in some cases, allow the use of slower, less expensive memory devices. The following table illustrates the locations of the external memory control signals.

Table 6-2. External Memory Addressing Pin Assignments

	SIGNAL	MULTIPLEXED ($\overline{MUX}$ = 0)	DEMULTIPLEXED ($\overline{MUX}$ = 1)
ADDRESS	A21	P6.5	P6.5
	A20	P6.4	P6.4
	A19	P4.7	P4.7
	A18	P4.6	P4.6
	A17	P4.5	P4.5
	A16	P4.4	P4.4
	A15–A8	P2.7–P2.0	P2.7–P2.0
	A7–A0	**P0.7–P0.0**	**P7.7–P7.0**
DATA	D7–D0	P0.7–P0.0	P0.7–P0.0
CHIP ENABLES	$\overline{CE7}$	P6.3	P6.3
	$\overline{CE6}$	P6.2	P6.2
	$\overline{CE5}$	P6.1	P6.1
	$\overline{CE4}$	P6.0	P6.0
	$\overline{CE3}$	P4.3	P4.3
	$\overline{CE2}$	P4.2	P4.2
	$\overline{CE1}$	P4.1	P4.1
	$\overline{CE0}$	P4.0	P4.0
PERIPHERAL CHIP ENABLES	$\overline{PCE2}$	P5.7	P5.7
	$\overline{PCE1}$	P5.6	P5.6
	$\overline{PCE0}$	P5.5	P5.5
		P5.4	P5.4

Each upper-order address line (A16–A21) and chip or peripheral enable is individually enabled by the P4CNT, P5CNT, and P6CNT registers. Enabling upper-order address lines increases the maximum size of the external memories that can be addressed, and enabling chip or peripheral enables controls the number of external memories that can be addressed. For example, if P4CNT.5-3 are set to 010b, A17 and A16 are enabled (along with A15–A0), permitting each chip-enable access a maximum memory device size of 2^{18} or 256kB. Note that the desired chip-enable signals must be enabled in order to become active for a defined memory range.

The configurable program/code chip-enable ($\overline{CEx}$) and MOVX chip-enable ($\overline{PCEx}$) signals issued by the microcontroller are used when accessing multiple external memory devices. External chip-enable lines are only required if more than one physical block of memory are used. In the standard 8051 configuration, $\overline{PSEN}$ is used as the output enable for the program memory device, and $\overline{RD}$ and $\overline{WR}$ control the input or output functions of the data (SRAM) device. Typically, the chip enables of the program and data memory devices can be connected to their active state if only one of each is used. To support a larger amount of memory, however, the microcontroller must generate chip or data enables to select one of several memory devices. The following tables demonstrate how to enable various combinations of high-order address lines and chip enables.

Table 6-3. Extended Address Generation

P4CNT.5-3	P6.5	P6.4	P4.7	P4.6	P4.5	P4.4	MAX MEMORY ACCESSIBLE PER $\overline{CE}$
000	I/O	I/O	I/O	I/O	I/O	I/O	32kB[1]
001	I/O	I/O	I/O	I/O	I/O	A16	128kB
010	I/O	I/O	I/O	I/O	A17	A16	256kB
011	I/O	I/O	I/O	A18	A17	A16	512kB
100	I/O	I/O	A19	A18	A17	A16	1MB
101	I/O	A20	A19	A18	A17	A16	2MB
110 or 111 (default)	A21	A20	A19	A18	A17	A16	4MB[2]

[1] Only 32kB of memory is accessible per chip enable for the P4CNT.5-3 = 000b setting, which means, at least, two chip enables are needed in order to address the standard 16-bit (0–FFFFh) address range.

[2] The default P4CNT.5-3 = 111b setting (4MB accessible per $\overline{CE}$) requires only four chip enables in order to access the maximum 24-bit (0–FFFFFFh) address range.

Table 6-4. Chip-Enable Generation

	PORT PIN 6 FUNCTION					PORT PIN 4 FUNCTION			
P6CNT.2-0	P6.3	P6.2	P6.1	P6.0	P4CNT.2-0	P4.3	P4.2	P4.1	P4.0
000 (default)	I/O	I/O	I/O	I/O	000	I/O	I/O	I/O	I/O
100	I/O	I/O	I/O	$\overline{CE4}$	100	I/O	I/O	I/O	$\overline{CE0}$
101	I/O	I/O	$\overline{CE5}$	$\overline{CE4}$	101	I/O	I/O	$\overline{CE1}$	$\overline{CE0}$
110	I/O	$\overline{CE6}$	$\overline{CE5}$	$\overline{CE4}$	110	I/O	$\overline{CE2}$	$\overline{CE1}$	$\overline{CE0}$
111	$\overline{CE7}$	$\overline{CE6}$	$\overline{CE5}$	$\overline{CE4}$	111 (default)	$\overline{CE3}$	$\overline{CE2}$	$\overline{CE1}$	$\overline{CE0}$

Table 6-5. Peripheral Chip-Enable Generation

P5CNT.2-0	P5.7	P5.6	P5.5	P5.4	P6CNT.5-3	MAX MEMORY ACCESSIBLE PER $\overline{PCE}$
000 (default)	I/O	I/O	I/O	I/O	000	32kB
100	I/O	I/O	I/O	$\overline{PCE0}$	001	128kB
101	I/O	I/O	$\overline{PCE1}$	$\overline{PCE0}$	010	256kB
110	I/O	$\overline{PCE2}$	$\overline{PCE1}$	$\overline{PCE0}$	011	512kB
111	$\overline{PCE3}$	$\overline{PCE2}$	$\overline{PCE1}$	$\overline{PCE0}$	100	1MB

The following table illustrates how program memory is segmented based on the setting of the port 4 configuration control bits (P4CNT.5-3).

Table 6-6. Program Memory Chip-Enable Boundaries

P4CNT.5-3 →	000h (32kB/CE)	001h (128kB/CE)	010h (256kB/CE)	011h (512kB/CE)	100h (1MB/CE)	101h (2MB/CE)	101h (4MB/CE)
$\overline{CE0}$	0h–7FFFh	0h–1FFFFh	0h–3FFFFh	0h–7FFFFh	0h–FFFFFh	0h–1FFFFFh	0h–3FFFFFh
$\overline{CE1}$	8000h–FFFFh	20000h–3FFFFh	40000h–7FFFFh	80000h–FFFFFh	100000h–1FFFFFh	200000h–3FFFFFh	400000h–7FFFFFh
$\overline{CE2}$	10000h–17FFFh	40000h–5FFFFh	80000h–BFFFFh	100000h–17FFFFh	200000h–2FFFFFh	400000h–5FFFFFh	800000h–BFFFFFh
$\overline{CE3}$	18000h–FFFFh	60000h–7FFFFh	C0000h–FFFFFh	180000h–1FFFFFh	300000h–3FFFFFh	600000h–7FFFFFh	C00000h–FFFFFFh
$\overline{CE4}$	20000h–27FFFh	80000h–9FFFFh	100000h–13FFFFh	200000h–27FFFFh	400000h–4FFFFFh	800000h–9FFFFFh	—
$\overline{CE5}$	28000h–2FFFFh	A0000h–BFFFFh	140000h–17FFFFh	280000h–2FFFFFh	500000h–5FFFFFh	A00000h–BFFFFFh	—
$\overline{CE6}$	30000h–37FFFh	C0000h–DFFFFh	180000h–1BFFFFh	300000h–37FFFFh	600000h–6FFFFFh	C00000h–DFFFFFh	—
$\overline{CE7}$	38000h–3FFFFh	E0000h–FFFFFh	1C0000h–1FFFFFh	380000h–3FFFFFh	700000h–7FFFFFh	E00000h–FFFFFFh	—

Following any reset, the device defaults to 16-bit mode addressing (ACON.1-0 = 00b), with P6.5, P6.4, and P4.7–P4.4 serving as address lines (P4CNT.5-3 = 111b) and P4.3–P4.0 configured as $\overline{CE3\text{-}0}$ (P4CNT.2-0 = 111b). The first program fetch is performed from 000000h with $\overline{CE0}$ active (low).

Figures 6-3 and 6-4 illustrate the multiplexed ($\overline{MUX}$ = 0) and demultiplexed ($\overline{MUX}$ = 1) external memory interfaces. Both examples access 512kB of program memory and 256kB of data memory.

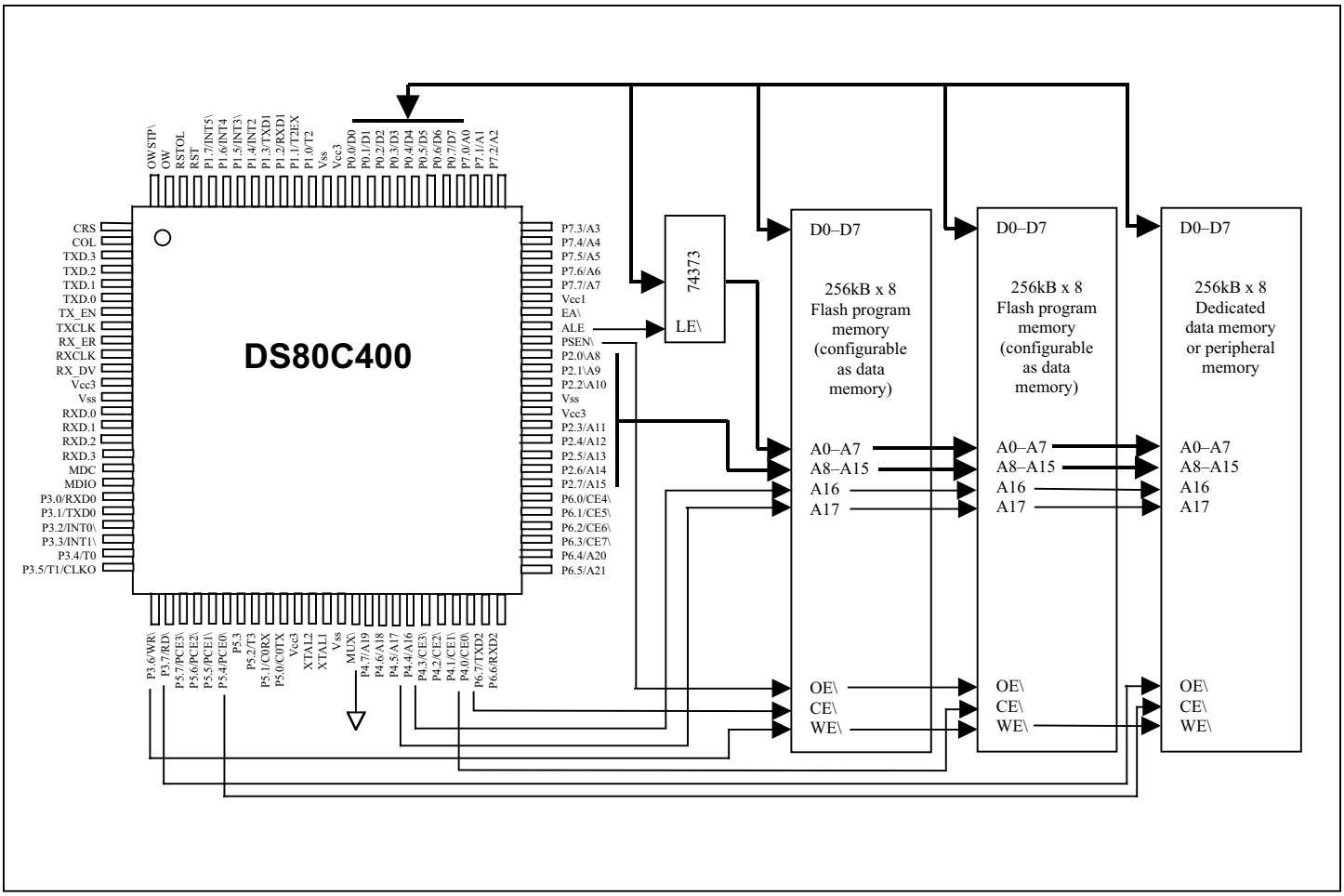


Figure 6-3. Multiplexed Address/Data Bus

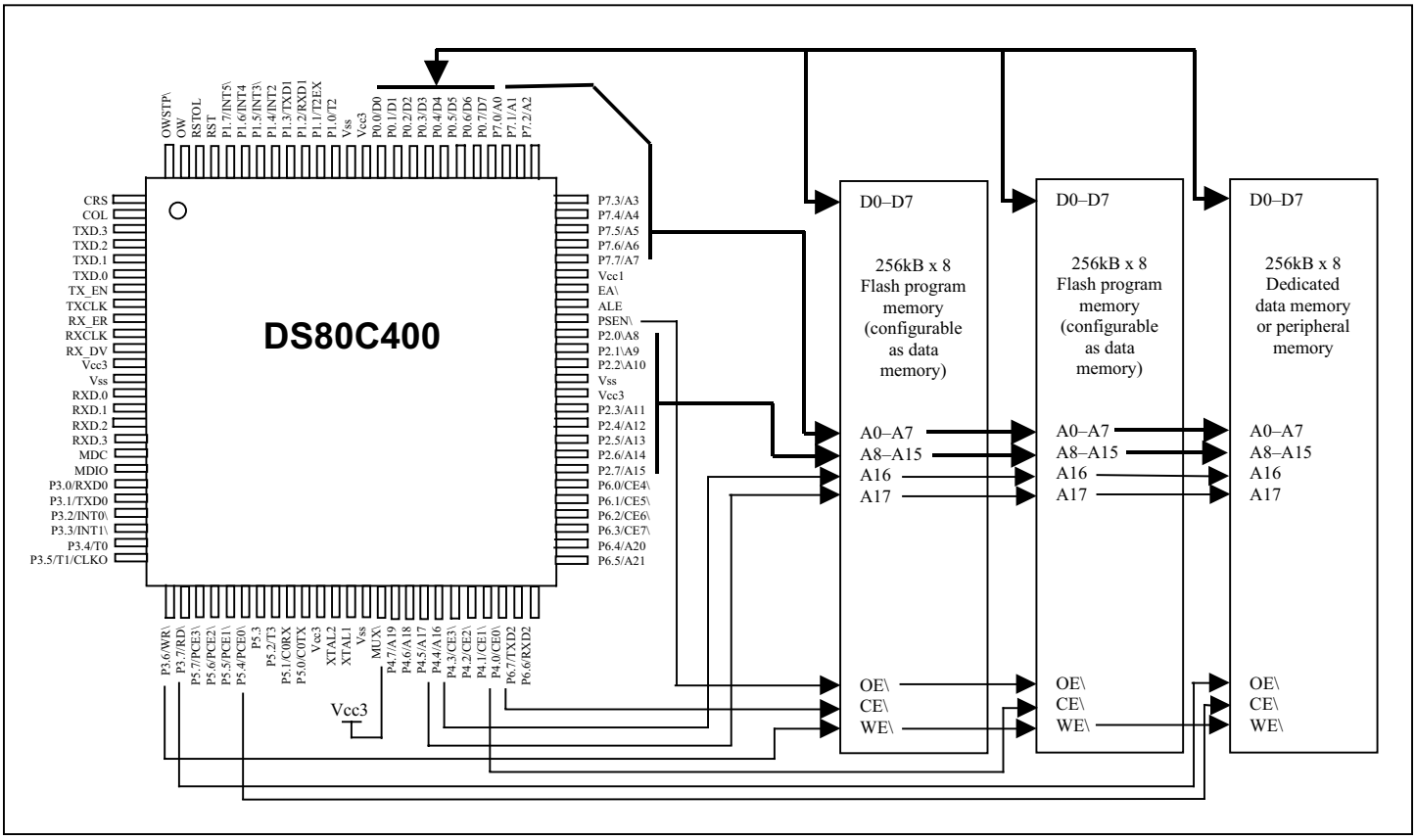


Figure 6-4. Demultiplexed Address/Data Bus

USING THE COMBINED CHIP-ENABLE SIGNALS

The DS80C400 incorporates a feature allowing $\overline{\text{PCEx}}$ and $\overline{\text{CEx}}$ signals to be combined to provide a merged external program/data memory area. Setting the one or more $\overline{\text{PDCE7}}$–$\overline{\text{PDCE0}}$ bits (MCON1.3–0 and MCON.3–0) causes the corresponding chip-enable signal to be asserted for both MOVC and MOVX operations. Write access to combined program and data memory blocks is controlled by the $\overline{\text{WR}}$ signal, and read access is controlled by the $\overline{\text{PSEN}}$ signal. This feature is especially useful if the design achieves in-system reprogrammability through external flash memory, where a single device can be accessed through both MOVC instructions (program fetch) and MOVX write operations (updates to code memory). The following figures illustrate some examples of merged program/data memory configurations.

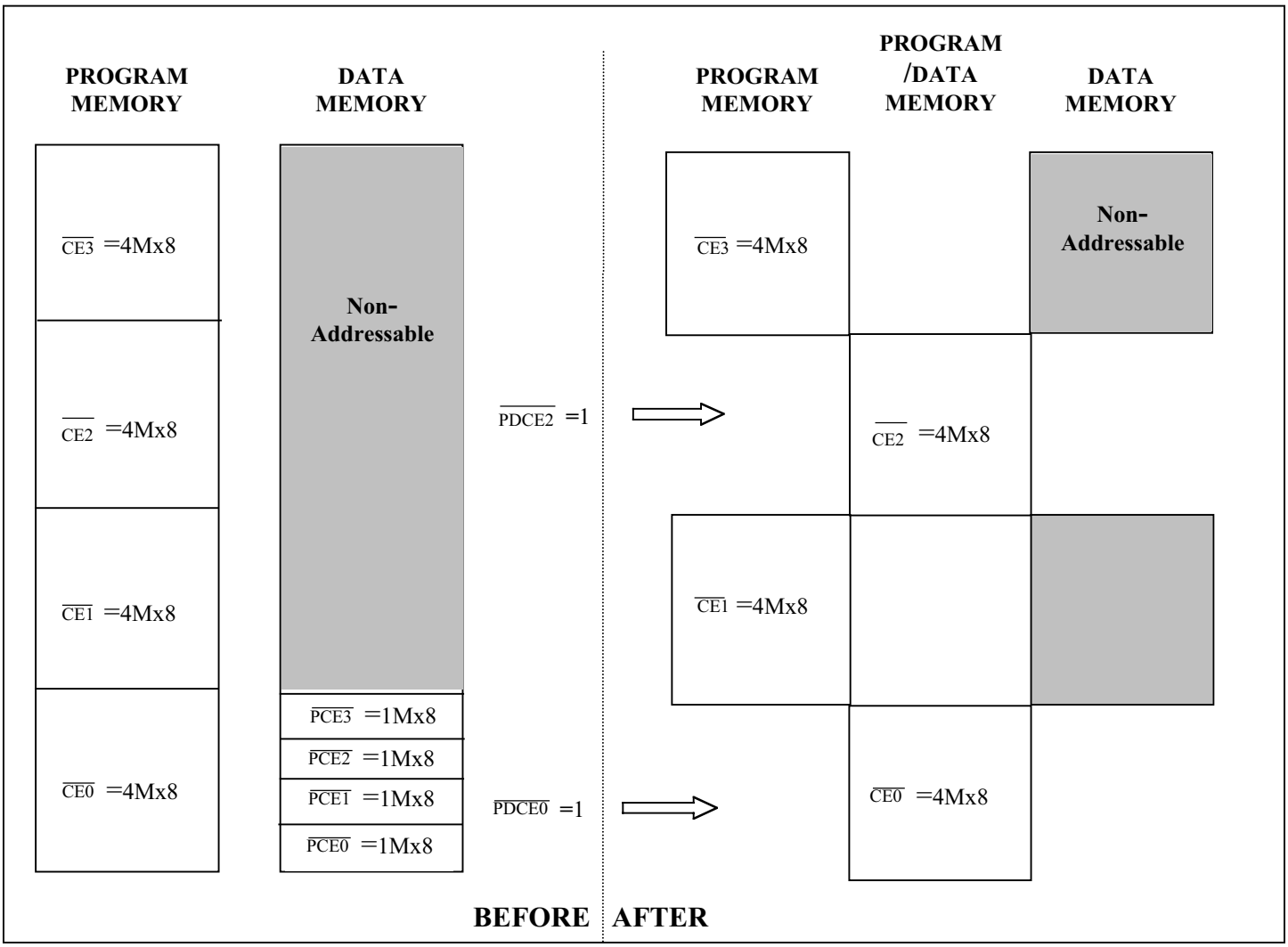


Figure 6-5. Merged Program/Data Access Under $\overline{CE0}$, $\overline{CE2}$, $\overline{PCE0}$–$\overline{PCE3}$ Becomes Inaccessible

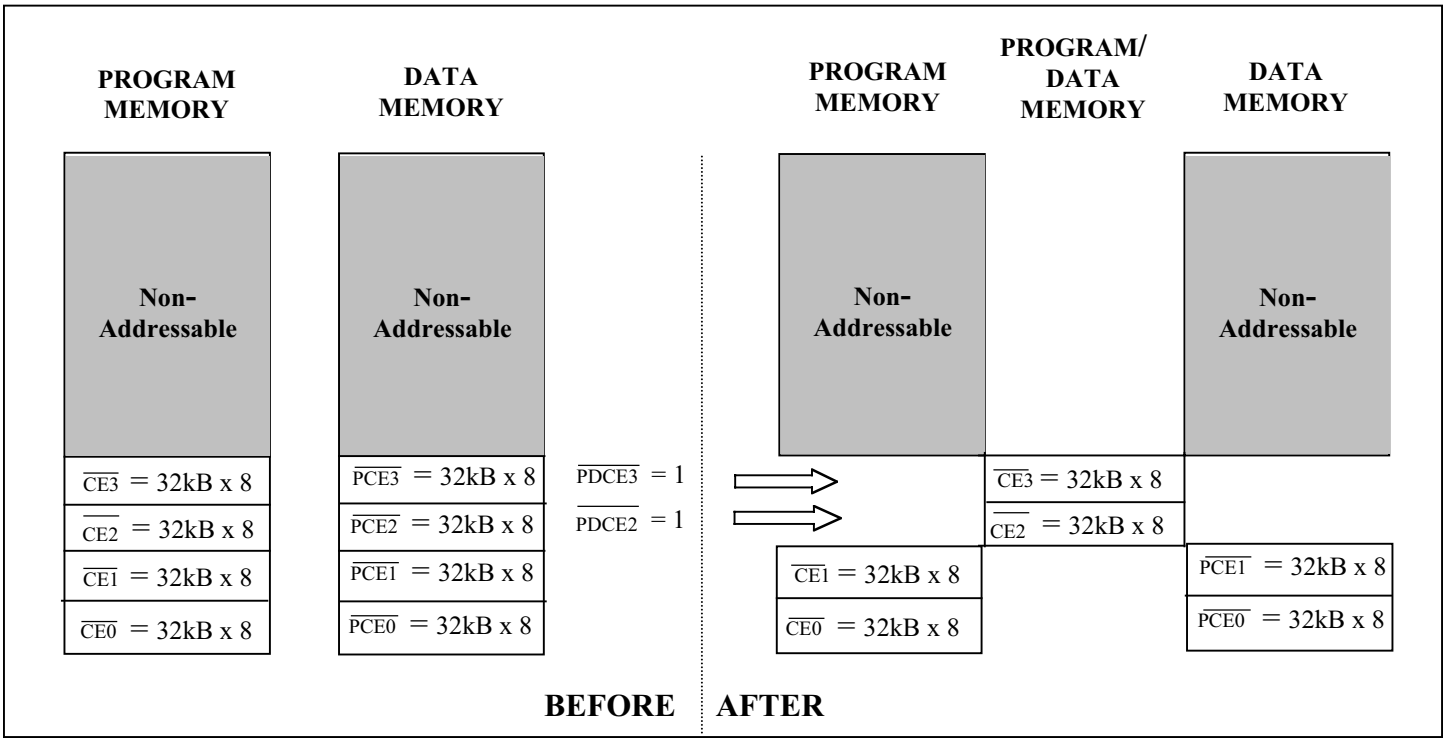


Figure 6-6. Merged Program/Data Access Under CE2, CE3, PCE2, and PCE3 Becomes Inaccessible

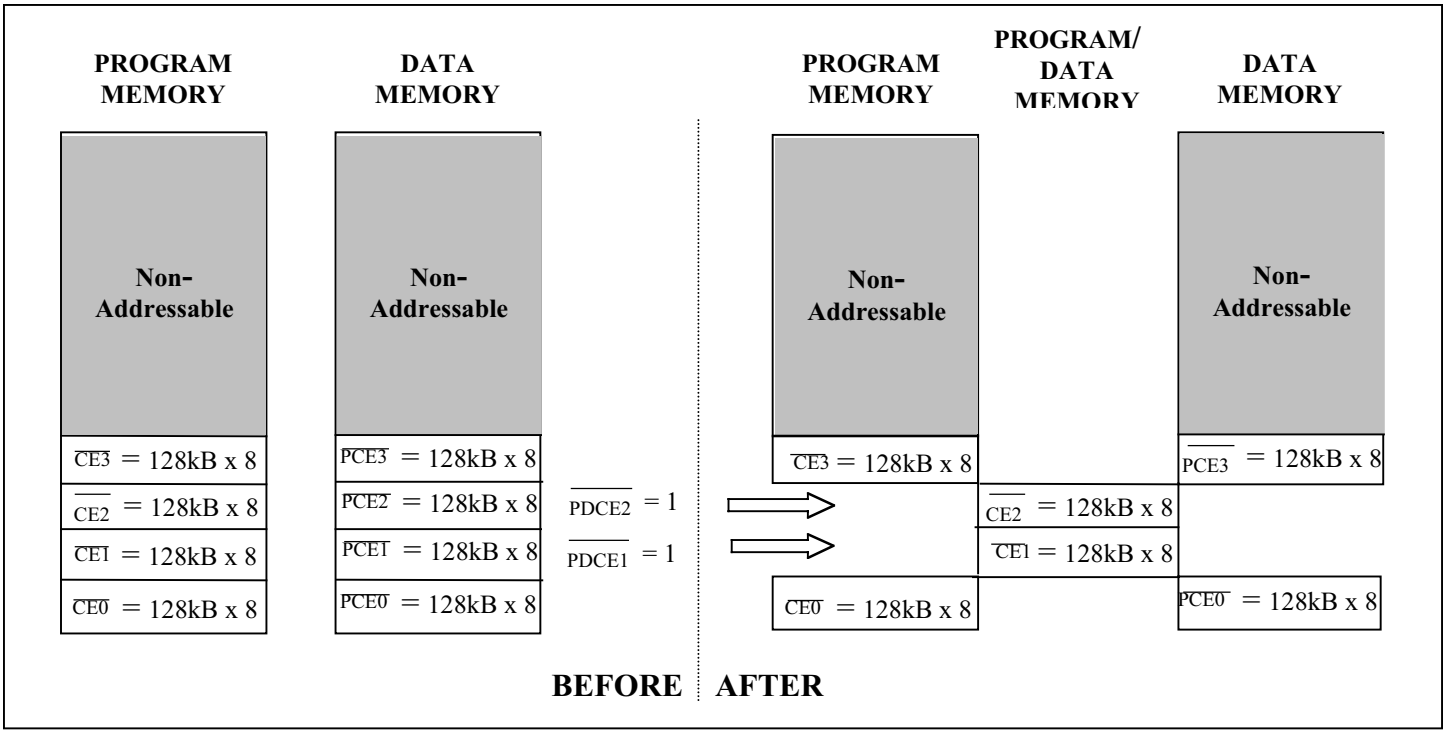


Figure 6-7. Merged Program/Data Access Under CE1, CE2, PCE1, and PCE2 Becomes Inaccessible

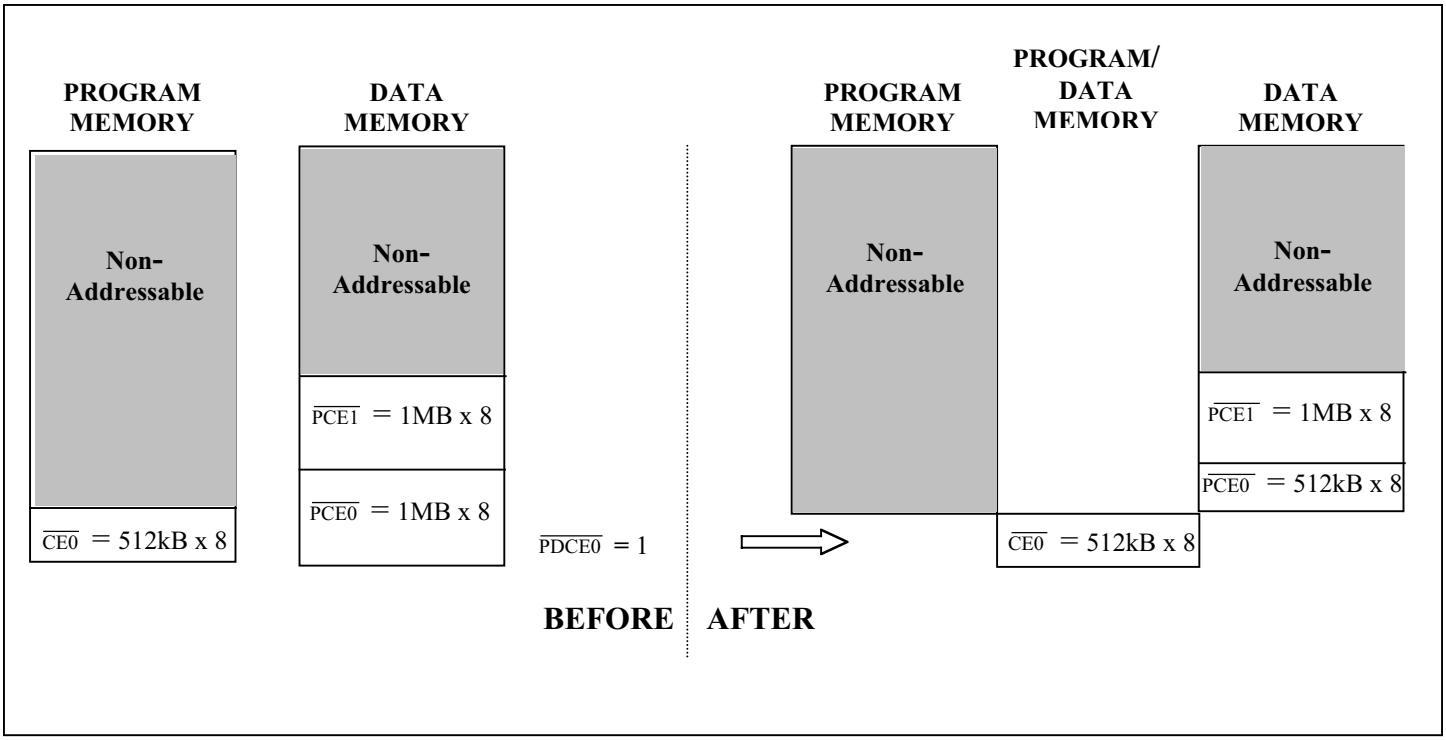


Figure 6-8. Merged Program/Data Access Under $\overline{CE0}$ and $\overline{PCE0}$ Becomes Partially Inaccessible

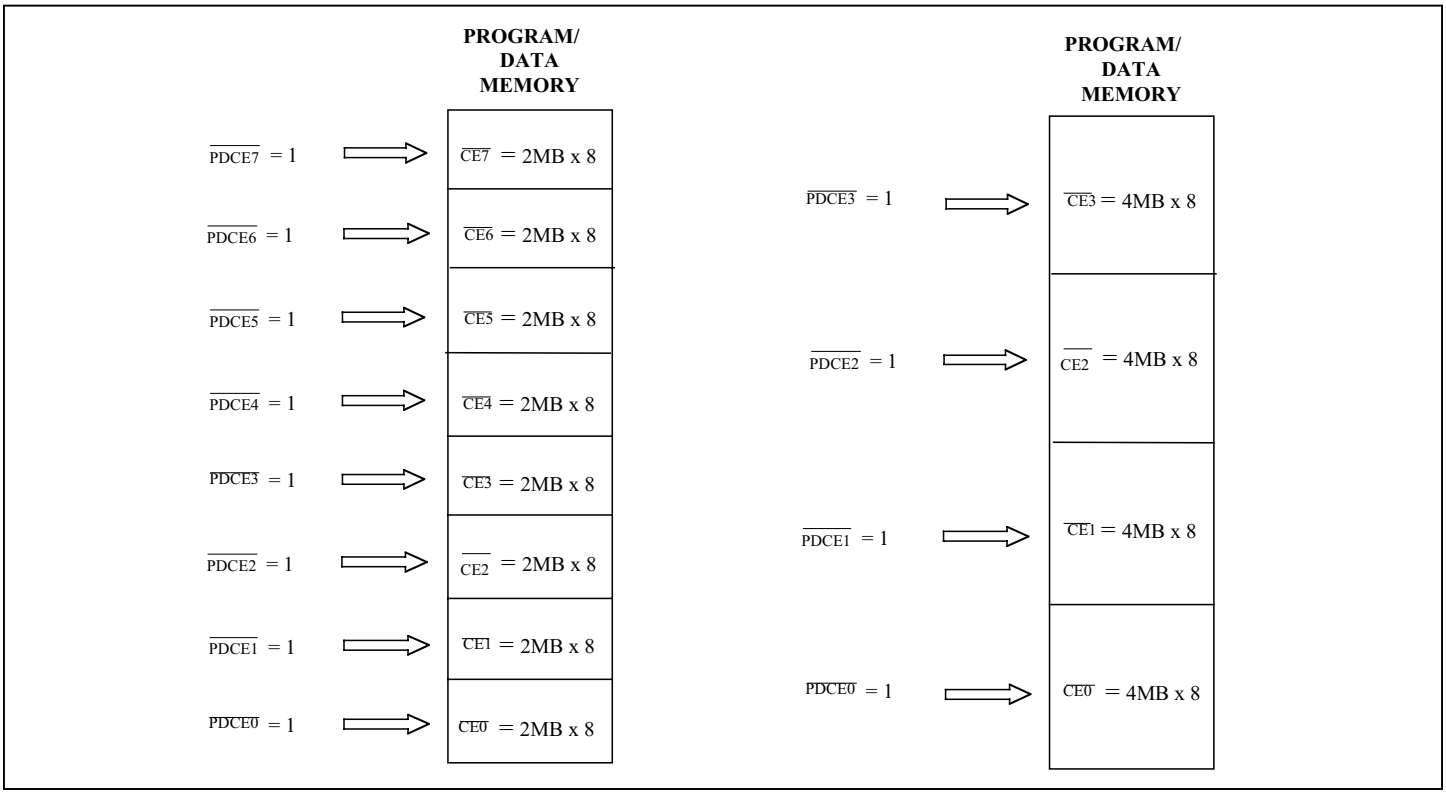


Figure 6-9. Full 16MB Merged Program/Data Memory Map Options

WRITE-PROTECTION FEATURE

When combined program/data memory access is enabled, there is the potential to inadvertently modify code that one meant to leave fixed. For this reason, the DS80C400 provides the ability to write protect the first 0–16kB of memory accessible through each of the chip enables $\overline{CE3}$, $\overline{CE2}$, $\overline{CE1}$, and $\overline{CE0}$. The write-protection feature for each chip enable is invoked by setting the appropriate WPE3–0 (MCON2.3–0) bit. The protected range is defined by the WPR2-0 (MCON2.6-4) bit settings as shown in Table 6-7. Any MOVX instructions attempting to write to a protected area is disallowed and the write-protected interrupt flag (WPIF–MCON2.7) is set by hardware.

Table 6-7. Write-Protection Range

MCON2.6-4	RANGE PROTECTED
000	0–2kB
001	0–4kB
010	0–6kB
011	0–8kB
100	0–10kB
101	0–12kB
110	0–14kB
111	0–16kB

ENHANCED QUAD DATA POINTERS

The DS80C400 offers enhanced features for accelerating the access and movement of data. The DS80C400 contains four data pointers (DPTR0, DPTR1, DPTR2, and DPTR3) instead of the single data pointer offered on the original 8051. DPTR0 is located at the same address as the original 8051 data pointer, allowing the DS80C400 to execute standard 8051 code with no modifications. The registers making up the second, third, and fourth data pointers are located at SFR address locations not used in the original 8051. To access the extended 24-bit address range supported by the DS80C400, a third, high-order byte (DPXn) has been added to each pointer, so that each data pointer is now comprised of the SFR combination DPXn + DPHn + DPLn. Table 6-8 summarizes the SFRs that make up each data pointer.

Table 6-8. Data Pointer SFR Locations

DATA POINTER	DPX + DPH + DPL COMBINATION
DPTR0	DPX (93h) + DPH (83h) + DPL (82h)
DPTR1	DPX1 (95h) + DPH1 (85h) + DPL1 (84h)
DPTR2	DPX2 (EBh) + DPH2 (F3h) + DPL2 (F2h)
DPTR3	DPX3 (EDh) + DPH3 (F5h) + DPL3 (F4h)

Two bits, SEL1 and SEL, both located in the data pointer select (DPS: 86h) register, select which one of the four pointers is active. For the SEL1, SEL bits, the 00b state selects DPTR0, 01b selects DPTR1, 10b selects DPTR2, and 11b selects DPTR3. To allow for code compatibility with previous dual data pointer microcontrollers, the bits adjacent to SEL are not implemented so that the INC DPS instruction can still be used to quickly toggle between DPTR0 and DPTR1 or between DPTR2 and DPTR3. Each data pointer also has an associated increment/decrement control bit. This bit defines, for each data pointer, whether the INC DPTR instruction increments or decrements the pointer when it is selected. When the active data pointer ID (increment/decrement) control bit is clear (= 0), the INC DPTR instruction increments the pointer, whereas a decrement occurs if the active pointer's ID bit is set (= 1) when the INC DPTR instruction is performed. The increment/decrement control bits for DPTR0, DPTR1 and ID0, ID1 respectively, can be found in the DPS (86h) register, while controls for DPTR2, DPTR3 and ID2, ID3 are found in the DPS1 (F6h) register.

ID0 = DPS.6

ID1 = DPS.7

ID2 = DPS1.6

ID3 = DPS1.7

To expedite data transfer and copy routines, the DS80C400 features the ability to automatically advance the data pointer and/or automatically toggle to a different data pointer in response to execution of certain instructions. The autotoggle feature does not toggle

between all four data pointers, nor does it allow the user to select which data pointers to toggle between. When the toggle select bit (TSL: DPS.5) is set to 1, the SEL bit (DPS.0) is automatically toggled every time one of the following DPTR instructions is executed. Thus, depending upon the state of the SEL1 bit (DPS.3), the active data pointer toggles between the DPTR0, DPTR1 pair or the DPTR2, DPTR3 pair.

```
Auto-Toggle (if TSL=1)
INC DPTR
MOV DPTR, #data16
MOV DPTR, #data24
MOVC A, @A+DPTR
MOVX A, @DPTR
MOVX @DPTR, A
```

When the autoincrement/decrement bit (AID: DPS.4) is set to 1, the active data pointer is automatically incremented or decremented every time one of the following DPTR instructions is executed.

```
Autoincrement/Decrement (if AID=1)
MOVC A, @A+DPTR
MOVX A, @DPTR
MOVX @DPTR, A
```

When used in conjunction, the autotoggle and auto-increment/decrement features can produce very fast routines for copying or moving data. For example, the basic loop needed to copy data between two different MOVX data memory ranges can be simplified to the following code:

```
;R6:R7 are loop variables controlling copy length
LOOP:
MOVX A, @DPTR        ;read from source
MOVX @DPTR, A        ;write to destination
DJNZ   R7, LOOP
DJNZ   R6, LOOP
```

The advantage to having four data pointers is that the second pair of data pointers can be selected for use whenever the first pair is already in use. This allows the application to: 1) conserve stack space that would normally be needed to save the data pointer(s) prior to another task that needs the data pointer(s), and 2) respond more quickly to a task that requires use of the data pointer(s).

For example, suppose that a pair of data pointers are being used to load data into Ethernet transmit buffer memory. While loading transmit buffer memory, an Ethernet receive interrupt request occurs that needs servicing. Servicing the interrupt requires that a received packet be transferred from internal data buffer memory to external data memory. Normally, the dual data pointer values would be pushed onto the stack and restored following the interrupt service routine to resume the loading of transmit buffer memory. With an additional pair of data pointers (DPTR2 and DPTR3) with which to work, the application need not push the current data pointer values onto the stack; it can simply select the second pair.

ADDENDUM TO SECTION 7: POWER MANAGEMENT

The DS80C400 supports the general power-management features of the DS87C520. Exceptions are noted as follows.

PRECISION VOLTAGE MONITOR

The DS80C400 does incorporate a precision bandgap reference, but unlike previous high-speed microcontrollers, it requires dual power supplies. The core power supply (V_{CC1}) is a nominal 1.8V supply, and the I/O power supply (V_{CC3}) is a nominal 3.3V supply. Therefore, two sets of voltage thresholds (one for each supply) have been implemented. The respective power-fail and reset voltage thresholds for V_{CC1} are V_{PFW1} and V_{RST1}, while the V_{CC3} thresholds are V_{PFW3} and V_{RST3}. The minimum, typical, and maximum values for these thresholds are specified in the *DC Electrical Characteristics* section of the DS80C400 data sheet.

EARLY WARNING POWER-FAIL INTERRUPT

The PFI status bit is set if either $V_{CC1} < V_{PFW1}$ or $V_{CC3} < V_{PFW3}$. Two additional status bits, V1PF (STATUS1.3) and V3PF (STATUS1.2), have been implemented so that the application can assess whether the V_{CC1} or V_{CC3} supply caused the PFI bit to be set. Manually setting either V1PF or V3PF status bit causes the PFI bit to be set. Clearing the PFI bit automatically clears both status bits.

POWER-FAIL RESET

The DS80C400 automatically invokes a reset if either $V_{CC1} < V_{RST1}$ or $V_{CC3} < V_{RST3}$. The only exception is if the device is in stop mode and BGS = 0. For this exception, the bandgap voltage reference is disabled, leaving the microcontroller unable to detect if either supply falls below its power-fail reset threshold.

POWER-ON RESET

The DS80C400 power-on reset sequence does not begin until both supplies are above their respective reset thresholds (i.e., $V_{CC1} > V_{RST1}$ and $V_{CC3} > V_{RST3}$). Once this condition is met, the 65536 oscillator clock warmup period begins. If either supply falls below its reset threshold during this warmup period, the process restarts.

BANDGAP SELECT

Refer to the *DC Electrical Characteristics* section of the DS80C400 data sheet for stop mode current specifications when the bandgap circuitry is enabled (BGS = 1) and disabled (BGS = 0).

POWER-MANAGEMENT SUMMARY

In addition to the bits listed in the *High-Speed Microcontroller User's Guide*, the following bits have been added:

STATUS1.3 V1PF—V_{CC1} power-fail. This bit is set to indicate that V_{CC1} has fallen below V_{PFW1}. The bit is cleared automatically when the PFI (WDCON.4) bit is cleared. Writing a 1 to this bit causes the PFI bit to be set.

STATUS1.2 V3PF—V_{CC3} power-fail. This bit is set to indicate that V_{CC3} has fallen below V_{PFW3}. The bit is cleared automatically when the PFI (WDCON.4) bit is cleared. Writing a 1 to this bit causes the PFI bit to be set.

POWER-MANAGEMENT MODES

A single power-management mode, referenced as PMM2 in the *High-Speed Microcontroller User's Guide*, is supported by the DS80C400. This PMM provides a machine cycle equal to the oscillator frequency divided by 1024. Power-management mode 1 (PMM1) is not supported on the DS80C400. A STATUS1 (F7h) register has been added, which contains transmit and receive activity indicators for serial port 2. These status indicators can be interrogated to prevent entry into PMM at an inopportune time.

STATUS1.1 SPTA2—Serial port 2 transmit activity. This bit indicates that data is currently being transmitted by serial port 2.

STATUS1.0 SPRA2—Serial port 2 receive activity. This bit indicates that data is currently being received by serial port 2.

PMM AND PERIPHERAL FUNCTIONS

Invoking PMM alters the system clock and those peripheral functions that derive their timing from the system clock. In addition to the peripherals mentioned in the *High-Speed Microcontroller User's Guide*, the following DS80C400 peripherals are affected by use of PMM:

- Timer 3
- Serial port 2
- CAN controller
- Ethernet controller
- 1-Wire bus master

The user is advised against using these peripheral functions while in PMM. The switchback feature, detailed in the *High-Speed Microcontroller User's Guide*, provides a means of quickly exiting PMM so that these peripherals can be operated at the default divide-by-4 system clock.

SWITCHBACK

In addition to the switchback sources listed in the *High-Speed Microcontroller User's Guide*, the following sources can also trigger a switchback:

- Serial start bit detected, serial port 2
- Transmit buffer loaded, serial port 2
- Ethernet activity, when the Ethernet controller is in sleep mode and the Ethernet power-management interrupt has been enabled
- CAN 0 bus activity (CAN0BA = 1), provided that the CAN controller is in one of the following states: CRST = 1, SWINT = 1, or PDE = 1

STOP MODE

Stop mode can only be invoked if the CAN processor has been disabled (by either the CRST or SWINT bits) and the Ethernet controller is in sleep mode. The stop (PCON.1) bit cannot be set until both of these conditions have been met.

PIN STATES IN IDLE OR STOP MODE

When either idle or stop modes are invoked, the pins exhibit the following states:

PIN	PIN STATE	DRIVE STRENGTH
ALE, PSEN	1	I_{OH3}
MII Outputs (TX_EN, TXD[3:0], MDC, MDIO)	No change (Note 1)	I_{OH3} or I_{OL2}
1-Wire Master I/O (OW, OWSTP)	No change (Note 1)	Open drain or I_{OL3}
Memory Interface Pins Port 0, Port 2, and Ports 3, 4, 5, 6, 7 (when used as any of the following: A21–A0, WR, RD, CE0–7, PCE0-3)	1	I_{OH1}
Port I/O pin	Port data (Note 2)	I_{OHI} or I_{OL1}

Note 1: Pin continues driving the same output state as was present when idle or stop modes were invoked. Since idle mode does not stop internal clocks, it does not necessarily force static states on alternate function pins with dedicated hardware driven by internal clocks (e.g., timer 2 clock output, 1-Wire master, Ethernet MII outputs, etc.) if that hardware has been configured to operate before Idle mode was invoked.
Note 2: Port reflects the data stored in the corresponding Port SFR.

Switching Between Clock Sources

The ring oscillator on the DS80C400 is similar to that on the DS80C320. As such, it does not support the run from ring feature, which allows the microcontroller to use the ring oscillator as a clock source after the external crystal has stabilized (CKRY = 1). The ring oscillator is used exclusively for resumption from stop mode when this feature has been enabled (RGSL = 1). The DS80C400 ring oscillator operates at approximately 15MHz.

ADDENDUM TO SECTION 8: RESET CONDITIONS

This section supersedes the corresponding section in the *High-Speed Microcontroller User's Guide.*

The microcontroller provides several ways to place the CPU in a reset state. It also offers the means for software to determine the cause of a reset. The reset state of most register bits is independent of the cause of the reset, but selected bits do depend on the reset source.

The reset sources, the reset state, and the function of the $\overline{\text{RSTOL}}$ pin are described in this section.

RESET SOURCES

The microcontroller has four ways of entering a reset state, described as follows:

- Power-on/power-fail reset
- Watchdog timer reset
- External reset
- Oscillator fail-detect reset

Power-On/Power-Fail Reset

The DS80C400 incorporates an internal voltage reference that holds the CPU in the power-on reset state if V_{CC1} is below V_{RST1} or V_{CC3} is below V_{RST3}. Once both supplies have risen above the respective thresholds (i.e., $V_{CC1} > V_{RST1}$ and $V_{CC3} > V_{RST3}$), the microcontroller then restarts the oscillation of the external crystal and counts 65,536 clock cycles. This helps the system maintain reliable operation by permitting operation only when both supply voltages are in a known good state.

The CPU exits the reset condition once the above conditions are met. This happens automatically, needing no external components or action. Execution begins at the standard reset vector address of 000000h. Software can determine when a power-on reset has occurred using the power-on reset flag (POR) located at WDCON.6. The POR flag is set only by a power-on reset and is unaffected by other reset sources. Since all resets cause a vector to location 000000h, software can read the POR flag to assess whether power failure was the reason for the reset. If the POR bit is intended to identify a power-on reset, software must clear the POR bit after reading it. This ability to distinguish a power-on reset from other reset sources allows different processing routines in accordance to the reset source.

When either power supply fails ($V_{CC1} < V_{RST1}$ or $V_{CC3} < V_{RST3}$), the power monitoring circuitry invokes the reset state again. This reset condition remains while the power is below the threshold. When the power supply rises above the reset threshold, a full power-on reset is performed. Thus, a brownout that causes one of the supplies to drop below the specified minimum threshold appears the same as a power-up.

Watchdog Timer Reset

The watchdog timer is a free-running timer with a programmable interval. The watchdog supervises CPU operation by requiring software to reset it before the timeout expires. If the timer is enabled, and software fails to clear it before this interval expires, a watchdog interrupt can be generated, if enabled. Furthermore, if the watchdog reset function has been enabled, the CPU can be placed into a reset state. The reset state is maintained for two machine cycles. Once the reset is removed, the software resumes execution at 000000h.

The watchdog timer is fully described in Section 11 of the *High-Speed Microcontroller User's Guide*. Software can determine that a watchdog timeout was the reason for the reset by using the watchdog timer reset flag (WTRF) located at WDCON.2. Hardware sets this bit to a logic 1 if the watchdog timer generates a reset. If a watchdog timer reset occurs, software should clear this flag manually. This allows software to detect the event if it occurs again.

External Reset

If the RST input is taken to a logic 1, the CPU is forced into a reset state. This does not occur instantaneously, as the condition must be detected and then clocked into the microcontroller. It requires between one and two machine cycles to detect and invoke the reset state. Thus, the reset is a synchronous operation and the internal CPU clock (derived from the external crystal oscillator or the ring oscillator) must be active to detect an external reset.

Once the reset state is invoked, it is maintained as long as RST = 1. When the RST is removed, the CPU exits the reset state within one to two machine cycles and begins execution at address 000000h. All registers default to their reset state. There is no flag to indicate that an external reset was applied. However, since the other three sources have associated flags, the RST pin is the default source when neither POR, WTRF, nor OFDF is set.

If an RST is applied while the processor is in the stop mode, the scenario changes slightly. As mentioned above, the reset is synchronous and requires a clock to be running. Since the stop mode stops all clocks, the RST initially causes the oscillator to begin running and forces the program counter to 000000h. Rather than the two-machine cycle delay described above, the processor applies the full power-on delay (65536 clocks) to allow the oscillator to stabilize.

Oscillator Fail-Detect Reset

Most members of the high-speed microcontroller family contain a watchdog timer. The intent of this timer is to force the processor into a known good state (reset) if it ever entered a runaway situation where it was not executing code properly. This is very powerful feature, but could be made stronger with a simple addition. Since the watchdog timer clock was derived from the main crystal oscillator, it was possible (though very unlikely) that the oscillator would fail (stop), leaving the processor in an undesirable state. Since the watchdog timer runs from the same clock, the timer would stop counting, which would prevent a watchdog timeout and the generation of a watchdog reset. This possibility is eliminated in the DS80C400 by the inclusion of an oscillator fail-detection circuit. When enabled, this circuit causes the processor to be reset if the oscillator frequency falls below 100kHz. This puts the processor into a known good state, regardless of the watchdog timer, if the main crystal oscillator should ever fail. Although the oscillator has failed when this reset occurs, the CPU is clocked into the normal reset state by other internal clocks.

The oscillator fail-detect feature is enabled by setting the OFDE (PCON.4) bit with software. This bit can be modified at any time. When

an oscillator fail detection occurs, the flag OFDF (PCON.5) bit is set by hardware when the processor enters reset. This bit must be cleared by software. The oscillator fail-detection circuit is not active during the crystal warmup period, and is not triggered when the oscillator is stopped upon entering stop mode.

RESET OUTPUTS

The microcontroller has one reset output, the $\overline{\text{RSTOL}}$ pin.

Reset Output Low ($\overline{\text{RSTOL}}$)

This external output pin is active low whenever the microcontroller is in a reset state. It can be used to signal to external devices that an otherwise invisible internal reset is in progress. It is active under the following conditions:

- When the processor has entered reset through the RST pin
- During the crystal warmup period following a power-on reset, or exit from stop mode (if RGMD = 0)
- During a watchdog timer reset ($\overline{\text{RSTOL}}$ active for two machine cycles)
- During an oscillator failure (if OFDE = 1)

RESET STATE

Regardless of the source of the reset, the state of the microcontroller is the same while in reset. When in reset, the oscillator is running, but no program execution is allowed. When the reset source is external, the user must remove the reset stimulus. Anytime power is applied to the device, completion of the power-on delay removes the reset state automatically.

Resets do not affect the scratchpad RAM. Thus, any data stored in RAM is preserved. The contents of internal MOVX data memory also remain unaffected by a reset. However, the minimum data retention voltage for these internal memories is not specified, so RAM data must be assumed lost whenever the POR bit has been set.

The reset state of SFR bits is described in Section 4. Bits that are marked SPECIAL have conditions that can affect their reset state. Consult the individual bit descriptions for more information. Note that the stack pointer is also reset. Thus, the stack is effectively lost during a reset, even though the RAM contents cannot be altered. Interrupts and timers are disabled. The watchdog timeout defaults to its shortest interval on any reset. I/O ports are taken to a weak high state (FFh). This leaves each port pin configured with the data latch set to a 1. Ports do not go to the 1 state instantly when a reset is applied, but are taken high within two machine cycles of asserting a reset (unless the reset was applied while the device was in stop mode). If the reset is applied while the device is in stop mode, ports 0 and 2 (and port 7, when $\overline{\text{MUX}}$ = 1) are taken high only after the 65,536 crystal oscillator warmup period has elapsed. When the reset stimulus is removed, program execution begins at address 000000h.

IN-SYSTEM DISABLE MODE

The in-system disable (ISD) feature allows the device to be three-stated for in-circuit emulation or board testing. During ISD mode, the device pins take on the following states:

DEVICE PIN	STATE DURING ISD
XTAL1, XTAL2	Oscillator remains active
$\overline{\text{RSTOL}}$	Driven per I_{OH3} specification
All other pins	True three-state

The following procedure is used to enter ISD mode at power-up:

1. Assert reset by pulling RST high.

2. Pull ALE low and pull $\overline{\text{PSEN}}$ high.

3. Verify that at least one of the following pins is high: P2.7, P2.6, and P2.5.

4. Release RST.

5. Device is now in ISD mode; release ALE and $\overline{\text{PSEN}}$, if desired.

Note: Pins P2.7, P2.6, and P2.5 should not be driven low when RST is released. This places the device into a reserved test mode. Because these pins have a weak pullup during reset, they can be left floating. The test mode is only sampled on the falling edge of RST and, once RST is released, its state does not affect device operation. In a similar manner, the ALE, $\overline{\text{PSEN}}$ and RST pins can be released, once their state does not affect device operation, and ISD mode is invoked. Power must be cycled to exit ISD mode.

ADDENDUM TO SECTION 9: INTERRUPTS

NAME	DESCRIPTION	VECTOR	NATURAL PRIORITY	FLAG BIT	ENABLE BIT	PRIORITY CONTROL BIT
PFI	Power-fail interrupt	33h	0	PFI(WDCON.4)	EPFI(WDCON.5)	N/A
INT0	External interrupt 0	03h	1	IE0(TCON.1)[2]	EX0(IE.0)	PX0(IP.0)
TF0	Timer 0	0Bh	2	TF0(TCON.5)[1]	ET0(IE.1)	PT0(IP.1)
INT1	External interrupt 1	13h	3	IE1(TCON.3)[2]	EX1(IE.2)	PX1(IP.2)
TF1	Timer 1	1Bh	4	TF1(TCON.7)[1]	ET1(IE.3)	PT1(IP.3)
TI0 or RI0	Serial port 0	23h	5	RI_0(SCON0.0), TI_0(SCON0.1)	ES0(IE.4)	PS0(IP.4)
TF2 or EXF2	Timer 2	2Bh	6	TF2(T2CON.7)	ET2(IE.5)	PT2(IP.7)
TI1 or RI1	Serial port 1	3Bh	7	RI_1(SCON1.0), TI_1(SCON1.1)	ES1(IE.6)	PS1(IP.6)
INT2 INT3 INT4 INT5/OWMI	External interrupts 2–5 1-Wire bus master interrupt	43h	8	IE2 (EXIF.4), IE3 (EXIF.5), IE4 (EXIF.6), IE5 (EXIF.7)[3]	EX2–5 (EIE.0) EOWMI[3]	PX2–5 (EIP.0)
TF3	Timer 3	4Bh	9	TF3 (T3CM.7)	ET3 (EIE.1)	PT3 (EIP.1)
TI2 or RI2	Serial port 2	53h	10	IE4 (EXIF.6)	ES2 (EIE.2)	PS2 (EIP.2)
WPI	Write-protect interrupt	5Bh	11	WPIF (MCON2.7)	EWPI (EIE.3)	PWPI (EIP.3)
C0I	CAN 0 interrupt	6Bh	12	Various	C0IE (EIE.6)	C0IP (EIP.6)
EAI	Ethernet activity	73h	13	TIF (BCUC.5), RIF (BCUC.4)	EAIE (EIE.5)	EAIP (EIP.5)
WDTI	Watchdog timer	63h	14	WDIF (WDCON.3)	EWDI (EIE.4)	PWDI (EIP.4)
EPMI	Ethernet power mode	7Bh	15	EPMF (BCUC.6)	EPMIE (EIE.7)	EPMIP (EIP.7)

Unless marked, all flags must be cleared by the application software.

[1]Cleared automatically by hardware when the service routine is entered.

[2]If edge triggered, the flag is cleared automatically by hardware when the service routine is entered. If level triggered, the flag follows the state of the interrupt pin.

[3]The global 1-Wire interrupt enable bit (EOWMI) and individual 1-Wire interrupt source enables are located in the internal 1-Wire registers and must be accessed by the OWMAD and OWMDR SFRs. Individual 1-Wire interrupt source flag bits located in the internal 1-Wire bus master interrupt flag register are accessed in the same way.

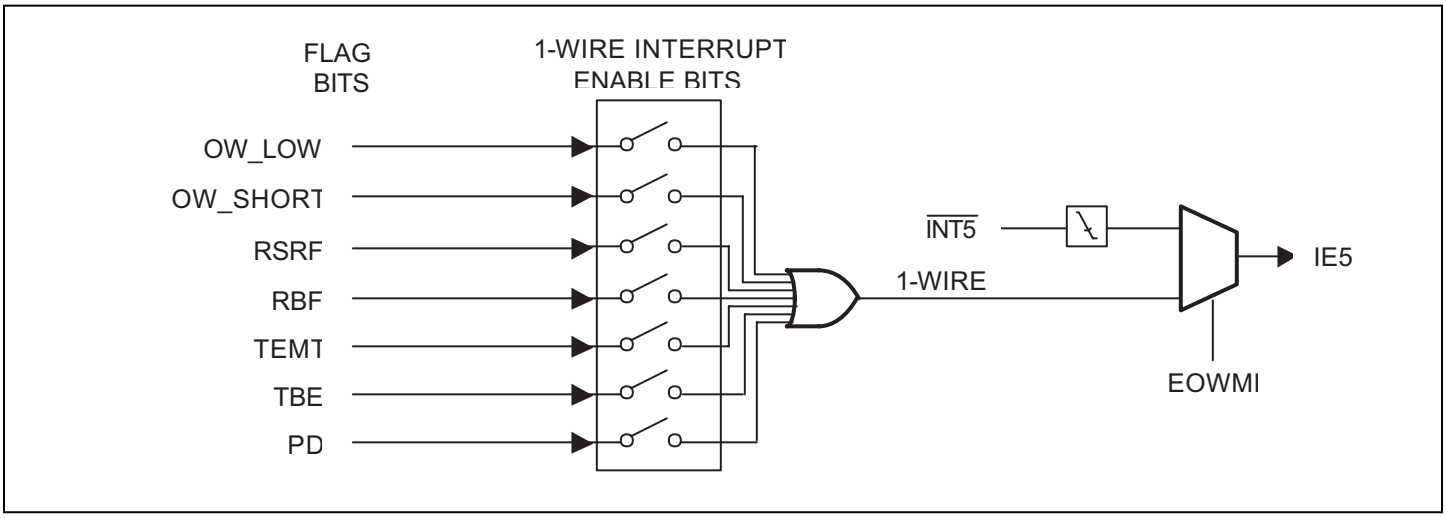


Figure 9-1. 1-Wire Interrupt Source

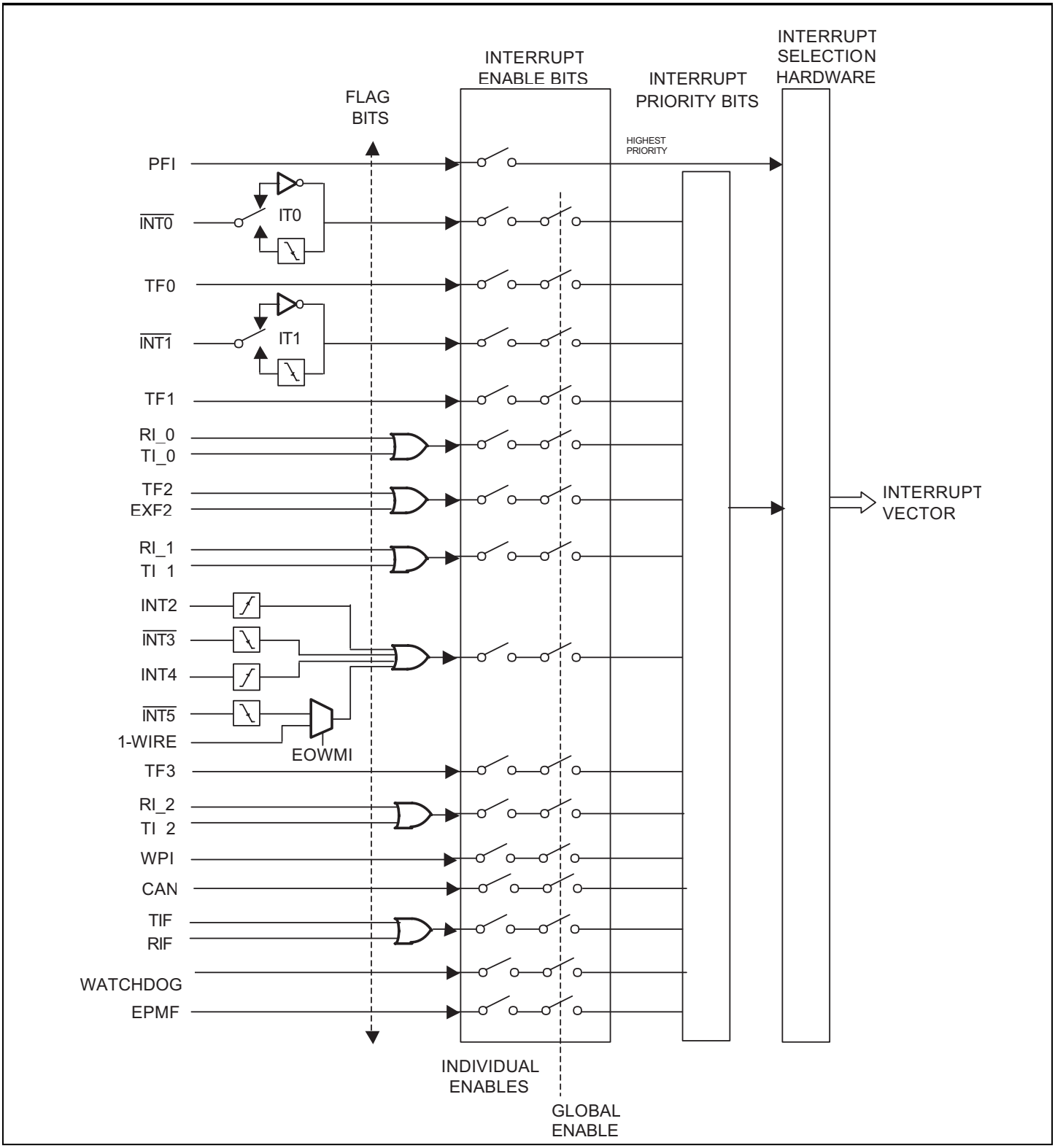


Figure 9-2. Interrupt Functional Diagram

ADDENDUM TO SECTION 10: PARALLEL I/O

Changes to this section primarily involve the additional functionality associated with ports 4–7.

Port 0

Ports 0 cannot be used for general-purpose I/O, hence no directly addressable SFR is provided for P0. The traditional P0 SFR address (80h) has been assigned to port 4.

Ports 4–7

Ports 4–7 are general-purpose I/O ports with optional special functions associated with each pin. Enabling the special function automatically converts the I/O pin to that function. To ensure proper operation, each alternate function pin should be programmed to a logic 1. The alternate functions for each port pin are listed as follows:

Port Pin	Alternate Function	Port Pin	Alternate Function
P4.7	A19–Address 19	P6.7	TXD2–Serial port 2 TXD
P4.6	A18–Address 18	P6.6	RXD2–Serial port 2 RXD
P4.5	A17–Address 17	P6.5	A21–Address 21
P4.4	A16–Address 16	P6.4	A20–Address 20
P4.3	CE3–Chip enable 3	P6.3	CE7–Chip enable 7
P4.2	CE2–Chip enable 2	P6.2	CE6–Chip enable 6
P4.1	CE1–Chip enable 1	P6.1	CE5–Chip enable 5
P4.0	CE0–Chip enable 0	P6.0	CE4–Chip enable 4
P5.7	PCE3–Peripheral chip enable 3	P7.7	A7–Address 7
P5.6	PCE2–Peripheral chip enable 2	P7.6	A6–Address 6
P5.5	PCE1–Peripheral chip enable 1	P7.5	A5–Address 5
P5.4	PCE0–Peripheral chip enable 0	P7.4	A4–Address 4
P5.3	—— None ——	P7.3	A3–Address 3
P5.2	T3 – Timer/counter 3 input	P7.2	A2–Address 2
P5.1	C0RX–CAN receive input	P7.1	A1–Address 1
P5.0	C0TX–CAN transmit output	P7.0	A0–Address 0

General-Purpose I/O

All of the above port pins (P4–P7) can serve as a general-purpose I/O. Data written to the port latch serves to set both level and direction of the data on the pin. Details on the function of port pins, when configured as general-purpose I/O, are provided in the *High-Speed Microcontroller User's Guide*. The DC electrical specifications for these pins, when used as I/O, can be found in the *DS80C400 data sheet*.

Alternate Functions A0–A7, A16–A21, $\overline{\text{CE0–7}}$, and $\overline{\text{PCE0–3}}$

When any P4, P5, P6, or P7 pin is configured in its alternate function, and that function concerns memory interfacing (A0–A7, A16–A21, $\overline{\text{PCE0–3}}$ or $\overline{\text{CE0–7}}$), the pin is driven using the stronger memory interface values (I_{OL2}, I_{OH3}), as shown in the *DC Electrical Characteristics* section of the *DS80C400 data sheet*.

Current-Limited Transitions

The DS80C400 does not employ the current-limited transition feature described in the *High-Speed Microcontroller User's Guide*.

5V-Tolerant I/O

In order for the DS80C400 to provide 5V-tolerant I/O, additional circuitry has been incorporated to detect I/O pad voltages that exceed

V_{CC3}. When these levels are detected, the circuitry enables protective switching to prevent undesirable voltages from reaching internal V_{CC3} logic. During the protective switching process, the pin sinks additional current, not to exceed 100µA, to V_{CC3}. I/O signals with long rise or fall times spend more time transitioning through the protective switching process, drawing more current, and should be avoided in 5V-tolerant mode. Steady state voltages on I/O between (V_{CC3} + 0.3V) and (V_{CC3} + 0.7V) draw excessive current in 5V-tolerant mode. When the switch completes, the external pad input is required to source ~2µA in order to sustain the internal switching circuit. Following the protective switch, the pad input cannot drive the internal logic nodes completely to V_{CC3}, therefore resulting in some additional static V_{CC3} current draw. The amount of additional current depends upon factors that include I/O pad voltage, V_{CC3} voltage, and temperature, but again should not exceed 100µA per pin and typically is in the ~1-10µA range. If the I/O are limited to the V_{CC3} supply range, these additional currents are not present. A simplified depiction of the 5V-tolerant I/O protection scheme is shown in Figure 10-1.

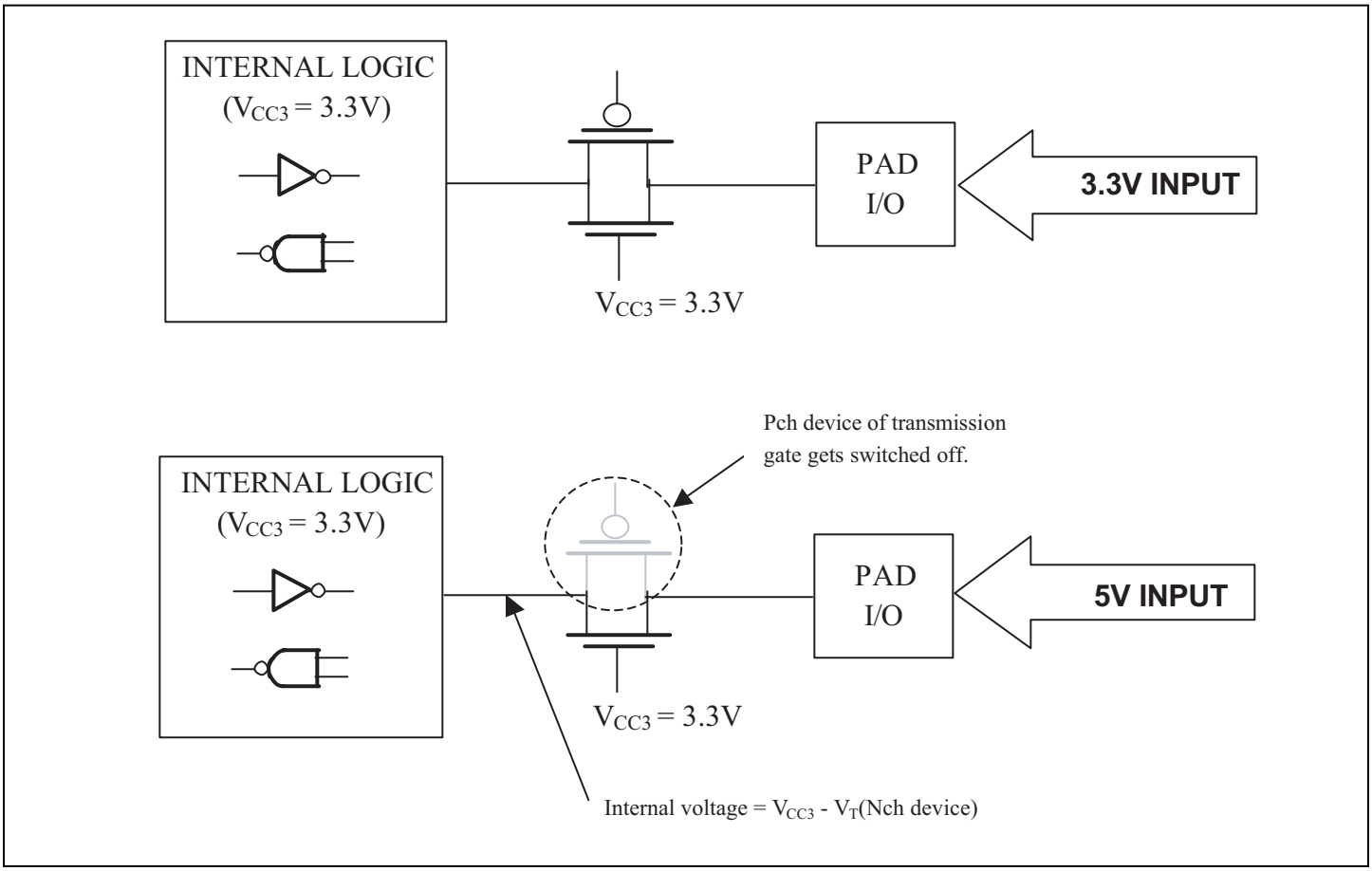


Figure 10-1. 5V-Tolerant I/O Pad

ADDENDUM TO SECTION 11: PROGRAMMABLE TIMERS

The timers of the DS80C400 are very similar to those described in the *High-Speed Microcontroller User's Guide*. The primary changes concern the removal of the PMM2 option, the inclusion of the clock multiplier settings (CD1:0, 4X/), and the addition of a fourth timer, timer 3. Timer 3 supports three modes of operation: 13-bit timer/counter, 16-bit timer/counter, and 8-bit timer/counter with autoreload. Timer 3 is accessible to serial port 2 for baud-rate generation in serial modes 1 and 3. The SFR bits controlling the function of timer 3 are summarized in Table 11-1 and Figures 11-1 through 11-8.

Table 11-1. Timer 3 SFR Bit Summary

BIT NAMES	DESCRIPTION	REGISTER LOCATION	BIT POSITIONS
GATE	Gate control enable for INT3 pin	T3CM–FDh	T3CM.3
C/T	Counter/timer select	T3CM–FDh	T3CM.2
M1, M0	Timer mode select bits	T3CM–FDh	T3CM.1
TF3	Timer overflow flag	T3CM–FDh	T3CM.7
TR3	Timer run control	T3CM–FDh	T3CM.6
T3M	Input clock select (/ 4)	T3CM–FDh	T3CM.5
	Timer LSB	TL0–FBh	
	Timer MSB	TH0–FCh	

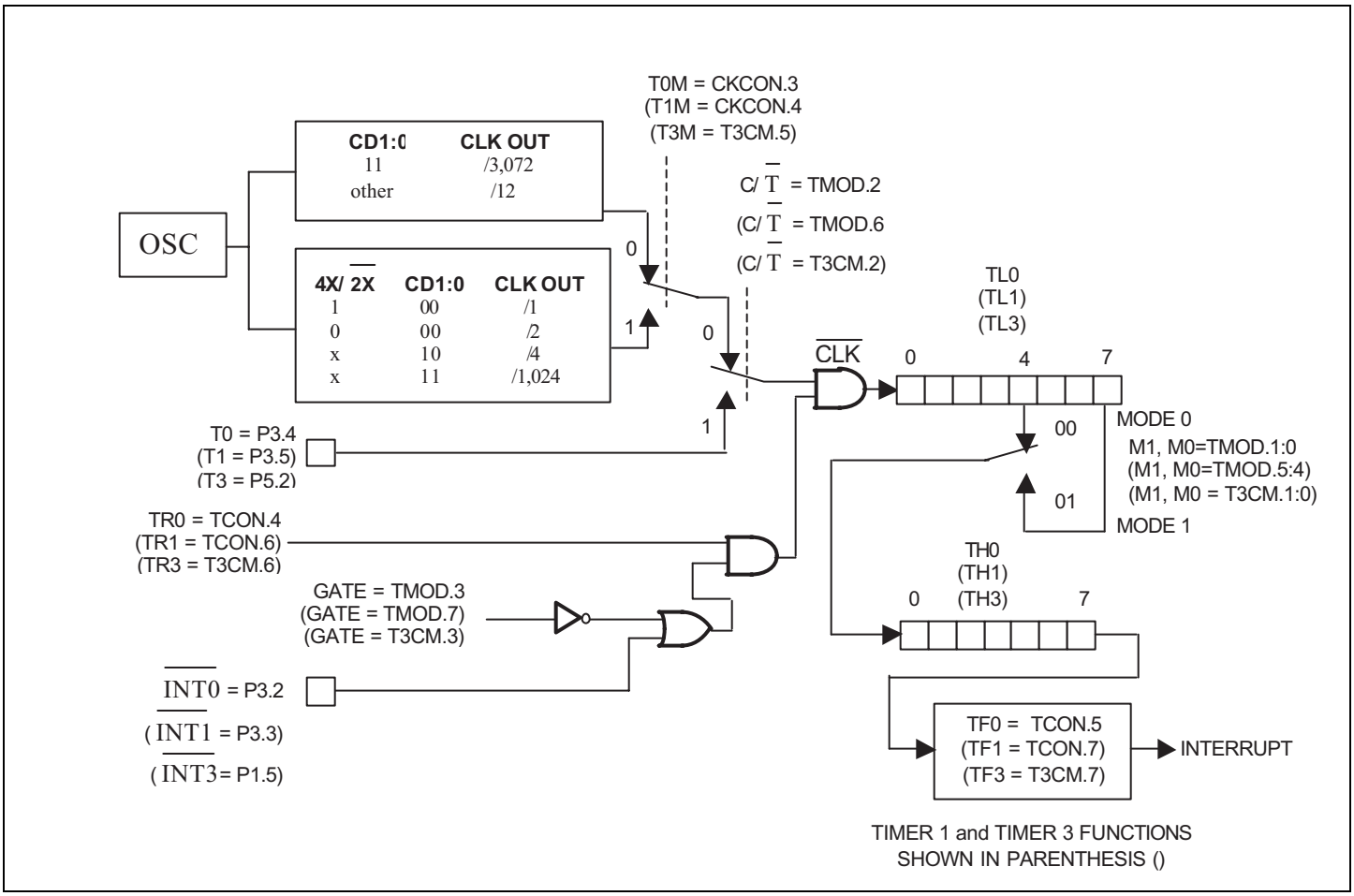


Figure 11-1. Timers/Counters 0, 1, and 3, Modes 0 and 1

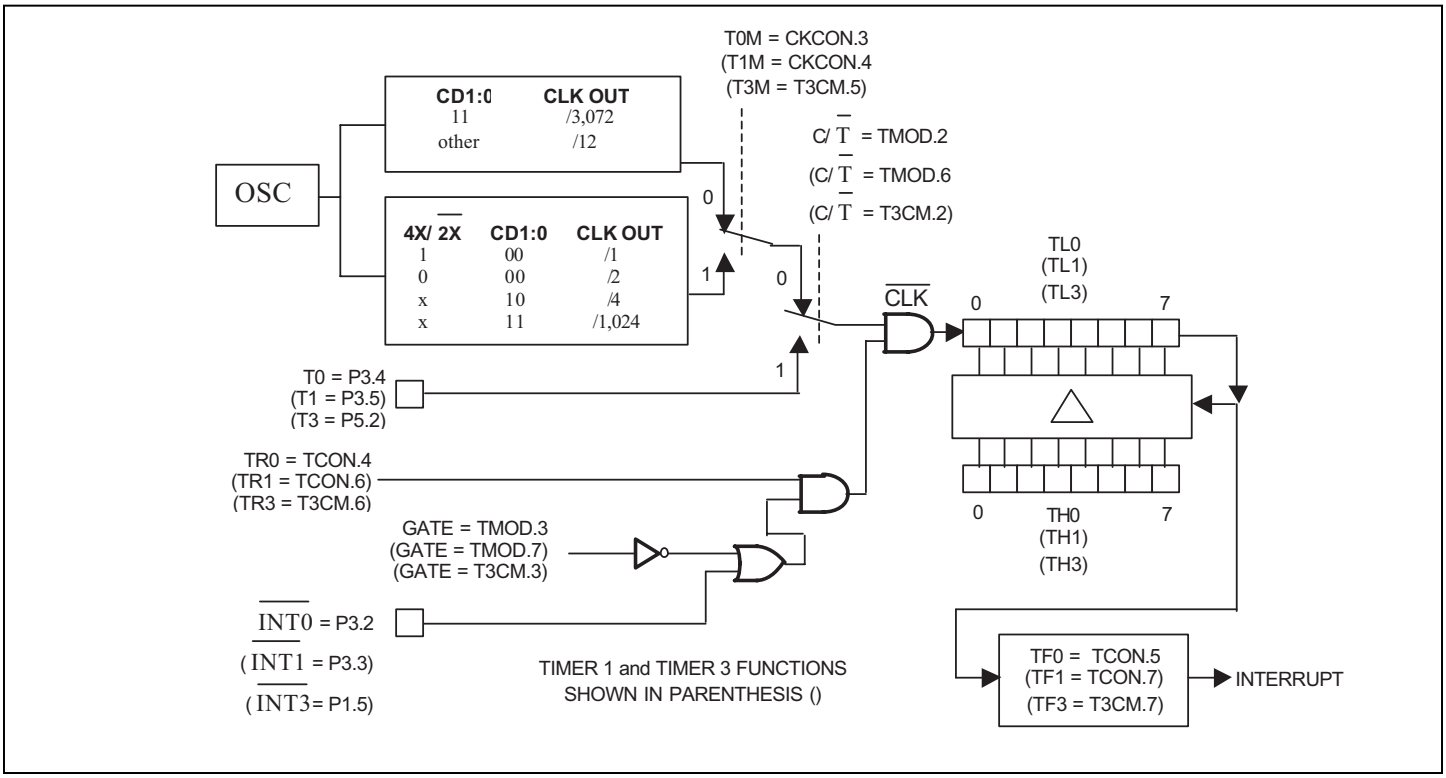


Figure 11-2. Timers/Counters 0, 1, and 3, Mode 2

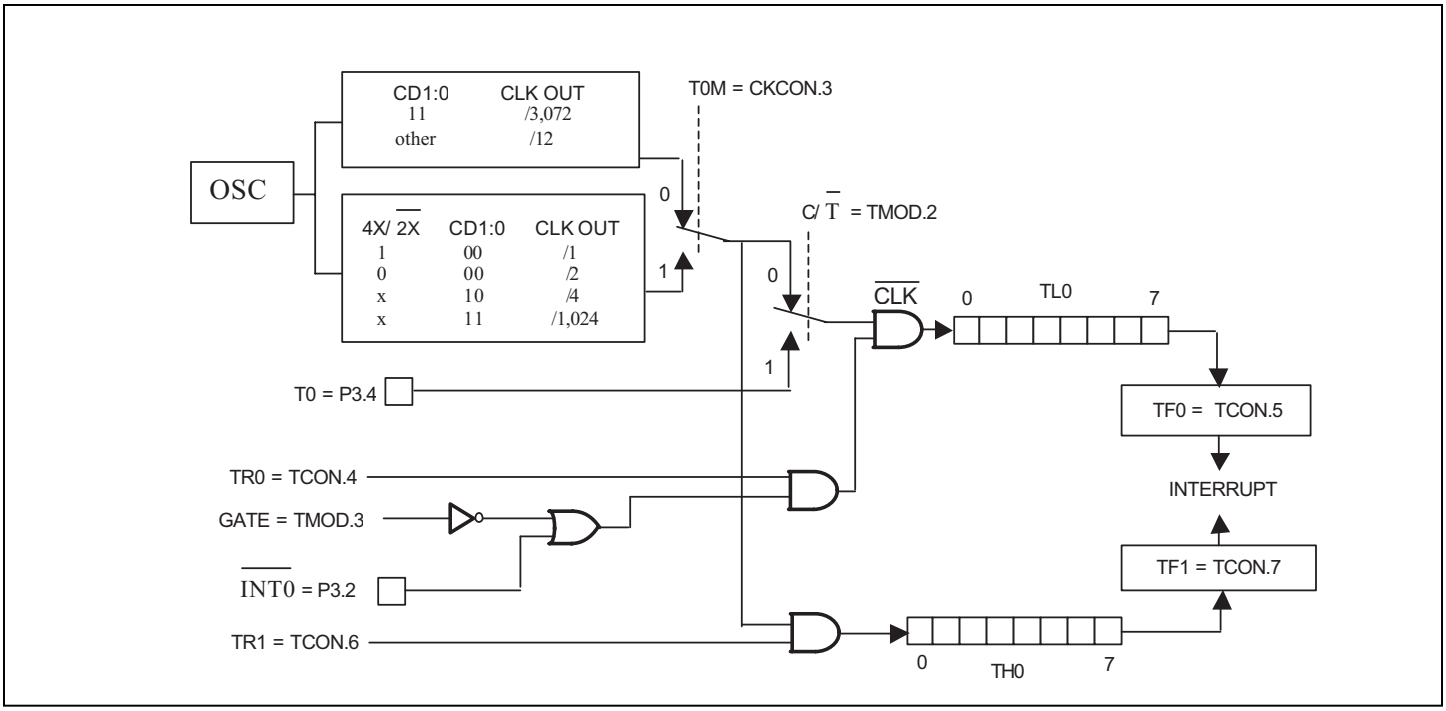
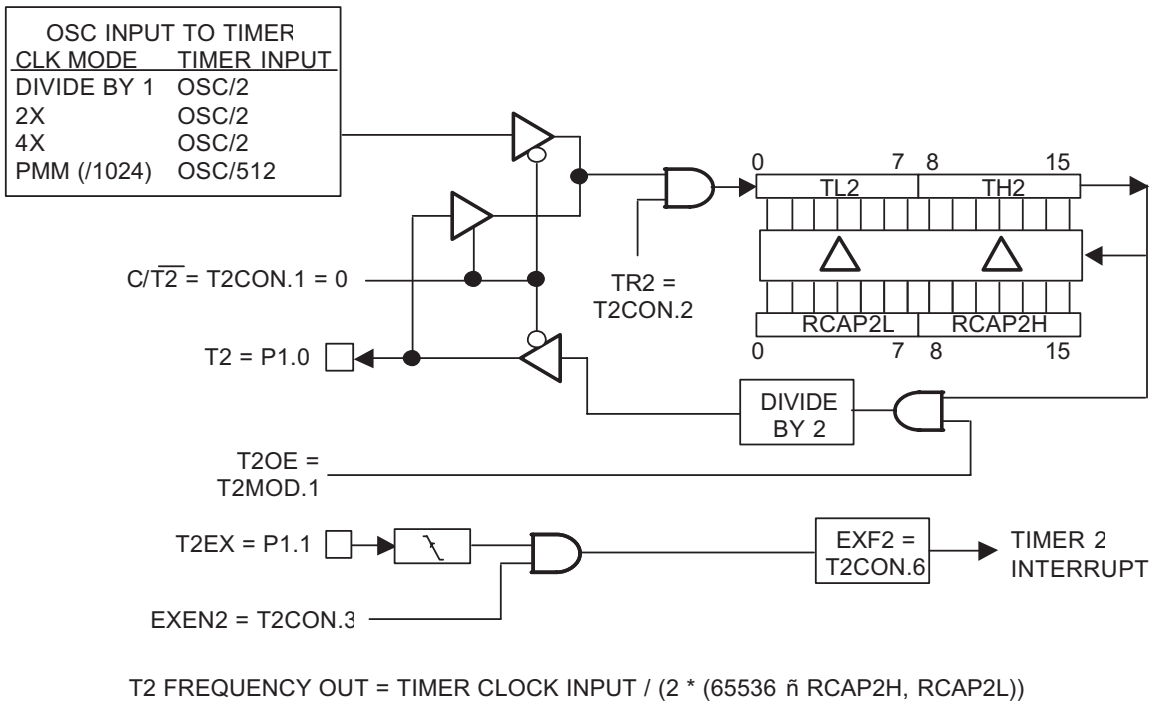
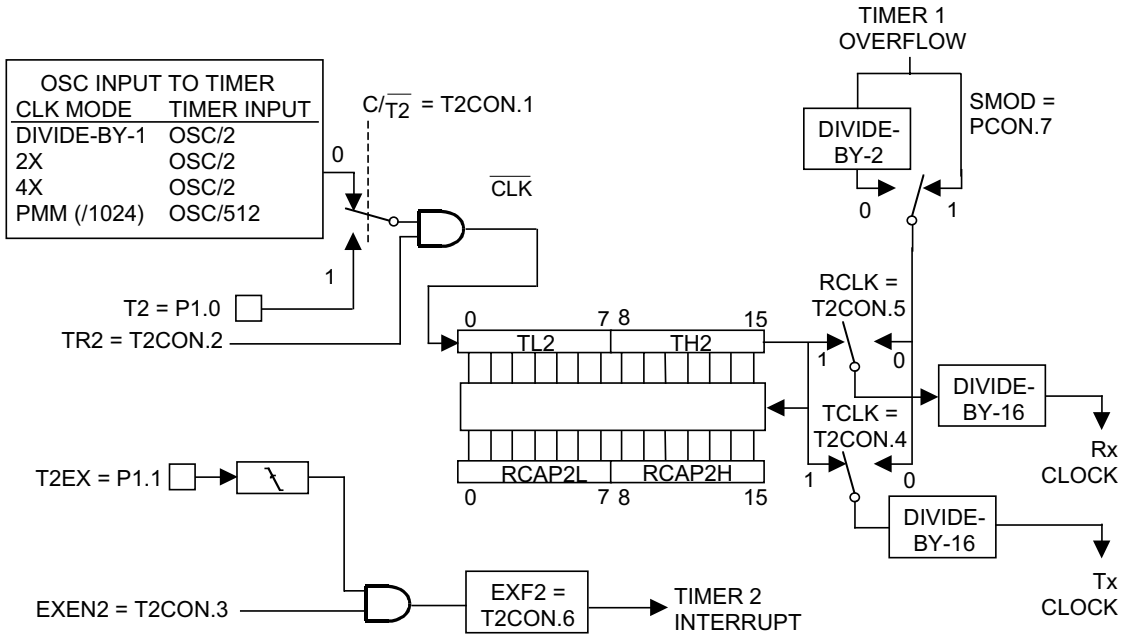


Figure 11-3. Timer/Counter 0, Mode 3



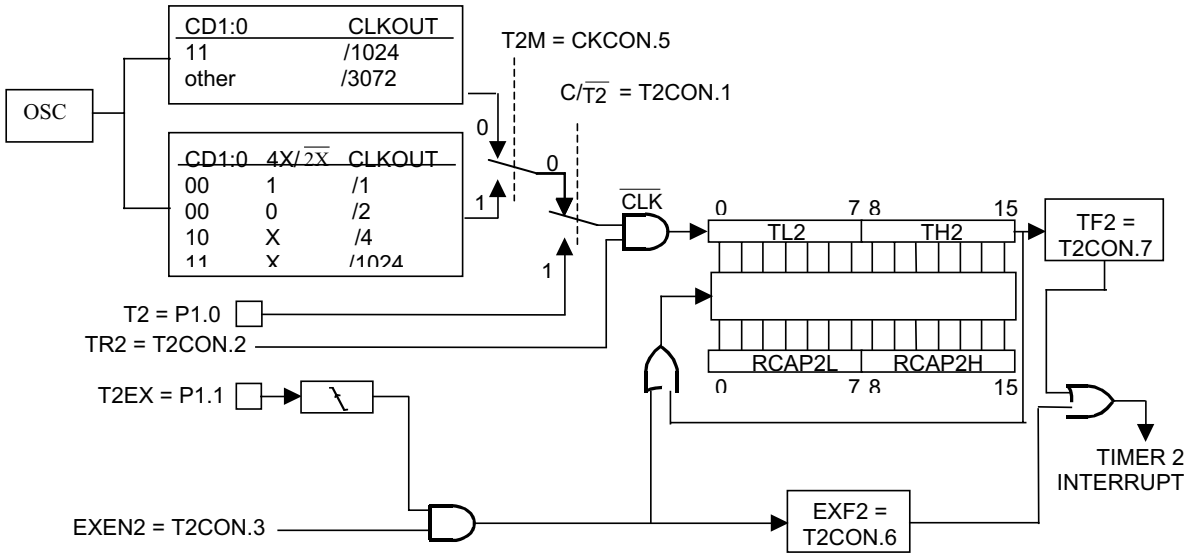
NOTE: *CPRL2 (T2CON.0) = 0.*

Figure 11-4. Timer/Counter 2 Clock-Out Mode



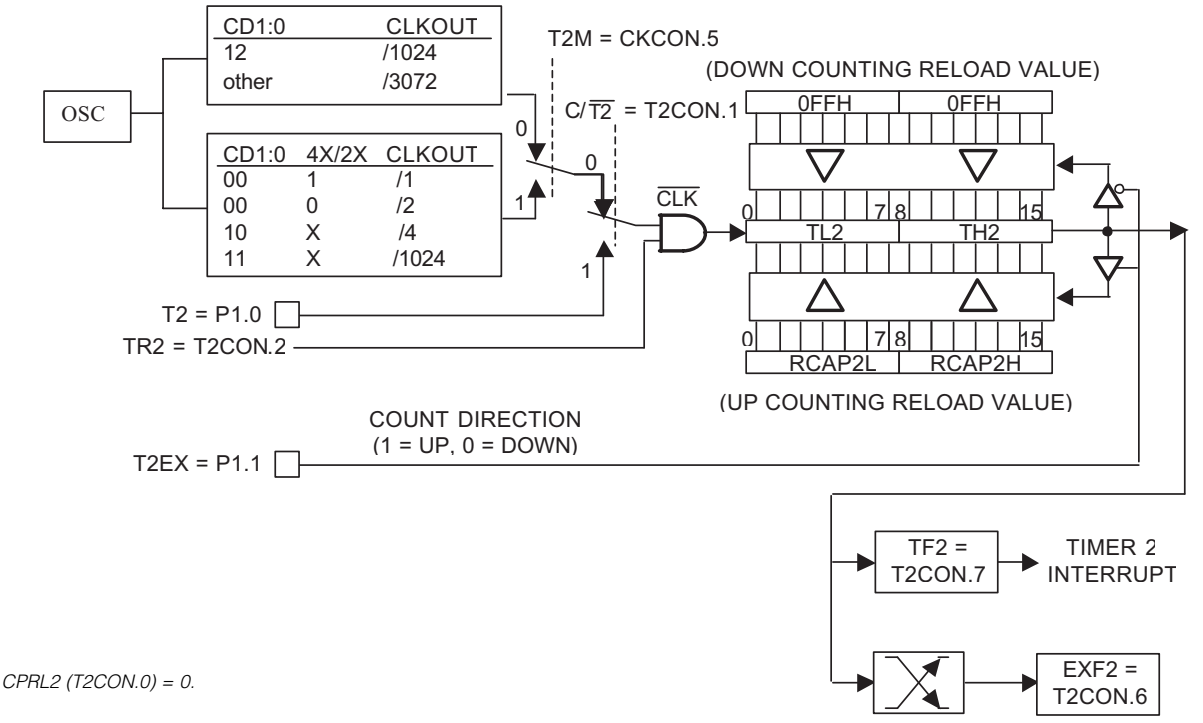
NOTE: *CPRL2 (T2CON.0) = 0, TCLK (T2CON.4) = 1, OR RCLK (T2CON.5) = 1.*

Figure 11-5. Timer/Counter 2 Baud-Rate Generator Mode



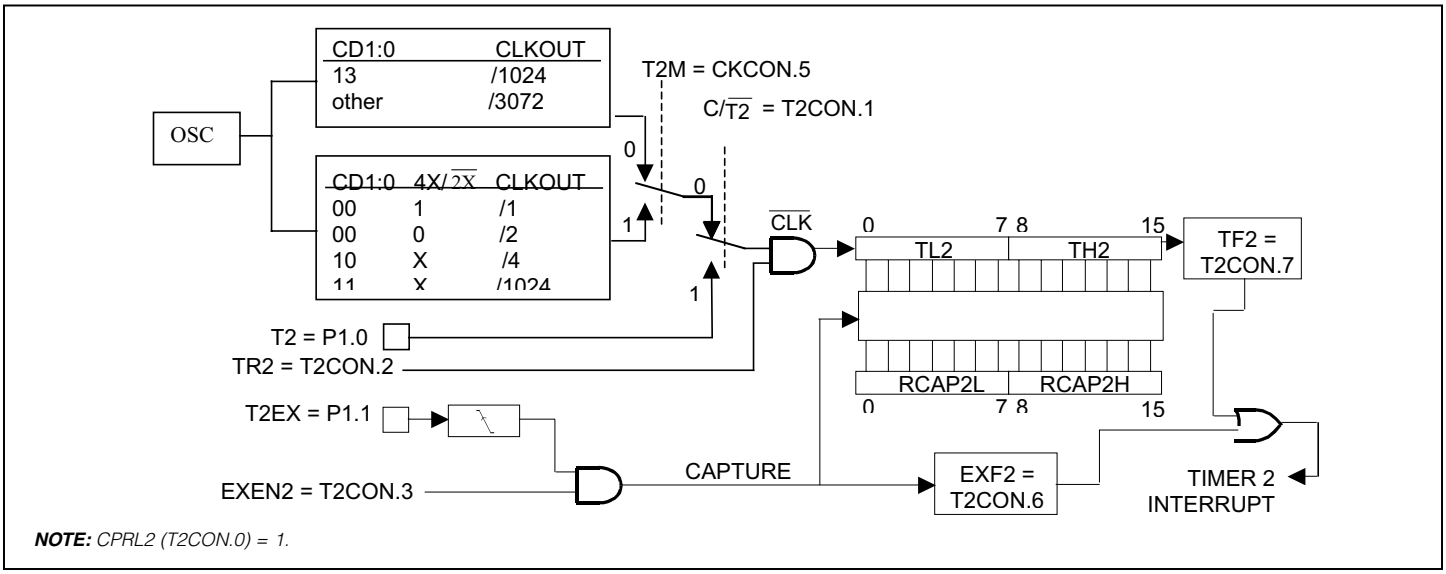
NOTE: *CPRL2 (T2CON.0) = 0.*

Figure 11-6. Tiimer/Counter 2 Autoreload Mode, DCEN = 0



NOTE: *CPRL2 (T2CON.0) = 0.*

Figure 11-7. Timer/Counter 2 Autoreload Mode, DCEN = 1



NOTE: *CPRL2 (T2CON.0) = 1.*

Figure 11-8. Timer/Counter 2 with Optional Capture

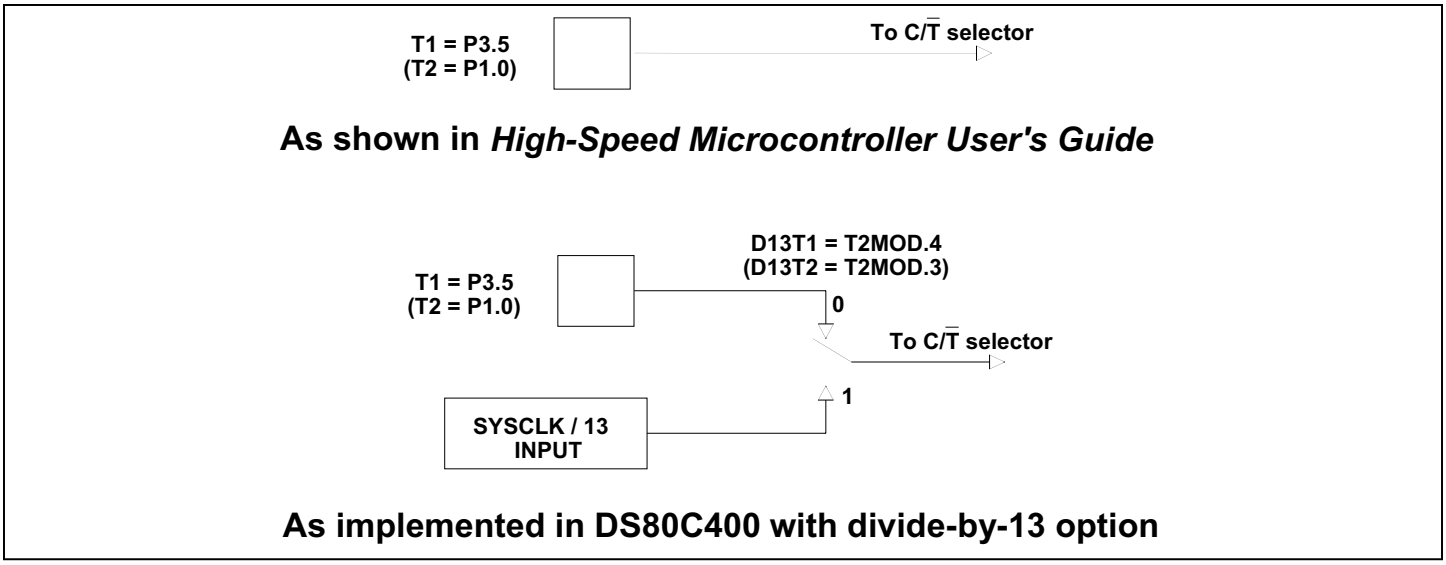


Figure 11-9. Operation of Divide-by-13 Bits

Divide-by-13 Option

Another change to the timers associated with the DS80C400 is the inclusion of a divide-by-13 option for timer 1 and timer 2. The option is independently enabled for each timer by setting the D13T1 (for timer 1) or D13T2 (for timer 2) bits. When enabled by setting the appropriate bits, the timer input from the T1 or T2 external pins is replaced by a time base equal to the system clock frequency divided by 13. Figure 11-9 illustrates the operation of these bits.

The setting of the divide-by-13 bits affects all operations of timer 1 and all operations of timer 2 except baud-rate generator mode. The baud-rate generator mode of timer 2 is not affected by the setting of the D13T2 bit.

The divide-by-13 settings of Timer 1 and Timer 2 allow the microprocessor to simultaneously generate standard serial baud rates and standard CAN baud rates within generally acceptable error tolerances. In an original divide-by-12 timer an 11.0592MHz crystal frequency (or multiple thereof) was usually needed to generate standard serial baud rates. The divide-by-13 setting offers the additional option of selecting the microprocessor clock frequency by 12/13. Thus, using a 12MHz crystal and the divide-by-13 setting, the effective baud rate becomes (12/13) x 12 = 11.0769MHz. This differs from the ideal baud rate frequency by only 0.16%, an insignificant dif-

ference, but allows the use of a crystal frequency that is acceptable for serial port and CAN operation. Using this feature, standard serial rates of up to 38.4k baud are available with a 16MHz crystal. A 24MHz crystal allows serial rates up to 115.2k baud.

Programmable Clock Output

When enabled, the DS80C400 can output a 50% duty cycle square wave on external pin P3.5. This signal is free running, and not synchronized to the external clock source. To enable this feature, three conditions must be met:

1.) Select the output frequency of the system clock divided by 2, 4, 6, or 8 through the clock output divide select bits (COR.2-1).

2.) The external clock output enable bit, CLKOE, must be set (COR.0). Steps 1 and 2 can be combined and must use the timed-access procedure.

3.) The P3.5 latch bit (P3.5) must be set.

IrDA Clock Output

The Infrared Data Association (IrDA) communication protocol is a popular way to connect physically separated devices up to a distance of 1m. The physical layer of the protocol is very easy to implement: configure the DS80C400's serial port 0 by selecting crystal speed, baud rate, etc. The microcontroller is then connected to an external IR encoder/decoder (ENDEC), which modulates the output of the serial port and communicates with an infrared transceiver.

The DS80C400 incorporates special circuitry that makes it simple to add IR capability to your design. Most IR encoders require the controlling microprocessor to supply a 16x clock to perform the modulation. The DS80C400 can provide this special 16x clock to the encoder without requiring the use of a timer. After the serial port is configured, set the IRDACK (COR.7) and CLKOE (COR.0) bits using the timed-access procedure. The P3.5 latch bit (P3.5) must also be set. At this point, a clock signal with a frequency of 16 times the serial port 0 baud rate is presented on P3.5.

Figure 11-10 illustrates how to add IrDA capability to a DS80C400-based system. The receive and transmit signals of serial port 0 are connected directly to an ENDEC chip that, in turn, interfaces to an infrared transceiver. The external clock output pin (P3.5) is connected to the 16XCLK pin of the ENDEC to provide the modulation clock.

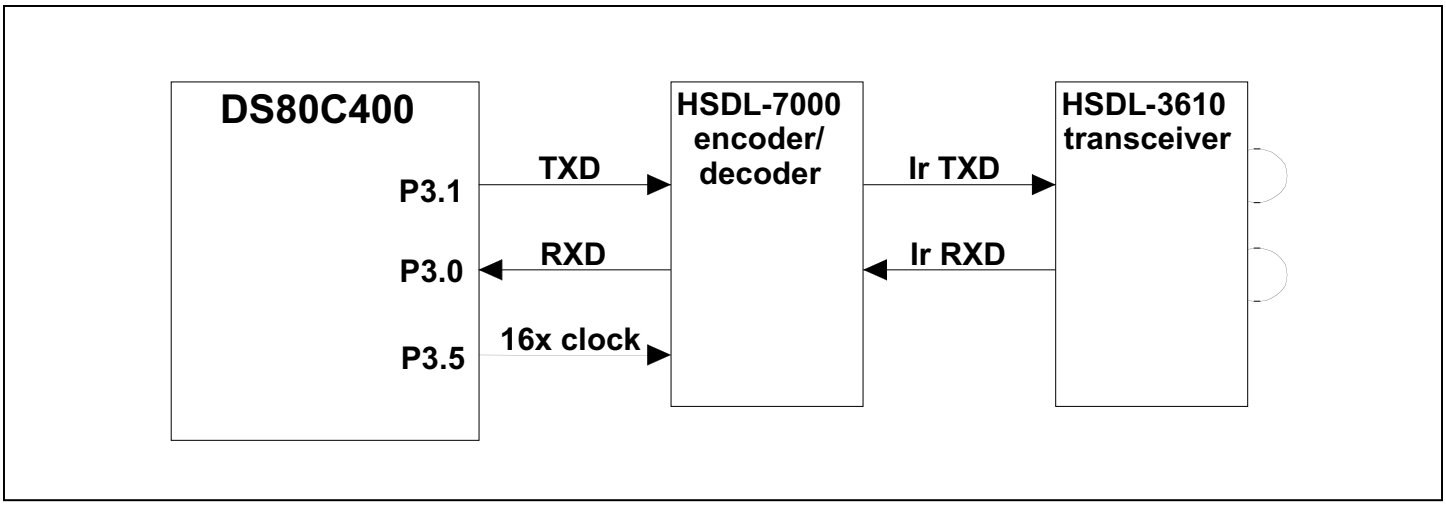


Figure 11-10. Sample IrDA Implementation

ADDENDUM TO SECTION 12: SERIAL I/O

The DS80C400 high-speed microcontroller provides a third fully independent UART (serial port 2) nearly identical to the second serial port (serial port 1). The transmit and receive pins associated with the third serial port are alternate functions for P6.7 (TXD2) and P6.6 (RXD2). The primary functional difference between the third serial port and the previous two is that it can only use timer 3 for baud clock generation in asynchronous serial modes 1 and 3.

The third UART has its own control register (SCON2: FEh) and transmit/receive buffer (SBUF2: FFh). These two registers, and others related to operation of the third serial port, are summarized in Table 12-1.

Table 12-1. Serial Port 2 Special Function Registers/Bits

BIT NAMES	DESCRIPTION	REGISTER LOCATION	BIT POSITIONS
SM0/FE_2	Serial mode select 0 or framing error	SCON2–FEh	SCON2.7
SM1_2	Serial mode select 1	SCON2–FEh	SCON2.6
SM2_2	Serial mode select 2	SCON2–FEh	SCON2.5
REN_2	Receive enable	SCON2–FEh	SCON2.4
TB8_2	9th transmit data bit	SCON2–FEh	SCON2.3
RB8_2	9th receive data bit	SCON2–FEh	SCON2.2
TI_2	Transmit interrupt flag	SCON2–FEh	SCON2.1
R1_2	Receive interrupt flag	SCON2–FEh	SCON2.0
SMOD_2	Baud-rate doubler bit	T3CM–FDh	T3CM.4
SPTA2	Serial port 2 transmit activity	STATUS1–F7h	STATUS1.1
SPRA2	Serial port 2 receive activity	STATUS1–F7h	STATUS1.0
SMOD0	Framing error-detection enable	PCON–87h	PCON.6
	Serial data buffer	SBUF2–FFh	
	Slave address mask enable	SADEN2–F1h	
	Slave address	SADDR2–D9h	

SERIAL MODE SUMMARY

All three serial ports provide four operating modes. These serial modes offer different communication protocols and baud rates. The four serial operating modes are shown in Table 12-2. Full details of each serial mode of operation are provided in the *High-Speed Microcontroller User's Guide*. The DS80C400 includes a clock multiplier that, when utilized, can affect the serial baud clock for certain modes. Provisions for use of the serial ports in conjunction with the clock multiplier and power-management mode are discussed later in this section.

Table 12-2. Serial I/O Modes

SERIAL MODE	SYNC/ ASYNC	BAUD CLOCK OPTIONS LISTED BY SERIAL PORT†			DATA BITS	START/STOP BITS	9TH BIT FUNCTION
		0	1	2			
0	Sync	$4t_{CLK}$ or $12t_{CLK}$			8	None	None
1	Async	Timer 1 or 2	Timer 1	Timer 3	8	1 start, 1 stop	None
2	Async	$32t_{CLK}$ or $64t_{CLK}$			9	1 start, 1 stop	0, 1, parity
3	Async	Timer 1 or 2	Timer 1	Timer 3	9	1 start, 1 stop	0, 1, parity

†*Use of the clock multiplier or power-management mode affects the baud clock.*

BAUD RATES

Each serial mode has baud-rate-generating hardware associated with it. This hardware must be properly configured by the user prior to operation of the serial port. The following baud-rate descriptions are separated by mode. Block diagrams have also been included to show any deviations from the diagrams included in the *High-Speed Microcontroller User's Guide*. Note that the baud clock input corresponding to the power-management mode has been omitted from each of the block diagrams. Reference Table 12-3 for power-management mode baud clock rates.

Mode 0

Mode 0 is synchronous, so the shift clock output frequency is the baud rate. Table 12-3 summarizes baud-rate generation as a function of the external oscillator frequency.

Table 12-3. Baud-Rate Generation, Mode 0

OSCILLATOR CYCLES PER MACHINE CYCLE	PMR REGISTER BITS 4X/2$\overline{X}$, CD1, CD0	MODE 0 SERIAL PORT CLOCK FREQUENCY	
		SM2 = 0 (default)	SM2 = 1
1 (4x mode)	100	OSC / 3	OSC / 1
2 (2x mode)	000	OSC / 6	OSC / 2
4 (default)	X01, X10	OSC / 12	OSC / 4
1024 (PMM)	X11	OSC / 3072	OSC / 1024

The default case is divide-by-12. The user can select the shift clock frequency using the SM2 bit in the associated SCON register. For serial port 0, the SM2_0 bit is SCON0.5. For serial port 1, the SM2_1 bit is SCON1.5. For serial port 2, the SM2_2 bit is SCON2.5.

When SM2 is set to a logic 0, the baud rate is a divide-by-12 of the system clock frequency, unless power-management mode is invoked. When operating in power-management mode with the SM2 bit clear (= 0), the serial port clock frequency is the oscillator frequency divided by 3072.

When SM2 is set to a logic 1, the baud rate is generated using the system clock frequency divided by 4, unless power-management mode is invoked. When power-management mode is used with the SM2 bit set (= 1), the serial port clock frequency tracks the system clock frequency. Note that this use of SM2 differs from a standard 80C32. In that device, SM2 had no valid use when the UART was in mode 0. Since the bit was generally set to a 0 and no clock multiplication was available on the 80C32, there should be no compatibility problems. The DS80C400 defaults to an oscillator divided by 12 serial port clock frequency.

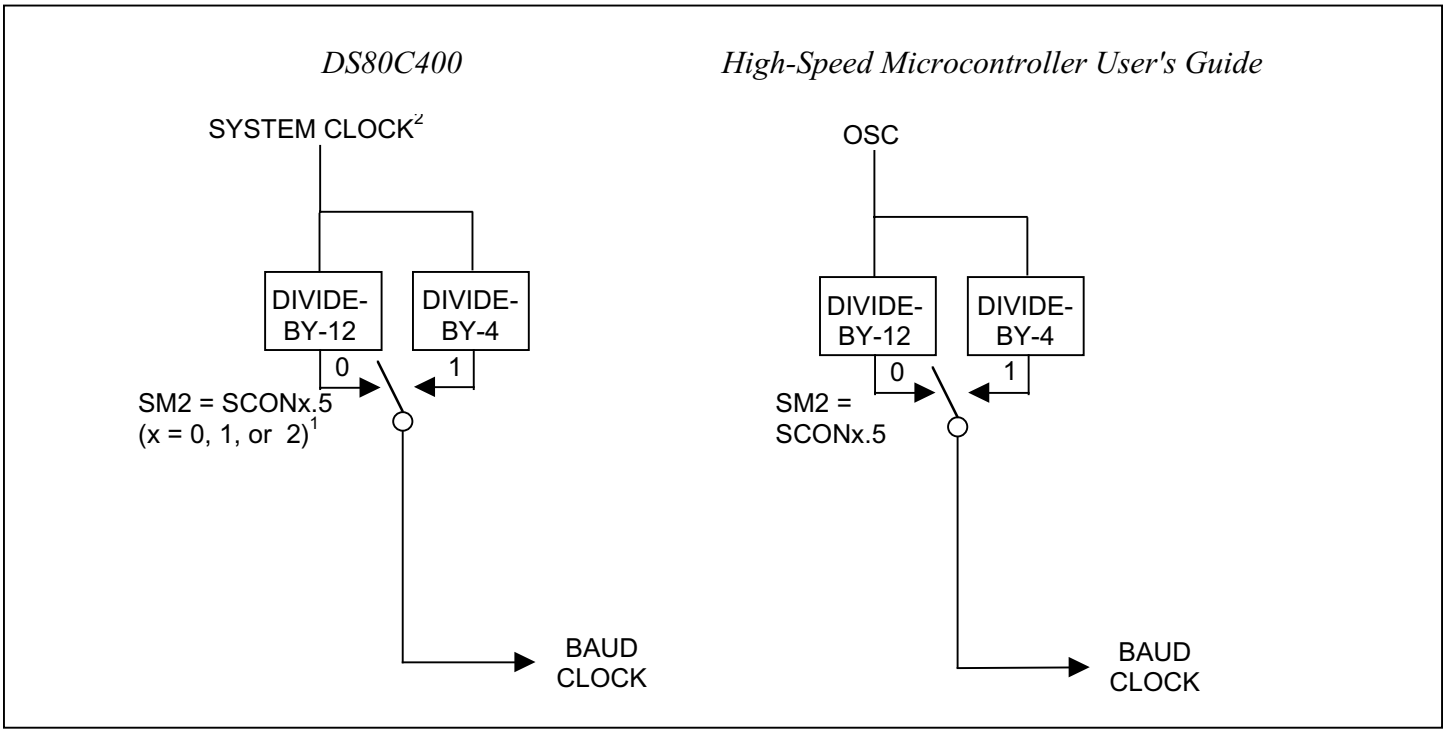


Figure 12-1. Serial Port Mode 0 Block Diagram Change

[1] *This change is due to the addition of the third serial port (serial port 2).*

[2] *This change is directly related to the inclusion of the clock multiplier.*

Mode 2

In this asynchronous mode, baud rates are always derived from the oscillator input. Table 12-4 summarizes baud-rate generation as a function of the external oscillator frequency. This mode works identically to the original 8051 family.

The default case is divide-by-64. The user can effectively double the serial port clock frequency by setting the SMOD bit to a logic 1 for the associated UART. For serial port 0, the SMOD_0 bit is PCON.7. This is the original location in the 8051 family. For serial port 1, the SMOD_1 bit is WDCON.7. For serial port 2, the SMOD_2 bit is T3CM.4. When operating in the power-management mode (CD1:0 = 11b), the serial port clock frequency is the oscillator frequency divided by 16384 when the SMOD bit is a logic 0, and twice that frequency (OSC / 8192) when the SMOD doubler bit is a logic 1. SMOD bits default to a logic 0 on all resets.

Table 12-4. Baud-Rate Generation, Mode 2

OSCILLATOR CYCLES PER MACHINE CYCLE	PMR REGISTER BITS 4X/2$\overline{X}$, CD1, CD0	MODE 2 SERIAL PORT CLOCK FREQUENCY	
		SMOD = 0	SMOD = 1
1 (4x mode)	100	OSC / 64	OSC / 32
2 (2x mode)	000	OSC / 64	OSC / 32
4 (default)	X01, X10	OSC / 64	OSC / 32
1024 (PMM)	X11	OSC / 16384	OSC / 8192

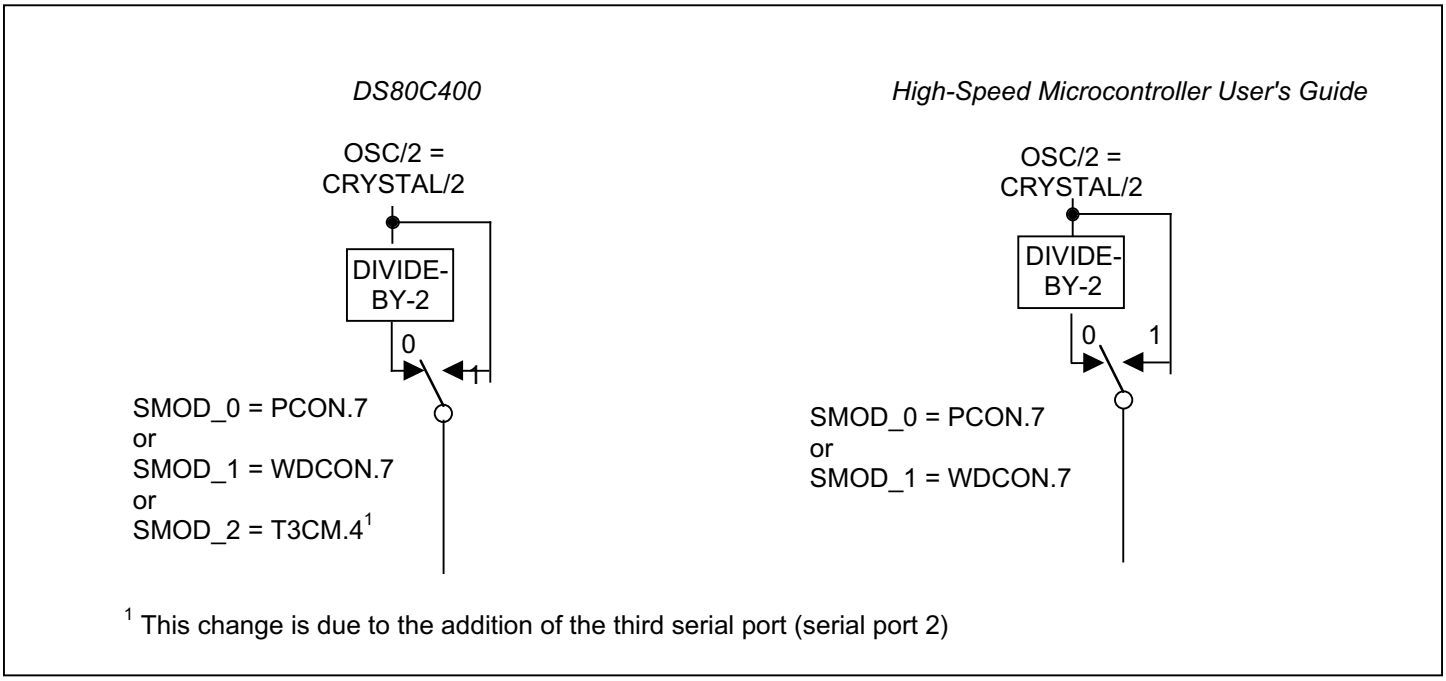


Figure 12-2. Serial Port Mode 2 Block Diagram Change

Mode 1 or 3

These asynchronous modes are commonly used for communication with PCs, modems, and other similar interfaces. The baud rates and bit timing are generated using timer 1, timer 2, or timer 3. The respective timer is placed in autoreload mode. When the timer reaches its rollover condition (timer 2: FFFFh, or timer 1, timer 3: FFh), a clock is sent to the baud-rate circuit. The baud-rate circuit generates the exact baud rate by further dividing the clock by 16 or 32 (depending upon the UART baud-rate doubler bit).

For serial port 0, either timer 1 or 2 can be used to generate baud rates. For serial port 1, only timer 1 can be used as the baud-rate generator. For serial port 2, only timer 3 can be used as the baud-rate generator.

Using Timer 1 or Timer 3 for Baud-Rate Generation

The following text and table describe the use of timer 1 for baud-rate generation. This information can also be used to describe the use of timer 3 for baud-rate generation by replacing every timer 1 reference with a corresponding timer 3 reference (timer 3 ≥ timer 1, TH3 ≥ TH1, T3M ≥ T1M).

To use timer 1 as the baud-rate generator, it is commonly put into the 8-bit autoreload mode. In this way, the CPU is not involved in baud-rate generation. Note that the timer interrupt should not be enabled. In the 8-bit autoreload mode (timer 1, mode 2), the reload value is stored in TH1. Thus, the combination of timer 1 input clock frequency and TH1 determines the baud rate.

The timer 1 input clock, relative to the external crystal clock, can be altered in two ways: 1) by changing the system clock, or 2) by changing the timer input clock divide ratio. Modifying the system clock is accomplished using the clock divide bits (CD1:0) found in the PMR SFR. This procedure is discussed in Section 5. The timer 1 input clock divide ratio is configurable using the T1M (CKCON.4) register bit. For the default T1M setting (= 0), a system clock frequency divided by 12 signals drives timer 1. Setting the T1M bit to a logic 1 provides a system clock divided by 4 input to timer 1. When using power-management mode, setting T1M to a logic 1 results in the system clock (OSC / 1024) being used as the input clock to timer 1. If T1M is clear (= 0) in power-management mode, the system clock divided by 3 (OSC / 3072) is provided to timer 1. Table 12-5 summarizes the relationship between the external crystal frequency and the timer 1 input clock for the various configurations.

Table 12-5. Relationship Between External Crystal Frequency and Timer 1

OSCILLATOR CYCLES PER MACHINE CYCLE	PMR REGISTER BITS 4X/2$\overline{X}$, CD1, CD0	MODE 1, 3 SERIAL PORT CLOCK FREQUENCY	
		T1M = 0	T1M = 1
1 (4x mode)	100	OSC / 12	OSC / 1
2 (2x mode)	000	OSC / 12	OSC / 2
4 (default)	X01, X10	OSC / 12	OSC / 4
1024 (PMM)	X11	OSC / 3072	OSC / 1024

Using timer 1 in the 8-bit autoreload mode, serial port baud rates for mode 1 or 3 can be calculated using the following formula:

$$\text{Mode 1, 3 baud rate} = \frac{2^{\text{SMOD\_X}}}{32} \times \frac{\text{Timer 1 input clock frequency}}{(256 - \text{TH1})}$$

Timer 1 input clock frequency can be found in Table 12-5, SMOD_x is the logic state of the baud-rate doubler bit for the associated UART, and TH1 is the user-assigned timer 1 reload value.

Often, users already know what baud rate is desired and need to calculate the timer reload value. An equation to calculate the timer reload value, TH1, follows:

$$\text{TH1} = 256 - \frac{2^{\text{SMOD\_X}} \times \text{timer 1 input clock frequency}}{32 \times \text{baud rate}}$$

Note that the 8-bit autoreload mode for timer 1 is the one most commonly used for serial port applications, but that it can actually be configured in any mode, even as a counter.

Using Timer 2 for Baud-Rate Generation

To use timer 2 as baud-rate generator for serial port 0, the timer is configured in autoreload mode. Then, either the TCLK or RCLK bit (or both) is set to a logic 1. TCLK = 1 selects timer 2 as the baud-rate generator for the transmitter and RCLK = 1 selects timer 2 for the receiver. Thus, serial port 0 can have the transmitter and receiver operating at different baud rates by choosing timer 1 for one data direction and timer 2 for the other. RCLK and TCLK reside in T2CON.4 and TCON.5, respectively.

Although the timer 2 input clock can be configured similarly to timer 1, it must be placed into a baud-rate generator mode in order to be used by serial port 0. Setting either RCLK or TCLK to a logic 1 selects timer 2 for baud-rate generation. When this is done, the timer 2 input clock becomes fixed to the oscillator frequency divided by 2. This is compatible with the 80C32. The only exception is when timer 2 is used for baud-rate generation within power-management mode. For PMM, the system clock (OSC / 1024) is used as the input clock for timer 2. The timer 2 interrupt is automatically disabled when either RCLK or TCLK is set. Also, the TF2 (TCON.7) flag cannot be set on a timer rollover. The manual reload pin, T2EX (P1.1), does not cause a reload either. Table 12-6 illustrates this relationship.

Table 12-6. Relationship Between External Crystal Frequency and Timer 2

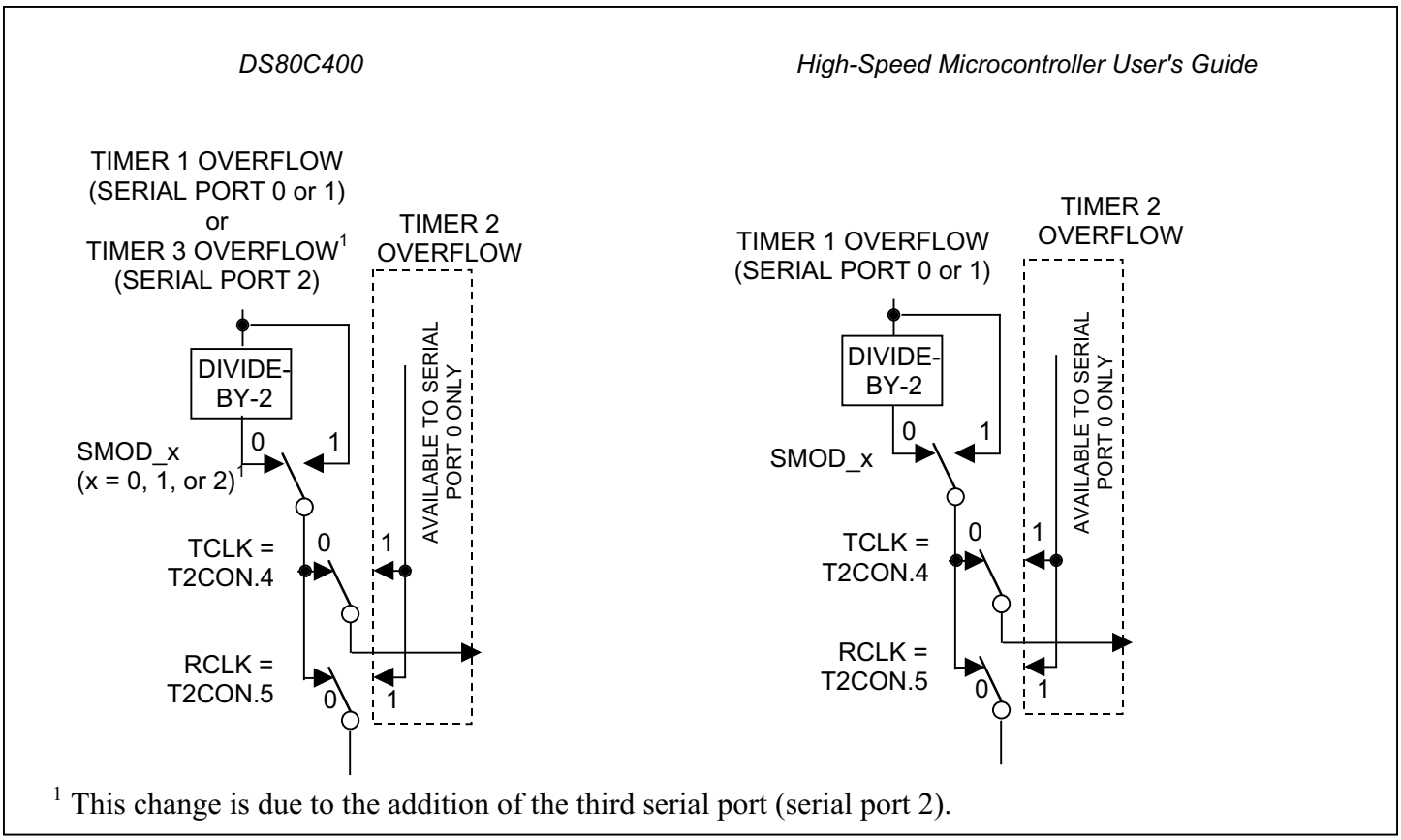
OSCILLATOR CYCLES PER MACHINE CYCLE	PMR REGISTER BITS 4X/2$\overline{X}$, CD1, CD0	TIMER 2 INPUT CLOCK FREQUENCY BAUD-RATE GENERATOR MODE (RCLK or TCLK = 1)
1 (4x mode)	100	OSC / 2
2 (2x mode)	000	OSC / 2
4 (default)	X01, X10	OSC / 2
1024 (PMM)	X11	OSC / 1024

When using timer 2 to generate baud rates, the formula is as follows. Note that the reload value is a 16-bit number as compared with timer 1, which uses only 8 bits. A second equation is provided here so that the timer 2 reload value can be calculated for a given baud rate:

$$\text{Mode 1, 3 baudrate} = \frac{1}{16} \times \frac{\text{Timer 2 Input Clock Frequency}}{65{,}536 - \text{RCAP2H, RCAP2L}}$$

$$\text{RCAP2H, RCAP2L} = 65{,}536 - \frac{\text{Timer 2 Input Clock Frequency}}{16 \times \text{Baud Rate}}$$

Timer 2 input clock frequency can be found in Table 12-6, and RCAP2H, RCAP2L is the user-assigned timer 2 reload value.



[1] This change is due to the addition of the third serial port (serial port 2).

Figure 12-3. Serial Port Modes 1, 3 Block Diagram Change

ADDENDUM TO SECTION 13: TIMED-ACCESS PROTECTION

A number of timed-access-protected bits are associated with the new features of the DS80C400. Please consult the High-Speed Microcontroller User's Guide *for complete information on the use of the timed-access feature.*

SFR	BIT(S)	BIT NAME	FUNCTION
EXIF (91h)	EXIF.0	BGS	Bandgap select
P4CNT (92h)	P4CNT.5–0		Port 4 pin configuration control bits
ACON (9Dh)	ACON.5	MROM	Merge ROM assignment
	ACON.4	BPME	Breakpoint mode enable
	ACON.3	BROM	Bypass ROM
	ACON.2	SA	Stack address mode
	ACON.1-0	AM1-AM0	Address mode-select bits
P5CNT (A2h)	P5CNT.2-0		Port 5 pin configuration control bits
C0C (A3h)	C0C.3	CRST	CAN 0 reset
P6CNT (B2h)	P6CNT.5–0		Port 6 pin configuration control bits
MCON (C6h)	MCON.7-6	IDM1-IDM0	Internal memory configuration bits
	MCON.5	CMA	CAN data memory assignment
	MCON.3–0	PDCE3–PDCE0	Program/data chip enables
COR (CEh)	COR.7	IRDACK	IRDA clock output enable
	COR.4-3	C0BPR7-C0BPR6	CAN 0 baud-rate prescale bits
	COR.2-1	COD1-COD0	CAN clock output divide bits
	COR.0	CLKOE	CAN clock output enable
MCON1 (D6h)	MCON1.3–0	PDCE7–PDCE4	Program/data chip enable
MCON2 (D7h)	MCON2.6-4	WPR2-WPR0	Write-protect range bits
	MCON2.3–0	WPE3–WPE0	Write-protect enable bits
WDCON (D8h)	WDCON.6	POR	Power-on reset flag
	WDCON.3	WDIF	Watchdog interrupt flag
	WDCON.1	EWT	Watchdog reset enable
	WDCON.0	RWT	Reset watchdog timer
EBS (E5h)	EBS.7	FPE	Flush filter failed-packet enable
	EBS.4–0	BS4–BS0	Buffer size configuration bits

ADDENDUM TO SECTION 16: INSTRUCTION SET DETAILS

The DS80C400 supports one of three different address modes, selected by the AM1 and AM0 bits in the ACON register. The processor operates in either the traditional 16-bit address mode, 24-bit paged address mode, or in a 24-bit contiguous address mode. When operating in the 16-bit addressing mode (AM1, AM0 = 00b), all instruction cycle timing and byte counts are identical to that found in Section 16 of the *High-Speed Microcontroller User's Guide*, with the exception of the INC DPTR instruction. The INC DPTR instruction for the DS80C400 executes in one machine cycle instead of three machine cycles, as required by previous high-speed microcontrollers.

The modification of the INC DPTR instruction is as follows.

MNEMONIC	INSTRUCTION CODE								HEX	BYTE	CYCLE	EXPLANATION
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0				
INC DPTR	1	0	1	0	0	0	1	1	A3	1	1	(DPTR) = (DPTR) + 1

Use of the 24-bit paged address mode is binary code-compliant with the traditional (16-bit) 8051 compilers, but allows for up to 16MB of program and 16MB of data memory to be supported by a new address page (AP) SFR, which controls an internal bank switch mechanism. The 24-bit contiguous mode requires a compiler that supports contiguous program flow over the entire 24-bit address range by the addition of an operand and/or cycles to seven basic instructions.

16-BIT (8051 STANDARD) ADDRESSING MODE

This addressing mode is identical to that used by the 8051 family and most members of the high-speed microcontroller. The microcontroller defaults to this mode following a reset. This mode can also be used to run code compiled or assembled for the 24-bit contiguous mode, as long as the following five instructions are not executed:

```
MOV DPTR, #data24
ACALL addr19
LCALL addr24
AJMP addr19
LJMP addr24
```

These five branch instructions are the only instructions that cause the compiler to generate additional operands relative to the 16-bit addressing mode. Note that the number of cycles per instruction can appear different from other instructions, but this is ignored by most assemblers or compilers and, as such, does not pose a problem with the binary output.

By selecting the 24-bit contiguous mode prior using any one of these five branch instructions, it is possible to run 24-bit contiguous compiled code in the default 16-bit address configuration. Once the AM0 and AM1 bits are set to the 24-bit contiguous address mode, the previously mentioned instructions execute properly. When the 24-bit paged address mode is selected, all instructions compiled under the traditional 16-bit address mode execute normally at any point in code.

24-BIT PAGED ADDRESSING MODE

The DS80C400 incorporates an internal 8-bit address page register (AP), four 8-bit extended data pointer registers (DPX, DPX1, DPX2, DPX3), and an 8-bit MOVX extended address register (MXAX) as hardware support for 24-bit addressing in the paged address mode (AM1, AM0 = 01b). This mode has four differences in code execution from the traditional 16-bit mode.

1) The first difference is the addition of one machine cycle when executing the ACALL, LCALL, RET, and RETI instructions, as well as when hardware vectors to and returns from an interrupt. This change should be transparent to most compilers, as the byte count remains identical for these instructions.
2) The second involves register-indirect MOVX instructions such as `MOVX @Ri, A` or `MOVX A, @Ri`. When in this mode, the MXAX register supplies the upper 8 bits of the 24-bit MOVX address. The complete address is formed by concatenating MXAX, P2, and R1 or R0 in this mode. The DPTR-related MOVX instructions do not utilize the P2 and MXAX registers.
3) The third involves MOVX and MOVC operations that utilize one of the four data pointers. When DPTR is referenced by a MOVX or MOVX instruction, the complete address is formed by concatenating the DPX, DPH, and DPL registers for the actively selected data pointer.
4) The fourth involves the LJMP instruction. Although the byte count and cycle count are identical to the 16-bit addressing mode, execution of the LJMP results in the AP register being used as the upper byte of the 24-bit program counter for the jump destination.

The DS80C400 supports interrupts from any location in the 24-bit address field. When an interrupt request is acknowledged, the cur-

rent contents of the 24-bit program counter (PC) are pushed onto the stack, and the page value (00h) and the lower 16-bit address of the interrupt vector are then written to the PC before the execution of the hardware LCALL. This means that all interrupt vectors are fetched from address 0000xxh, rather than the current page as defined by the AP register. The RETI instruction pops the three address bytes from the stack and restores these bytes back to the PC at the conclusion of the interrupt service routine. Interrupt service routines that branch over page boundaries must save the current contents of AP before altering the AP register, as it is not automatically saved on the stack. This mechanism supports up to three levels of nesting for interrupts.

One extra machine cycle is required to handle the additional byte associated with the extension to 24-bit addressing. The storage of the 24-bit address during an interrupt, LCALL, or ACALL instruction also requires three bytes of stack memory, as opposed to the traditional two bytes in the 16-bit address mode. In this mode, the third byte of the PC (PC[23:16]) is not incremented when the lower 16 bits in the lower two bytes of the PC (PC[15:0]) rolls over from FFFFh to 0000h. In the 24-bit paged address mode, PC[23:16] functions only as a storage register, which is loaded by the address page (AP) register whenever the processor executes an ACALL, LCALL, or LJMP instruction. PC[23:16] is stored and retrieved from the stack with the lower 16-bit of address in PC[15:0] when stack operation is required.

In paged address mode, MOVX instructions that utilize the data pointers (such as `MOVX @DPTR, A`) form the 24-bit data address by concatenating the contents of the currently selected extended DPTR register (DPX, DPX1, DPX2, or DPX3) with the contents of the DPTR. The extended data pointer register values are not affected when the lower 16 bits of the selected DPTR overflows or underflows.

To maintain compatibility with existing 8051 compilers, the `JMP @A+DPTR` and `MOVC A, @A+DPTR` instructions from 24-bit paged address mode are limited to the current 64kB page, as specified by the upper 8 bits of the current instruction execution address register. There is not a carry function into the contents of the extended data pointer register (DPX, DPX1, DPX2, or DPX3).

The modification of instructions in the 24-bit paged address mode is summarized in the following table.

MNEMONIC	INSTRUCTION CODE								HEX	BYTE	CYCLE	EXPLANATION
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0				
ACALL addr11	a_{10} a_7	a_9 a_6	a_8 a_5	1 a_4	0 a_3	0 a_2	0 a_1	1 a_0	Byte 1 Byte 2	2	4	$(PC_{15:0}) = (PC_{15:0}) + 2$ (SP) = (SP) + 1 ((SP)) = $(PC_{7:0})$ (SP) = (SP) + 1 ((SP)) = (PC_{15-8}) (SP) = (SP) + 1 ((SP)) = $(PC_{23:16})$ $(PC_{10:0})$ = addr11 $(PC_{23:16}) = (AP_{7:0})$
LCALL addr16	0 a_{15} a_7	0 a_{14} a_6	0 a_{13} a_5	1 a_{12} a_4	0 a_{11} a_3	0 a_{10} a_2	1 a_9 a_1	0 a_8 a_0	12 Byte 2 Byte 3	3	5	$(PC_{15:0}) = (PC_{15:0}) + 3$ (SP) = (SP) + 1 ((SP)) = $(PC_{7:0})$ (SP) = (SP) + 1 ((SP)) = (PC_{15-8}) (SP) = (SP) + 1 ((SP)) = $(PC_{23:16})$ (PC) = addr16 $(PC_{23:16}) = (AP_{7:0})$
LJMP addr16	0 a_{15} a_7	0 a_{14} a_6	0 a_{13} a_5	0 a_{12} a_4	0 a_{11} a_3	0 a_{10} a_2	1 a_9 a_1	0 a_8 a_0	02 Byte 2 Byte 3	3	4	(PC) = addr16 $(PC_{23:16}) = (AP_{7:0})$
RET	0	0	1	0	0	0	1	0	22	1	5	$(PC_{23:16})$ = ((SP)) (SP) = (SP) - 1 (PC_{15-8}) = ((SP)) (SP) = (SP) - 1 $(PC_{7:0})$ = ((SP)) (SP) = (SP) - 1
RETI	0	0	1	1	0	0	1	0	32	1	5	$(PC_{23:16})$ = ((SP)) (SP) = (SP) - 1 (PC_{15-8}) = ((SP)) (SP) = (SP) - 1 (PC_{7-0}) = ((SP)) (SP) = (SP) - 1

24-BIT CONTIGUOUS ADDRESSING MODE

When the AM1 bit is set, the DS80C400 operates in its 24-bit contiguous addressing mode. This addressing mode supports a full 24-bit program counter and eight modified instructions that operate over the full 24-bit address range. All modified branching instructions automatically store and restore the entire contents of the 24-bit program counter. The 24-bit DPTR, DPTR1, DPTR2, and DPTR3 registers function identically to the program counter to allow access to the full 24-bit data memory range.

All the DS80C400 instruction op codes retain binary compatibility to the 8051. Modified instructions are different only with respect to their cycle/byte/operand count and operate within a contiguous 24-bit address field. Note that all instructions utilizing the DPTR register now make use of a full 24-bit register (DPTR = DPXn + DPHn + DPLn where n = 0, 1, 2, or 3). This mode of operation requires software tools (assembler or compiler) specifically designed to accept the modified length of the new instructions.

In addition, the 24-bit contiguous mode utilizes the MXAX register to supply the upper 8 bits of the 24-bit MOVX address during register-indirect MOVX instructions such as `MOVX @Ri, A` or `MOVX A, @Ri`. In this mode, the complete MOVX address is formed by concatenating MXAX, P2, and R1 or R0. The DPTR-related MOVX instructions do not utilize the P2 and MXAX register.

The instructions modified to operate in the 24-bit contiguous address mode are summarized in the following table.

MNEMONIC	INSTRUCTION CODE								HEX	BYTE	CYCLE	EXPLANATION
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0				
ACALL addr 19	a_{18} a_{15} a_7	a_{17} a_{14} a_6	a_{16} a_{13} a_5	1 a_{12} a_4	0 a_{11} a_3	0 a_{10} a_2	0 a_9 a_1	1 a_8 a_0	Byte 1 Byte 2 Byte 3	3	5	(PC) = (PC) + 3 (SP) = (SP) + 1 ((SP)) = ($PC_{7:0}$) (SP) = (SP) + 1 ((SP)) = ($PC_{15:8}$) (SP) = (SP) + 1 ((SP)) = ($PC_{23:16}$) ($PC_{18:0}$) = addr19
AJMP addr 19	a_{18} a_{15} a_7	a_{17} a_{14} a_6	a_{16} a_{13} a_5	0 a_{12} a_8	0 a_{11} a_3	0 a_{10} a_2	0 a_9 a_1	1 a_8 a_0	Byte 1 Byte 2 Byte 3	3	5	(PC) = (PC) + 3 ($PC_{18:0}$) = addr19
LCALL addr24	0 a_{23} a_{15} a_7	0 a_{22} a_{14} a_6	0 a_{21} a_{13} a_5	1 a_{20} a_{12} a_4	0 a_{19} a_{11} a_3	0 a_{18} a_{10} a_2	1 a_{17} a_9 a_1	0 a_{16} a_8 a_0	12 Byte 2 Byte 3 Byte 4	4	6	(PC) = (PC) + 4 (SP) = (SP) + 1 ((SP)) = ($PC_{7:0}$) (SP) = (SP) + 1 ((SP)) = ($PC_{15:8}$) (SP) = (SP) + 1 ((SP)) = ($PC_{23:16}$) ($PC_{23:0}$) = addr24
LJMP addr24	0 a_{23} a_{15} a_7	0 a_{22} a_{14} a_6	0 a_{21} a_{13} a_5	0 a_{20} a_{12} a_4	0 a_{19} a_{11} a_3	0 a_{18} a_{10} a_2	1 a_{17} a_9 a_1	0 a_{16} a_8 a_0	02 Byte 2 Byte 3 Byte 4	4	5	($PC_{23:0}$) = addr24
MOV DPTR, #data24	1 d_{23} d_{15} d_7	0 d_{22} d_{14} d_6	0 d_{21} d_{13} d_5	1 d_{20} d_{12} d_4	0 d_{19} d_{11} d_3	0 d_{18} d_{10} d_2	0 d_{17} d_9 d_1	0 d_{16} d_8 d_0	90 Byte 2 Byte 3 Byte 4	4	3	(DPX) = #data23:9 (DPH) = #data15:8 (DPL) = #data7:0
RET	0	0	1	0	0	0	1	0	22	1	5	($PC_{23:16}$) = ((SP)) (SP) = (SP) - 1 ($PC_{15:8}$) = ((SP)) (SP) = (SP) - 1 ($PC_{7:0}$) = ((SP)) (SP) = (SP) - 1
RETI	0	0	1	1	0	0	1	0	32	1	5	($PC_{23:16}$) = ((SP)) (SP) = (SP) - 1 ($PC_{15:8}$) = ((SP)) (SP) = (SP) - 1 ($PC_{7:0}$) = ((SP)) (SP) = (SP) - 1

SECTION 17: TROUBLESHOOTING

SOFTWARE BREAKPOINT MODE

The DS80C400 provides a special breakpoint mode of operation to promote greater visibility and control during the code development cycle. When a breakpoint is generated, the following occurs:

- Clocks to Timers 0, 1, 2, 3 and watchdog timer are halted.
- Serial port activity, if driven by any of the Timers, will be halted.
- Timed Access state machine (for limiting access time to certain registers) is disabled.
- Hardware LCALL to 000083h.
- Return address (address immediately following the breakpoint instruction) is placed in the BPA3, BPA2, and BPA1 special function registers instead of the stack.

Temporarily disabling the clock driven functions allows execution of breakpoint code without disrupting the relationship between original program code and hardware timed functions. Insertion of breakpoint code will, however, alter the absolute timing relationship between the DS80C400 and externally interfaced devices. Issuing a breakpoint while transmitting or receiving serial data, for example, might easily stop the bit timing clocks needed for proper transmission/reception. Special attention should be paid to placement of breakpoints if critical peripheral interaces rely upon the clock sources which are halted during the breakpoint.

GENERATING A BREAKPOINT

In order to generate a breakpoint, breakpoint operation must first be enabled. The breakpoint mode of operation can be enabled by setting the BPME bit (ACON.4) to a '1'. Write access to this register bit requires Timed Access. Once enabled, a breakpoint can be generated by executing the breakpoint opcode (A5h). This opcode can physically be placed in the user code or can be force fed to the CPU using external hardware. The A5h instruction generates what could be considered a software interrupt. Much the same as an interrupt, code execution always branches to a fixed address location (000083h) and can jump/branch to other locations in order to service the breakpoint. Unlike an interrupt, the processor does not store the return address on the stack, but instead places them in special Breakpoint Address Registers. This prevents the stack and the stack pointer from being altered when issuing a breakpoint. The three Breakpoint Address SFR Registers, located at DAh, DBh and DCh, are provided to store the breakpoint address LSB, MSB and XSB values respectively. These registers are accessible during the breakpoint routine to allow the breakpoint software to return control to a different point than the original breakpoint, if so desired. These registers are not available outside the breakpoint routine and are always overwritten when a new breakpoint is initiated.

EXITING A BREAKPOINT

Execution of the same 'A5h' instruction that was used to generate the breakpoint is used to exit the breakpoint. When exiting a breakpoint, the processor returns to the address pointed to by the Breakpoint Address Registers. When BPME=1, the 'A5h' instruction serves as the start/stop toggle switch for execution of breakpoint code.

As noted earlier, the 'A5h' breakpoint opcode can be phsically inserted in the user code for the purpose of generating a breakpoint or it can be force fed to the processor in place of an opcode which exists in physical memory. If being force fed to the processor, it is important to note specific timing issues with respect to the processor opcode sampling.

All instructions other than MOVC and MOVX latch the incoming opcode during C4 time of the last machine cycle of the current instruction. Since the instruction fetch and memory access machine cycles for the MOVC and MOVX instructions are not contiguous in time, the incoming instruction is actually sampled on C4 of the first machine cycle for the MOVX and C4 of the first and second machine cycle of the MOVC. The three instruction timing possibilities are shown in the figures below. Note also that interrupts are sampled during the C3 time period of the last machine cycle of each instruction. A pending interrupt takes control of the system at the start of the next machine cycle. As a result, the sampled incoming instruction is ignored until the interrupt vectoring process has been completed. Monitoring of the address can determine if the force-fed 'A5h' or an interrupt was processed.

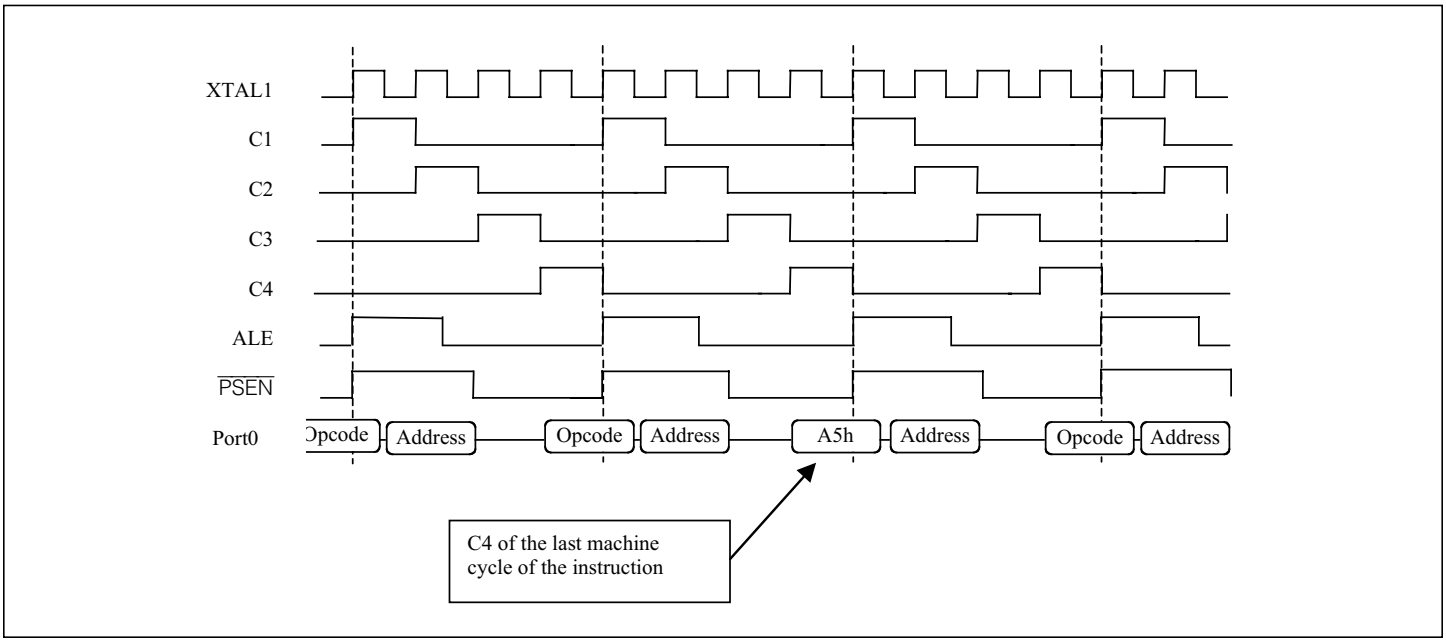


Figure 17-1. Force Feeding a Breakpoint During An Instruction Other Than Movc Or Movx

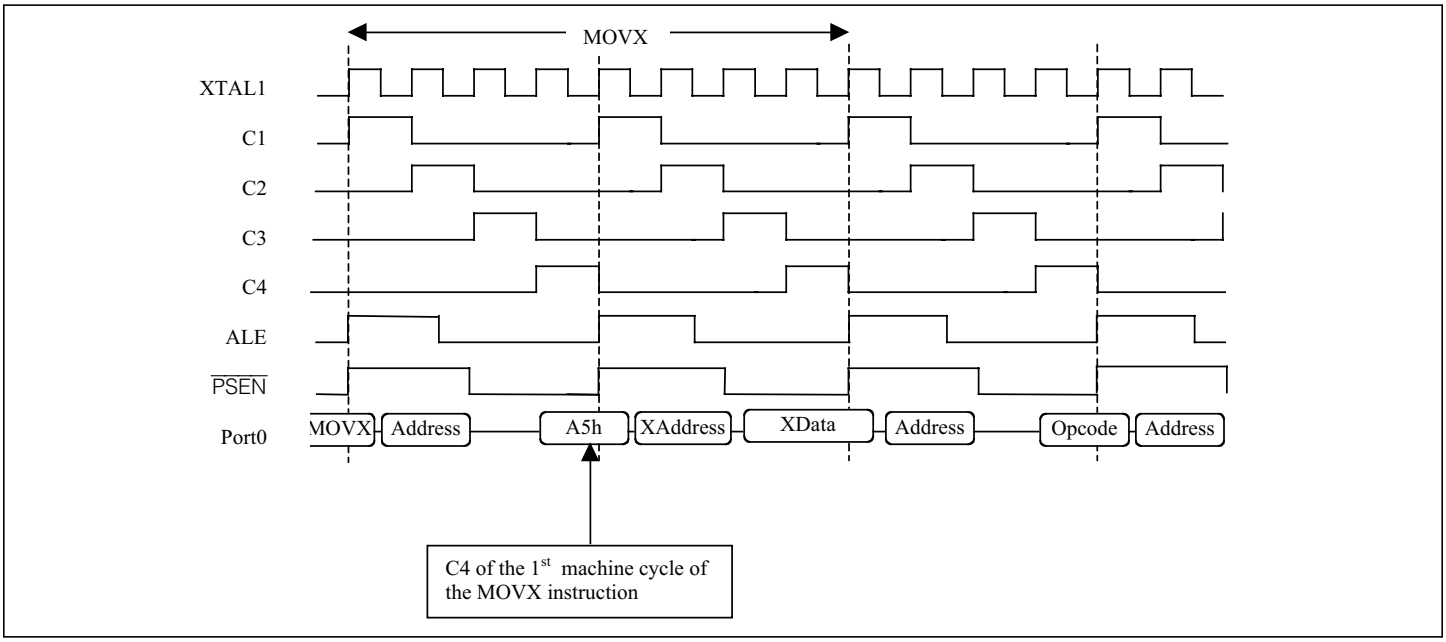


Figure 17-2 Force Feeding a Breakpoint During a Movx (2-Cycle)

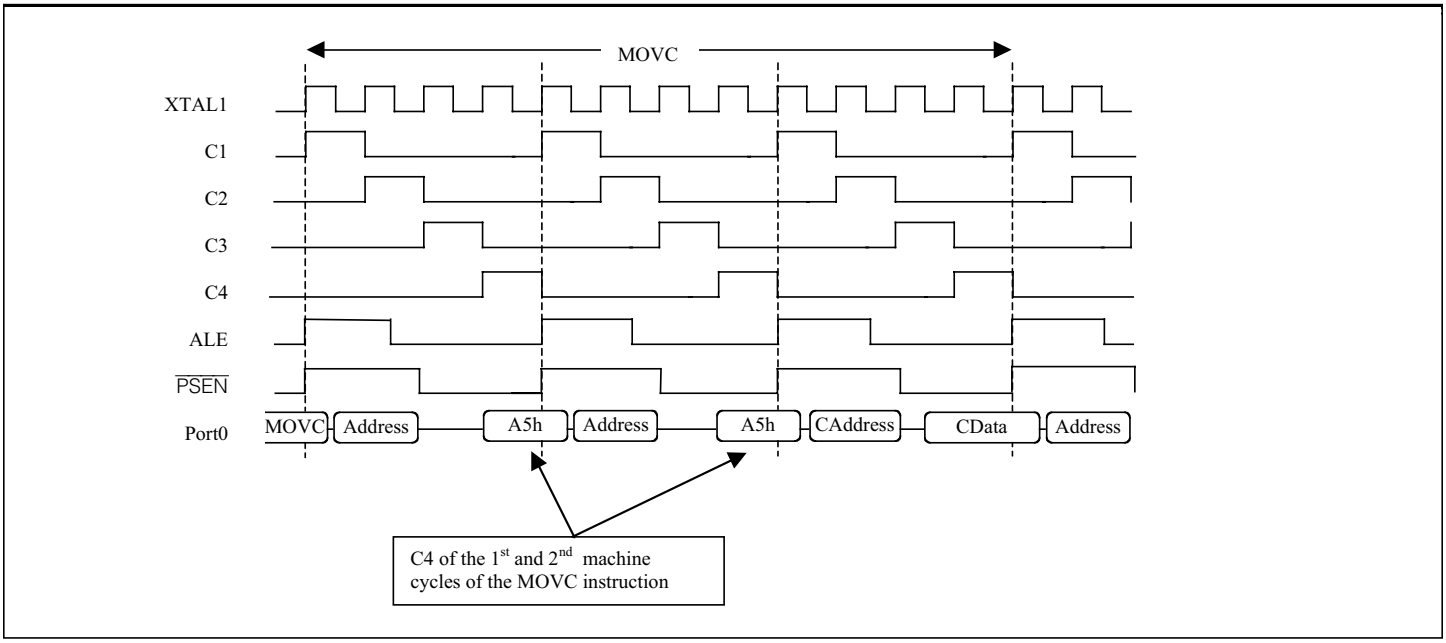


Figure 17-3. Force Feeding a Breakpoint Movc
Note: The figures above are intended to provide a high-level overview of where the instruction sampling occurs for various instruction possibilities. For exact timing constraints, please reference the AC specs contained in the DS80C400 datasheet.

SECTION 19: CONTROLLER AREA NETWORK (CAN) MODULE

The DS80C400 incorporates a single CAN controller (CAN 0), which provides operating modes that are fully compliant with the CAN 2.0B specification. The microcontroller interface to the CAN controller is broken into two groups of registers. To simplify the software associated with the operation of the CAN controller, all of the global CAN status and controls, as well as the individual message center control/status registers, are located in the SFR map. The remaining registers associated with the data identification, identification masks, format and data are located in the MOVX space. Each of the SFR and MOVX registers are configured as dual port memories to allow both the CAN controller and the microcontroller access to the required functions.

The basic functions covered by the CAN controller begin with the capability to use 11-bit standard or 29-bit extended acceptance identifiers, as programmed by the microcontroller for each message center. The CAN unit provides 15 message centers, each having a standard 8-byte data field. The first 14 message centers are programmable in either a transmit or receive mode. Message center 15 is designed as a receive-only message center and implements a second data buffer to prevent the inadvertent loss of data when the microcontroller is busy. This FIFO buffer is utilized when the microcontroller is not allowed time to retrieve the incoming message prior to the acceptance of a second message into message center 15. Message center 15 also utilizes an independent set of mask registers and identification registers, which are only applied once an incoming message has not been accepted by any of the first 14 message centers. A second filter test is also supported for all message centers (1–15) to allow the CAN controller to use two separate 8-bit media masks and media arbitration fields to verify the contents of the first two bytes of data of each incoming message, before accepting an incoming message. This feature allows the CAN unit to directly support the use of higher CAN protocols, which make use of the first and/or second byte of data as a part of the acceptance layer for storing incoming messages. Each message center can also be programmed independently to perform testing of the incoming data, with or without the use of the global masks.

Global controls and status registers in the CAN module allow the microcontroller to evaluate error messages, validate new data and the location of such data, establish the bus timing for the CAN bus, establish the identification mask bits, and verify the source of individual messages. In addition, each message center is individually equipped with the necessary status and controls to establish directions, interrupt generation, identification mode (standard or extended), data field size, data status, automatic remote frame request and acknowledgment, and masked or nonmasked identification acceptance testing.

The priority order associated with the CAN module transmitting or receiving a message is determined by the inverse of the number of the message center and is independent of the arbitration bits assigned to the message center. Thus, message center 2 has a higher priority than message center 14. To avoid a priority inversion, the CAN module is designed to reload the transmit buffer with the message of the highest priority (lowest message center number) whenever an arbitration is lost or an error condition occurs.

The following tables illustrate the locations of the MOVX SRAM registers and bits used by the CAN controller. Following the tables are descriptions of the function of the bits and registers.

MOVX MESSAGE CENTERS FOR CAN 0

CAN 0 CONTROL/STATUS/MASK REGISTERS									
REGISTER	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**MOVX DATA ADDRESS[1]**
C0MID0	MID07	MID06	MID05	MID04	MID03	MID02	MID01	MID00	xxxx00h
C0MA0	M0AA7	M0AA6	M0AA5	M0AA4	M0AA3	M0AA2	M0AA1	M0AA0	xxxx01h
C0MID1	MID17	MID16	MID15	MID14	MID13	MID12	MID11	MID10	xxxx02h
C0MA1	M1AA7	M1AA6	M1AA5	M1AA4	M1AA3	M1AA2	M1AA1	M1AA0	xxxx03h
C0BT0	SJW1	SJW0	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	xxxx04h
C0BT1	SMP	TSEG26	TSEG25	TSEG24	TSEG13	TSEG12	TSEG11	TSEG10	xxxx05h
C0SGM0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	xxxx06h
C0SGM1	ID20	ID19	ID18	0	0	0	0	0	xxxx07h
C0EGM0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	xxxx08h
C0EGM1	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	xxxx09h
C0EGM2	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	xxxx0Ah
C0EGM3	ID4	ID3	ID2	ID1	ID0	0	0	0	xxxx0Bh
C0M15M0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	xxxx0Ch
C0M15M1	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	xxxx0Dh
C0M15M2	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	xxxx0Eh
C0M15M3	ID4	ID3	ID2	ID1	ID0	0	0	0	xxxx0Fh
CAN 0 MESSAGE CENTER 1									
	Reserved								xxxx10h-11h
C0M1AR0	CAN 0 MESSAGE 1 ARBITRATION REGISTER 0								xxxx12h
C0M1AR1	CAN 0 MESSAGE 1 ARBITRATION REGISTER 1								xxxx13h
C0M1AR2	CAN 0 MESSAGE 1 ARBITRATION REGISTER 2								xxxx14h
C0M1AR3	CAN 0 MESSAGE 1 ARBITRATION REGISTER 3							WTOE	xxxx15h
C0M1F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	T/$\overline{R}$	EX/$\overline{ST}$	MEME	MDME	xxxx16h
C0M1D0–7	CAN 0 MESSAGE 1 DATA BYTES 0–7								xxxx17h–1Eh
	Reserved								xxxx1Fh

	CAN 0 MESSAGE CENTERS 2–14								
	MESSAGE CENTER 2 REGISTERS (similar to message center 1)								xxxx20h–2Fh
	MESSAGE CENTER 3 REGISTERS (similar to message center 1)								xxxx30h–3Fh
	MESSAGE CENTER 4 REGISTERS (similar to message center 1)								xxxx40h–4Fh
	MESSAGE CENTER 5 REGISTERS (similar to message center 1)								xxxx50h–5Fh
	MESSAGE CENTER 6 REGISTERS (similar to message center 1)								xxxx60h–6Fh
	MESSAGE CENTER 7 REGISTERS (similar to message center 1)								xxxx70h–7Fh
	MESSAGE CENTER 8 REGISTERS (similar to message center 1)								xxxx80h–8Fh
	MESSAGE CENTER 9 REGISTERS (similar to message center 1)								xxxx90h–9Fh
	MESSAGE CENTER 10 REGISTERS (similar to message center 1)								xxxxA0h–AFh
	MESSAGE CENTER 11 REGISTERS (similar to message center 1)								xxxxB0h–BFh
	MESSAGE CENTER 12 REGISTERS (similar to message center 1)								xxxxC0h–CFh
	MESSAGE CENTER 13 REGISTERS (similar to message center 1)								xxxxD0h–DFh
	MESSAGE CENTER 14 REGISTERS (similar to message center 1)								xxxxE0h–EFh
	CAN 0 MESSAGE CENTER 15								
—	Reserved								xxxxF0h–F1h
C0M15AR0	CAN 0 MESSAGE 15 ARBITRATION REGISTER 0								xxxxF2h
C0M15AR1	CAN 0 MESSAGE 15 ARBITRATION REGISTER 1								xxxxF3h
C0M15AR2	CAN 0 MESSAGE 15 ARBITRATION REGISTER 2								xxxxF4h
C0M15AR3	CAN 0 MESSAGE 15 ARBITRATION REGISTER 3							WTOE	xxxxF5h
C0M15F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	0	EX/$\overline{ST}$	MEME	MDME	xxxxF6h
C0M15D0–7	CAN 0 MESSAGE 15 DATA BYTE 0–7								xxxxF7h–FEh
	Reserved								xxxxFFh

[1]The first 2 bytes of the CAN 0 MOVX memory address are dependent on the setting of the CMA bit (MCON.5) CMA = 0, xxxx = 00DB; CMA = 1, xxxx = FFDB.

CAN MOVX Register Description

Most of the SRAM control registers, including the message centers proper, are mapped into a special location in the MOVX SRAM space. The specific location of the registers is a function of the CMA bit (MCON.5), which controls whether the CAN SRAM begins at location FFDBxxh or 00DBxxh.

The MOVX CAN registers consist of a set of one control/status/mask register and 15 message centers. Write access to the control/status/mask registers is possible only when the SWINT bit is set to 1. All message centers for a given CAN module are identical, with the exception of 15, which has some minor differences noted in the register descriptions. To simplify the documentation, only one set of registers is shown, with the following generic notation used for register names and addresses:

xxxx First four hexadecimal digits of register MOVX address

CMA	CAN 0
0	00DB (reset default)
1	FFDB

y Address based on message center number

Y	MESSAGE CENTER NUMBER
1	1
2	2
—	—
A	10
F	15

CAN 0 Media ID Mask Register 0 (C0MID0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx00h								

CAN 0 Media ID Mask Register 1 (C0MID1)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx02h								

CAN 0 media ID mask registers 1-0. These registers function as the mask when performing the media identification test. This register can be only modified during a software initialization (SWINT = 1). If MDME = 0, the media identification test cannot be performed, and the contents of these registers is ignored. If MDME = 1, the CAN module performs an additional qualifying test on data bytes 0 and 1 of the incoming message, regardless of the state of the EX/$\overline{ST}$ bit. Data byte 1 is compared against CAN media byte arbitration register 1 utilizing C0MID1 as a mask, and data byte 0 is compared against CAN media byte arbitration register 0 utilizing C0MID0 as a mask. Any bit in the C0MID1, C0MID0 masks programmed to 0 ignores the state of the corresponding data byte bit when performing the test. Any bit in the C0MID1, C0MID0 masks programmed to 1 forces the state of the corresponding data byte bit and CAN media byte arbitration registers 1 and 0 to match before considering the incoming message a match. Programming either media ID mask register to 00h effectively disables the media ID test for that byte. As such, the C0MID1, C0MID0 masks act as a "don't care" following a system reset.

CAN 0 Media Arbitration Register 0 (C0MA0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx01h								

CAN 0 Media Arbitration Register 1 (C0MA1)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx03h								

CAN 0 media arbitration register 1-0. These registers function as the arbitration field when performing the media identification test. If MDME = 0, the media identification test cannot be performed and the contents of these registers is ignored. If MDME = 1, the CAN module performs an additional qualifying test on data bytes 0 and 1 of the incoming message, as mentioned in the description of the CAN media ID mask registers. This register can be modified only during a software initialization (SWINT = 1).

CAN 0 Bus Timing Register 0 (C0BT0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx04h	SJW1	SJW0	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0

SJW1, SJW0
Bits 7-6

CAN synchronization jump width select. These bits specify the maximum number of time quanta (t_{qu}) cycles that a bit can be lengthened or shortened in one resynchronization to compensate for phase errors detected by the CAN controller when receiving data. These bits can be modified only during a software initialization (SWINT = 1).

SJW1	SJW0	SYNCHRONIZATION JUMP WIDTH (Number in parentheses is SJW value used in bit timing calculations)
0	0	1 t_{qu} (1)
0	1	2 t_{qu} (2)
1	0	3 t_{qu} (3)
1	1	4 t_{qu} (4)

BPR5–BPR0
Bits 5–0

CAN baud-rate prescaler. These bits specify the lower 6 bits (BPR5–BPR0) of the 8-bit prescale value (BPR7–BPR0). The 256 states defined by the binary combinations of the BPR7–BPR0 bits determine the value of the prescale that, in turn, defines the cycle time associated with one-time quanta. These bits can be modified only during a software initialization (SWINT = 1). The BPR7, BPR6 bits are located in the COR (CEh) SFR.

BPR7, BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	BAUD-RATE PRESCALE VALUE (BRPV)
00	0	0	0	0	0	0	1
00	0	0	0	0	0	1	2
—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—
11	1	1	1	1	1	0	255
11	1	1	1	1	1	1	256

CAN 0 Bus Timing Register 1 (C0BT1)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx05h	SMP	TSEG26	TSEG25	TSEG24	TSEG13	TSEG12	TSEG11	TSEG10

SMP
Bit 7

CAN sampling rate. The sampling rate (SMP) bit determines the number of samples to be taken during each receive bit time. Programming SMP = 0 takes only one sample during each bit time. Programming SMP = 1 directs the CAN logic to take three samples during each bit time, and to use a majority voting circuit to determine the final bit value. When SMP is set to a 1, two addition al t_{qu} clock cycles are added to time segment 1. SMP should not be set to 1 when the baud-rate prescale value (BRPV) is less than 4. This bit can be modified only during a software initialization (SWINT = 1).

TSEG26-24
Bits 6-4

CAN time segment 2 select. The eight states defined by the TSEG26–TSEG24 bits determine the number of clock cycles in the phase segment 2 portion of the nominal bit time, which occurs after the sample time. These bits can be modified only during a software initialization (SWINT = 1).

TSEG26	TSEG25	TSEG24	TIME SEGMENT 2 LENGTH (Number in parentheses is TS2_LEN value used in bit timing calculations)
0	0	0	Invalid
0	0	1	2 t_{qu} (2)
0	1	0	3 t_{qu} (3)
—	—	—	—
1	1	0	7 t_{qu} (7)
1	1	1	8 t_{qu} (8)

TSEG13–10

CAN time segment 1 select. The 16 states defined by the TSEG13–TSEG10 bits determine the

Bits 3–0 number of clock cycles in the phase segment 1 portion of the nominal bit time, which occurs before the sample time. These bits can be modified only during a software initialization (SWINT = 1)

TSEG13	TSEG12	TSEG11	TSEG10	TIME SEGMENT 1 LENGTH (Number in parentheses is TS1_LEN value used in bit timing calculations)
0	0	0	0	Invalid
0	0	0	1	2 t_{qu} (2)
0	0	1	0	3 t_{qu} (3)
–	–	–	–	–
1	1	1	0	15 t_{qu} (15)
1	1	1	1	16 t_{qu} (16)

CAN 0 Standard Global Mask Register 0 (C0SGM0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx06h	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21

CAN 0 Standard Global Mask Register 1 (C0SGM1)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx07h	MASK20	MASK19	MASK18	0	0	0	0	0

CAN standard global mask registers 1–0. These registers function as the mask when performing the 11-bit global identification test on incoming messages for message centers 1–14. If message identification masking is disabled (MEME = 0), the incoming message ID field must match the corresponding message center arbitration value exactly, effectively ignoring the contents of these registers. These registers are used only when performing the message identification test for message centers configured as standard receivers (EX/= 0) having message ID masking enabled (MEME = 1). Thus, the contents are ignored by message centers configured to receive messages with extended identifiers (EX/= 1). These registers can be modified only during a software initialization (SWINT = 1).

When MEME = 1, any mask bit in the C0SGM1, C0SGM0 mask programmed to a 0 creates a "don't care" condition when the respective bit in the incoming message ID field is compared with the corresponding arbitration bits in message centers 1–14. Any bit in these masks programmed to a 1 forces the respective bit in the incoming message ID field to match identically with the corresponding arbitration bits in message centers 1–14 before said message is loaded into message centers 1–14.

The 5 least significant bits in the C0SGM1 register are not used and do not perform any comparison of these bit locations. A read of these bits produces the last bit values written to these bit locations by the microcontroller or an indeterminate value following a power-up.

CAN 0 Extended Global Mask Register 0 (C0EGM0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx08h	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21

CAN 0 Extended Global Mask Register 1 (C0EGM1)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx09h	MASK20	MASK19	MASK18	MASK17	MASK16	MASK15	MASK14	MASK13

CAN 0 Extended Global Mask Register 2 (C0EGM2)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx0Ah	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5

CAN 0 Extended Global Mask Register 3 (C0EGM3)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx0Bh	MASK4	MASK3	MASK2	MASK1	MASK0	0	0	0

CAN 0 extended global mask registers 0–3. These registers function as the mask when performing the extended global identification test (EX/$\overline{ST}$= 1) when message ID masking is enabled (MEME = 1) for message centers 1–14. When EX/$\overline{ST}$= 0 or MEME = 0 for a given message center, the contents of this register are ignored. These registers can be modified only during a software initialization (SWINT = 1).

When EX/$\overline{ST}$= 1, the 29 bits of the message ID are compared against the 29 bits of the CAN message center y arbitration registers, using the 29 bits of the CAN extended global mask registers as a mask. Any bit in the extended global mask registers set to 0 ignores the state of the corresponding bit in the incoming message ID field when performing the test. Any bit in the extended global mask registers set to 1 forces the state of the corresponding bit in the incoming message ID field and CAN message center arbitration registers 0–3 to match before considering the incoming message a match.

The 3 least significant bits in the C0EGM3 are not used and do not perform any comparison of these bit locations. A read of these bits always returns 0, and writes to these bits are ignored.

Programming all mask registers to 00h effectively disables the global ID test for that message, accepting all messages. As such, the global mask registers act as a "don't care" following a system reset.

CAN 0 Message Center 15 Mask Register 0 (C0M15M0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx0Ch	MASK28	MASK27	MASK26	MASK25	MASK24	MASK23	MASK22	MASK21

CAN 0 Message Center 15 Mask Register 1 (C0M15M1)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx0Eh	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5

CAN 0 Message Center 15 Mask Register 2 (C0M15M2)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx0Eh	MASK12	MASK11	MASK10	MASK9	MASK8	MASK7	MASK6	MASK5

CAN 0 Message Center 15 Mask Register 3 (C0M15M3)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx0Fh	MASK4	MASK3	MASK2	MASK1	MASK0	0	0	0

MASK28–MASK0

CAN message center 15 mask registers 0–3. These registers function as the mask for the standard (EX/$\overline{ST}$ = 0) or extended (EX/$\overline{ST}$ = 1) global identification test (EX/$\overline{ST}$ = 1) when message ID masking has been enabled (MEME = 1) for message center 15. These registers can only be modified during a software initialization (SWINT = 1).

When EX/$\overline{ST}$ = 1, the 29 bits of the message ID are compared against the 29 bits of the CAN message center 15 arbitration registers, using the 29 bits of the CAN message center 15 mask registers as as mask. When EX/$\overline{ST}$ = 0, the 11 bits of the message ID are compared against the most significant 11 bits of the CAN message center 15 arbitration registers, using the most significant 11 bits of the CAN message center 15 mask registers as a mask. Any bit in the CAN 0 message center 15 mask registers set to 0 ignores the state of the corresponding bit in the incoming message ID field when performing the test. Any bit in the CAN message center 15 mask registers set to 1 forces the state of the corresponding bit in the incoming message ID field and CAN message center arbitration registers 0–3 to match before considering the incoming message a match.

The 3 least significant bits in the CnM15M3 register are not used and do not perform any comparison of these bit locations. A read of these bits always returns 0, and writes to these bits are ignored.

Programming all mask registers to 00h effectively disables the message center 15 ID test, accepting all messages. As such, the message center 15 mask registers act as a "don't care" following a system reset.

CAN MESSAGE CENTER MOVX REGISTER DESCRIPTIONS

CAN 0 Message Center y Arbitration Register 0 (C0MyAR0)

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxxy2h	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21

CAN 0 Message Center y Arbitration Register 1 (C0MyAR1)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy3h	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13

CAN 0 Message Center y Arbitration Register 2 (C0MyAR2)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy4h	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5

CAN 0 Message Center y Arbitration Register 3 (C0MyAR3)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy5h	ID4	ID3	ID2	ID1	ID0	0	0	WTOE

ID28–ID0

CAN 0 message center y arbitration registers 0–3. These bits form the arbitration value/identification number for message center y. When the message center is configured in a transmit mode, these registers are the source of the 29-bit ID message field (when EX/$\overline{ST}$ = 1) or the 11-bit ID message field (when EX/$\overline{ST}$ = 0). When EX/$\overline{ST}$ = 1, the 29 message ID bits correspond to ID28–ID0 as shown above. When EX/$\overline{ST}$ = 0, the message ID bits 10–0 correspond to ID28–18 in C0MyAR0 and C0MyAR1.

When configured in a receive mode, these registers serve as the arbitration value for message center y, against which incoming messages are compared to ascertain if they are valid for that message center. When EX/$\overline{ST}$ = 1, all 29 bits of the arbitration are used, but when EX/$\overline{ST}$ = 0, only the most significant 11 bits are used.

Note that when a message is successfully loaded, the entire message is loaded to the message center. So, if message ID masking was enabled (MEME = 1), it is possible to overwrite the arbitration register bits that were defined as "don't care" for incoming message acceptance.

Bits 2-1 (C0MyAR3 only)

Reserved. Bits 2 and 1 of the C0MyAR3 register are not used in arbitration. A read of these bits always return 0, and writes to these bits are ignored.

WTOE
Bit 0 (C0MxAR3 only)

Write-over enable. This bit controls the ability of a new message to overwrite an existing message in the corresponding message center in receive mode. The DTUP and EXTRQ bits for the message center in question must also be considered to determine the effect of this bit as follows. The WTOE bit can only be programmed when the SWINT bit is set.

WTOE	DTUP	EXTRQ	RESULT WHEN NEW MESSAGE DETECTED
0	0	0	There is currently no unread message or pending external frame request in the message center, so the matching message is written to appropriate message center (1–15).
0	1	x	The message center (1–15) has an unread message or pending external frame request. The incoming matching message is ignored and the message center remains unchanged. The CAN module proceeds to the next lower-priority message center to evaluate the incoming message ID and arbitration bits and related masking operations. (No overwrite.)
0	x	1	The message center (1–15) has an unread message or pending external frame request. The incoming matching message is ignored and the message center remains unchanged. The CAN module proceeds to the next lower-priority message center to evaluate the incoming message ID and arbitration bits and related masking operations. (No overwrite.)
1	0	x	There is currently no unread message or pending external request in the message center, so the matching message is written to appropriate message center (1–15).
1	1	x	The new matching message is stored, overwriting the previously stored message. The ROW bit is set to indicate the overwrite operation.

Special notes for message center 15. The ROW bit in message center 15 is associated with an overwrite of the shadow buffer for message center 15. The EXTRQ and DTUP bits are also shadow buffered to allow the buffered message and the message center 15 value to take on different relationships. The EXTRQ and DTUP values read by software are the current message center 15 values, rather than those of the shadow buffer, as is the case with the ROW bit. The shadow buffer is automatically loaded into message center 15 when both the DTUP bit and EXTRQ bit are cleared. If either DTUP = 1 or EXTRQ = 1 when clearing the other, any message in the shadow buffer is not transferred to the message 15 registers, and any incoming messages for message 15 are stored in the shadow buffer if WTOE = 1, or are lost if WTOE = 0.

Special notes concerning remote frames. For remote frames, which can be received by transmit message centers (1–14) in case of a matching identifier, WTOE and EXTRQ are evaluated. If (WTOE = 1) OR (WTOE = 0 and EXTRQ = 1), the respective transmit message center (1–14) arbitration bits can be overwritten.

CAN 0 Message Center y Format Register (C0MyF)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy6h	DTBYC3	DTBYC2	DTBYC1	DTBYC0	T/$\overline{R}$	EX/$\overline{ST}$	MEME	MDME

DTBYC3–0
Bits 7–4

Data byte count. These bits indicate the number of bytes within the data field of the message. When performing a transmit, software sets the DTBYC bits to establish the number of bytes that are to be transmitted. When receiving a message, the DTBYC bits indicate the (binary) number of bytes of data in the incoming message (i.e., 0000b = 0 data bytes and 1000b = 8 data bytes).

T/$\overline{R}$
Bit 3

Transmit/receive select. This bit is programmed by the application software to indicate if the message is to be transmitted (T/$\overline{R}$ = 1) or received (T/$\overline{R}$ = 0). This bit can only be modified when MSRDY = 0. This bit does not exist for message center 15, and always returns 0 when read from message center 15.

EX/$\overline{ST}$
Bit 2

Extended or standard identifier. This bit determines whether the respective message is to utilize the extended 29-bit identification format (EX/$\overline{ST}$ = 1) or the standard 11-bit identification format (EX/$\overline{ST}$ = 0). Message centers programmed for only one format receive/send extended messages in that format and ignore the alternate format. This bit can only be modified when MSRDY = 0.

MEME
Bit 1

Message identification mask enable. The MEME bit enables (MEME = 1) or disables (MEME = 0) the use of the message identification masking process, associated with the testing of the identifi

cation field in the incoming message. This bit can only be modified when MSRDY = 0.

0 = The mask registers are ignored when evaluating the identification bits of the incoming message, and the identification bits of the incoming message and the message center arbitration bits must match exactly to allow receipt of the incoming message. This is equivalent to programming the mask with all zeros. An exact match is also required before a remote data request is allowed.

1 = The mask registers are enabled, comparing only those bits message identification and arbitration bits that correspond to a 1 in the mask register. Since the entire message is loaded on a successful ID match, note that it is possible to overwrite the corresponding arbitration register bits that were defined as "don't cares" (0) in the standard or extended global ID mask.

MDME
Bit 0

Media identification mask enable. The MDME bit enables (MDME = 1) or disables (MDME = 0) the use of the first 2 bytes of the data field as a message qualifier. This bit can be only modified when MSRDY = 0.

0 = The first 2 bytes of the data field are ignored and not compared.

1 = The first 2 data bytes are masked by the respective media mask ID register and then compared with the media arbitration register 0 and 1 bytes. Only those bits in the first 2 data bytes and the arbitration registers corresponding to a 1 in the mask register are compared. When MDME = 1 the test is also performed before a remote request of data from a remote node is accepted.

CAN 0 Message Center y Data Byte 0 (C0MyD0)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy7h								

CAN 0 Message Center y Data Byte 1 (C0MyD1)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy8h								

CAN 0 Message Center y Data Byte 2 (C0MyD2)

MOVX Address[1]	7	6	5	4	3	2	1	0
Xxxxy9h								

CAN 0 Message Center y Data Byte 3 (C0MyD3)

MOVX Address[1]	7	6	5	4	3	2	1	0
XxxxyAh								

CAN 0 Message Center y Data Byte 4 (C0MyD4)

MOVX Address[1]	7	6	5	4	3	2	1	0
XxxxyBh								

CAN 0 Message Center y Data Byte 5 (C0MyD5)

MOVX Address[1]	7	6	5	4	3	2	1	0
XxxxyCh								

CAN 0 Message Center y Data Byte 6 (C0MyD6)

MOVX Address[1]	7	6	5	4	3	2	1	0
XxxxyDh								

CAN 0 Message Center y Data Byte 7 (C0MyD7)

MOVX Address[1]	7	6	5	4	3	2	1	0
XxxxyEh								

C0MyD0–C0MyD7 **CAN 0 message center y data bytes 0–7.** These bytes hold data to be transmitted or received data.

Frame Types

The CAN 2.0B protocol specifies two different message formats, the standard 11-bit (CAN 2.0A) and the extended 29-bit (CAN 2.0B), and four different frame types for CAN bus communications.

The standard format makes use of an 11-bit identifier, as follows.

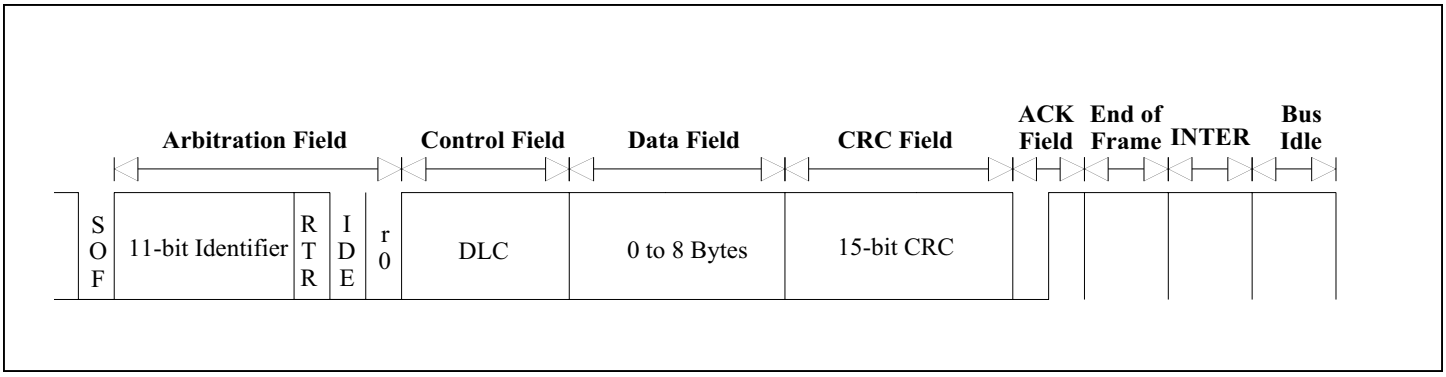


Figure 19-1. CAN 2.0A (Standard) Format

The following extended format makes use of a 29-bit identifier.

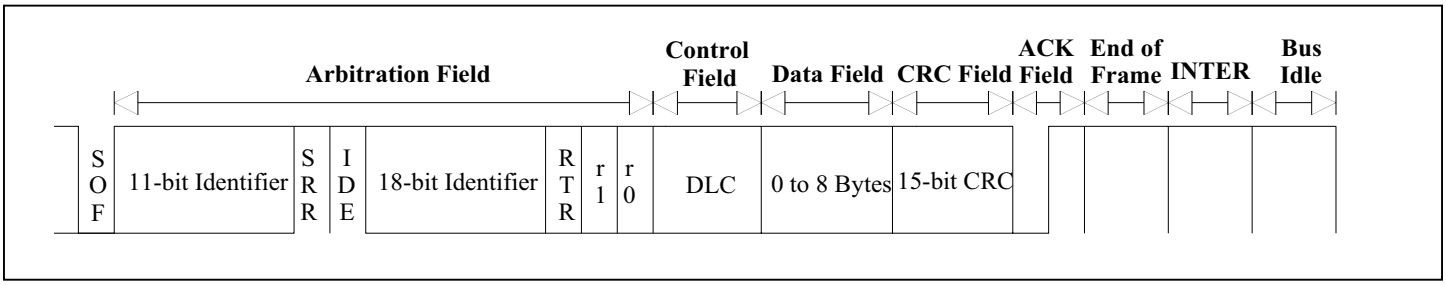


Figure 19-2. CAN 2.0B (Extended) Format

The four frame types for CAN bus communications are the data frame, the remote frame, the error frame, and the overload frame.

Data frame:

The data frame is formulated to carry data from a transmitter to a receiver. The preceding two figures are examples of data frames in the standard and extended formats. The data frame is composed of seven fields. These are the start of frame, arbitration field, control field, data field, CRC field, acknowledge field, and end of frame. A description of these fields follows.

Start of frame (SOF): (Standard and extended format)

The start of frame (SOF) is a dominant bit, which signals the start of a data or remote frame. The dominant bit forces a hard synchronization, initiating the CAN controller receive mode.

Arbitration field: (Standard and extended format)

The arbitration field contains the identifier of the message and a dominant remote request (RTR) bit. The identifier is composed of one field in the standard 11-bit format or two fields in the extended 29-bit format. Two additional bits, the substitution remote request (SRR) bit and the identifier extension (IDE) bit, separate the two fields in the extended format.

Remote request (RTR) bit: (Standard and extended format)

The remote request (RTR) bit is a dominant bit in data frames and a recessive bit in remote frames.

Substitution remote request (SRR) bit: (Extended format)

The substitution remote request (SRR) bit is a recessive bit and is substituted for the RTR bit when using the extended format.

Identifier extension (IDE) bit: (Extended format)

The identifier extension (IDE) bit is a dominant bit in the standard format and a recessive bit in the extended format. The IDE bit is located in the arbitration field in the standard format and is located in the control field in the extended format.

Control field: (Standard and extended format)

The control field is made up of 6 bits in two fields. The first field is made up of 2 reserved bits, which are transmitted as dominant bits. The second field contains 4 bits, which make up the data length code (DLC). The DLC determines the number of data bytes in the data field of the data frame, and is programmed through the use of the CAN message format registers, located in each of the 15 message centers.

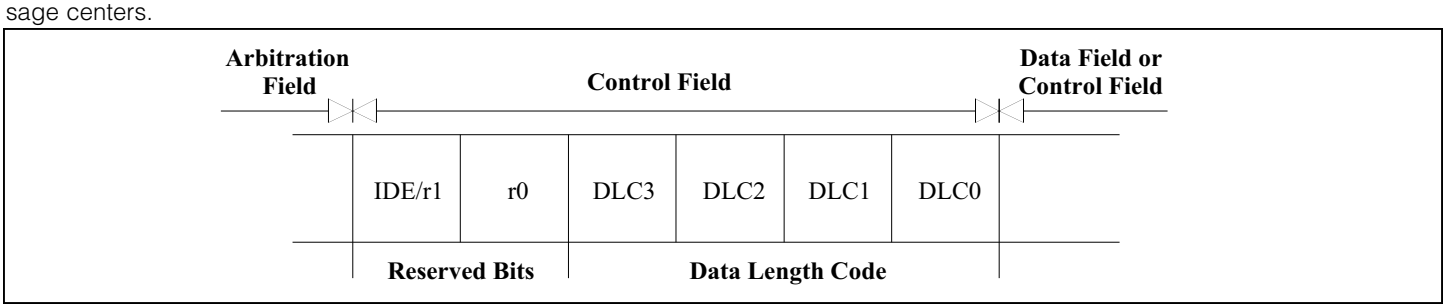


Figure 19-3. Control Field

Data field: (Standard and extended format)

The data field is made up of 0 to 8 bytes in a data frame and 0 bytes in a remote frame. The number of data bytes associated with a message center is programmed through the use of the CAN message format registers, located in each of the 15 message centers. The data field contents are saved to the respective message center. If the identifier test is successful, no errors are detected through the last bit of the end of frame, and an error frame does not immediately follow the data or remote frame. The data field is transmitted least significant byte first, with the most significant bit value of each byte transmitted first.

CRC field: (Standard and extended format)

The CRC field is made up of a 15-bit code, which is the computed cyclic redundancy check (after destuffing bits) from the start of frame, through the arbitration, control, data fields (when present), and a CRC delimiter. The CRC calculation is limited to 127-bit maximum code word (a shortened BCH code) with a CRC sequence length of 15 bits.

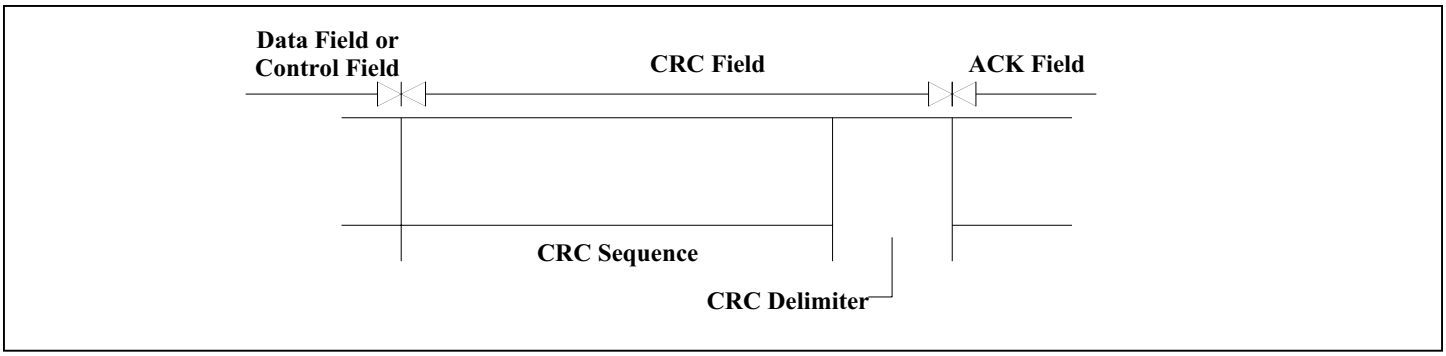


Figure 19-4. CRC Field

Acknowledge (ACK) field: (Standard and extended format)

The acknowledge (ACK) field is made up of 2 bits. The transmitting node sends 2 recessive bits in the ACK field. The receiving nodes, which have received the message and found the CRC sequence to be correct, reply by driving the ACK slot with a dominant bit. The ACK delimiter is always a recessive bit.

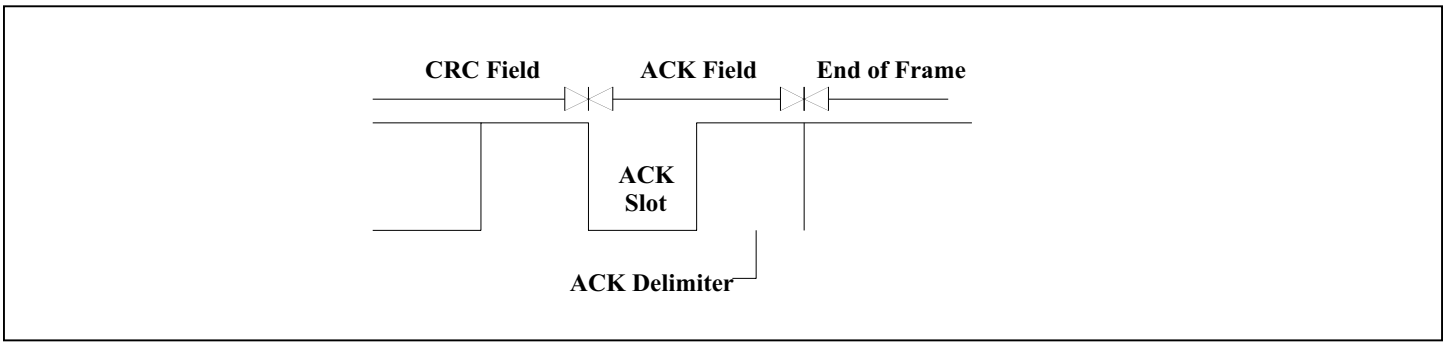


Figure 19-5. Acknowledge Field

End of frame: (Standard and extended format)

The end of frame for both the data and remote frame is established by the transmitter sending 7 recessive bits.

Interframe spacing (intermission): (Standard and extended format.)

Data frames and remote frames are separated from preceding frames by three recessive bits termed the intermission. During the intermission, the only allowed signaling to the bus is by an overload condition. No node is allowed to start a message transmission of a data or remote frame during this period. If no node becomes active following the interframe space, an indeterminate number of recessive bit times transpire in the bus idle condition until the next transmission of a new data or remote frame by a node.

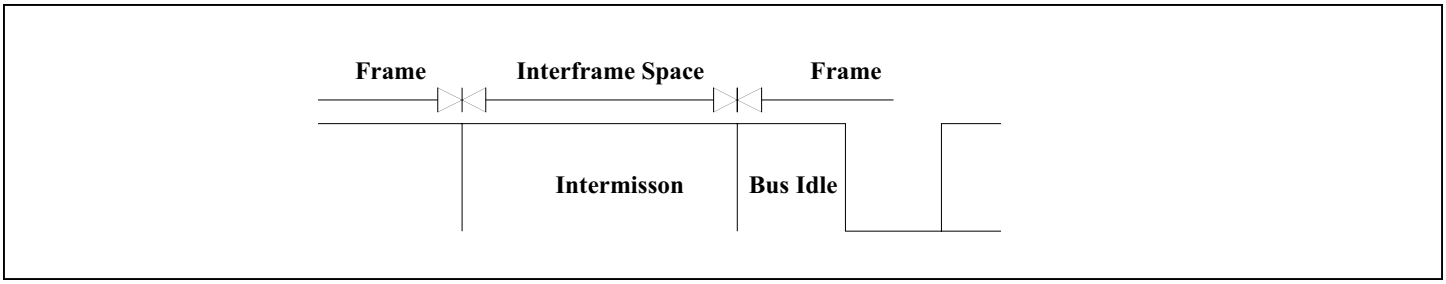


Figure 19-6. Intermission

Remote frame: (Standard and extended format)

The remote frame is transmitted by a CAN controller to request the transmission of the data frame with the same identifier. The remote frame is composed of seven fields. These are the start of frame, arbitration field, control field, data field, CRC field, acknowledge field, and an end of frame.

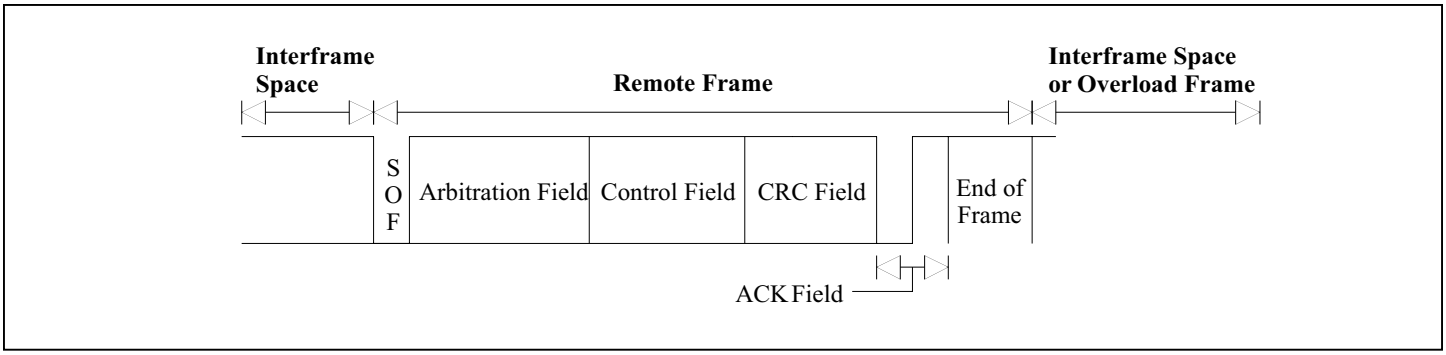


Figure 19-7. Remote Frame

The remote frame is used when a CAN processor wishes to request data from another node. Sending a remote frame initiates a transmission of data from a source node with the same identifier (masked groups included). The primary bit pattern difference between a data frame and a remote frame is the RTR bit, which is sent as a recessive bit in the remote frame, and is sent as a dominant bit in the data frame. The remote frame also does not contain a data field, being independent of the programmed values in the DTBYC3–DTBYC0 bits in the respective CAN message format register.

Error frame:

The error frame is transmitted by a CAN controller when the CAN processor detects a bus error. The error frame is composed of two different fields. These are 1) the superposition of the error flags from different nodes and 2) the error delimiter.

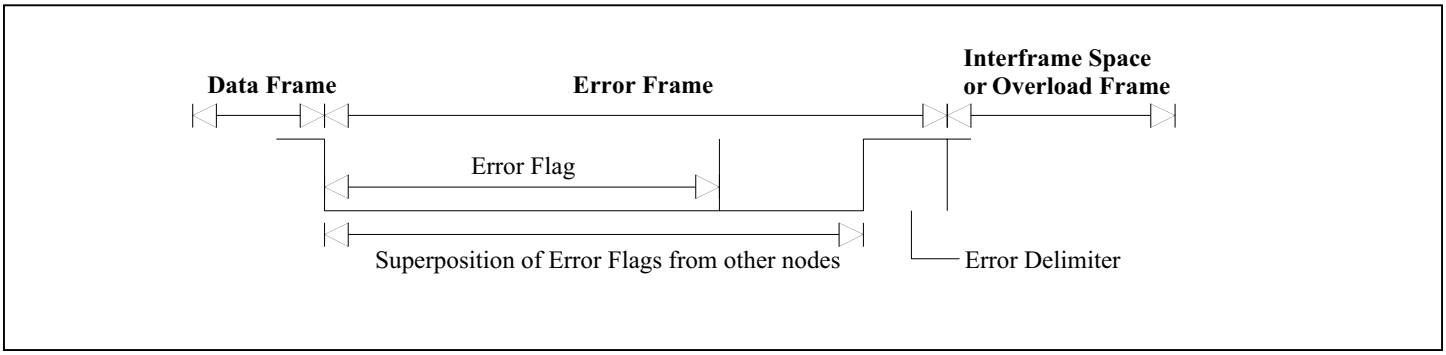


Figure 19-8. Error Frame

The error frame is composed of 6 dominant bits, which violates the CAN specification bit-stuffing rule. If either of the CAN processors detects an error condition, that CAN processor transmits an error frame. When this happens, all nodes on the bus detect the bit stuff error condition and transmit their own error frame. The superpositioning of all of these error frames leads to a total error frame length between 6 and 12 bits, depending on the response time and number of nodes in the system. Any messages (data or remote frame) received by the CAN processors (successful or not) that are followed by an error frame are discarded. After the transmission of an error flag, each CAN processor sends an error delimiter (8 recessive bits) and monitors the bus until it detects the change from the dominant to recessive bit level. The CAN modules issues an error frame each time an error frame is detected. Following a series of error frames, the CAN modules enter into an error-passive mode. In the error-passive mode, the CAN processors transmit 6 recessive bits, and wait until 6 equal bits of the same polarity have been detected. At this point, the CAN processor begins the next internal receive or transmission operation.

Overload frame:

The overload frame provides an extra delay between data or remote frames. The overload frame is composed of two fields: the overload flag and the overload delimiter.

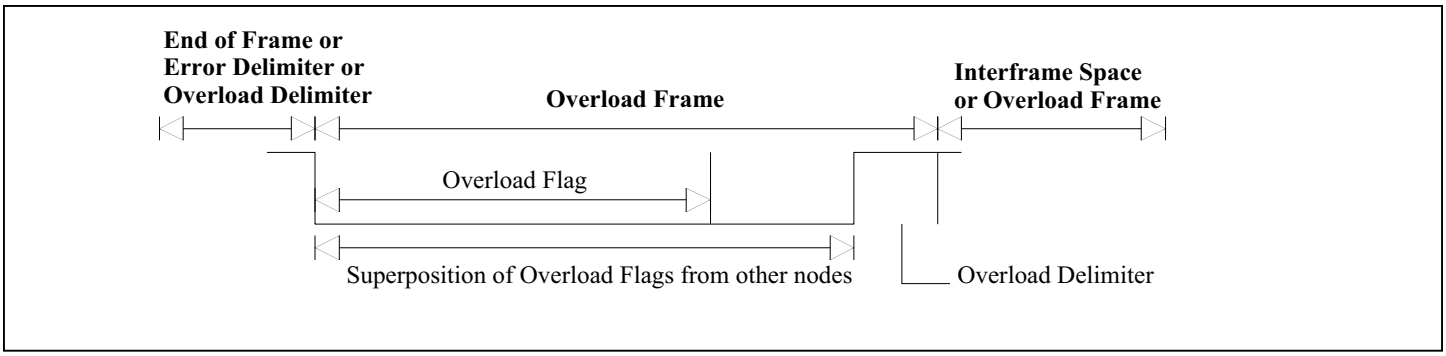


Figure 19-9. Overload Frame

Three conditions lead to the transmission of an overload flag:

1) The internal conditions of a CAN receiver require a delay before the next data or remote frame is sent. The DS80C400 CAN con

troller is designed to prevent this condition for data rates at or below the 1MB per second data rate.

2) The CAN processor detects a dominant bit at the first and second bit positions of the intermission.

3) If the CAN processor detects a dominant bit at the eighth bit of an error delimiter or overload delimiter, it starts transmitting an over load frame.

The error counters are not incremented as a result of number 3. The CAN processor starts an overload frame at the first bit of an expected intermission only if initiated by condition 1. Conditions 2 and 3 result in the CAN processor transmitting an overload frame starting one bit after detecting the dominant bit. The overload flag consists of 6 dominant bits that correspond to an error flag. Because the overload frame is only transmitted at the first bit time of the interframe space, it is possible for the CAN processor to discriminate between an error frame and an overload frame. The overload flag destroys the intermission field. When such a condition is detected, the CAN processor detects the overload condition and begins transmitting an overload frame. After the transmission of an overload frame, the CAN processors monitor the bus for a dominant to recessive level change. The CAN processor then begins the transmission of 6 additional recessive bits, for a total of 7 recessive bits on the bus. The overload delimiter consists of 8 recessive bits.

Initializing the CAN Controller

Software initialization of the CAN controller begins with the setting of the software initialization bit (SWINT) in the CAN 0 control SFR register. When SWINT = 1, the CAN module is disabled and the CAN transmit output (C0TX) is placed in a recessive state. This, in turn, allows the microcontroller to write information into the CAN MOVX SRAM control/status/mask registers without the possibility of corrupting data transmissions or receptions in progress. Setting SWINT does not clear the receive- and transmit-error counters, but does allow the microcontroller to write a common value to both error counters through the CAN 0 transmit-error SFR register. Consult the description of the SWINT bit for specifics of the software initialization process.

All CAN registers located in the SFR memory map, with the exception of the CAN 0 control register, are cleared to a 00 hex following a system reset. The CAN 0 control register is set to 07 hex following a system reset. CAN registers located in the MOVX memory map are indeterminate following a system reset. A system reset also clears both the receive- and transmit-error counters in the CAN controller, takes the CAN processor off line, and sets the SWINT bit in the CAN 0 control register.

Following a reset, the following CAN related registers must be initialized for proper operation of the CAN module. These registers are in addition to specific registers associated with mask, format, or specific message centers.

REGISTER	SIGNIFICANCE
P5CNT (SFR A2h)	C0_I/O (P5CNT.3) must be set to enable CAN 0 pins P5.1 and P5.0.
C0BT0, C0BT1 (MOVX SRAM xxxx04-5)	These MOVX SRAM control registers must be set to configure CAN 0 (C0BT0, C0BT1) bus timing. The exact values are dependent on the network configuration and environment.
COR (SFR CEh)	C0BPR7-6 (COR.4-3) must be configured as part of the CAN 0 bus timing.

CAN Interrupts

The CAN processor is assigned an interrupt that is individually enabled by the C0IE bit in the EIE register and globally enabled/disabled by the EA bit in the IE SFR register. A CAN 0 interrupt can be generated by either a receive/transmit acknowledgment from one of the 15 message centers or by a change in the CAN 0 status register.

CAN 0 transmit/receive interrupt sources are derived from a successful transmit or receive of data within one of the 15 message centers, as signaled by the INTRQ bit in the associated CAN 0 message (1–15) control register. Each message center (1–15) provides a separate receive interrupt enable (ERI) and transmit interrupt enable (ETI) bits in the respective CAN 0 message (1–15) control register to allow setting of the INTRQ bit in response to successful transmission or reception. The CAN 0 interrupt register (C0IR; A5h) SFR can then be used to determine which message center generated the interrupt request. Software must clear the respective INTRQ bit in the associated CAN 0 message (1–15) control register in order to clear the interrupt source before leaving the interrupt routine.

The CAN 0 interrupt source can also be connected to a change in the CAN 0 status register. Each of the bits in the CAN 0 status register represents a potential source for the interrupt. To simplify the application and testing of a device, these sources are broken into two groups that, for interrupt purposes, are enabled separately by the ERIE and STIE bits of the CAN 0 control (C0C) register. This allows the nonstandard errors typically associated with development to be grouped under the STIE enable. These include the successful receive RXS, successful transmit TXS, wake status WKS, and general set of error conditions reported by ER2–ER0. Also note that, since the RXS and TXS bit are cleared by software, if a second message is received or transmitted before the RXS or TXS bits are cleared and, after a read of the CAN 0 status register, a second interrupt is generated. The remaining error sources comprise the BSS and EC96/128 bits in the CAN 0 status register. These read-only bits are separately enabled by the ERIE bit in the CAN 0 control register. A read of the CAN 0 status register is required to clear either of the two groups of error interrupts. It is possible that multiple

changes to the status register can occur before the register is read; in that case, the status register generates only one interrupt. Figure 19-10 provides a graphical illustration of the interrupt sources and their respective interrupt enables.

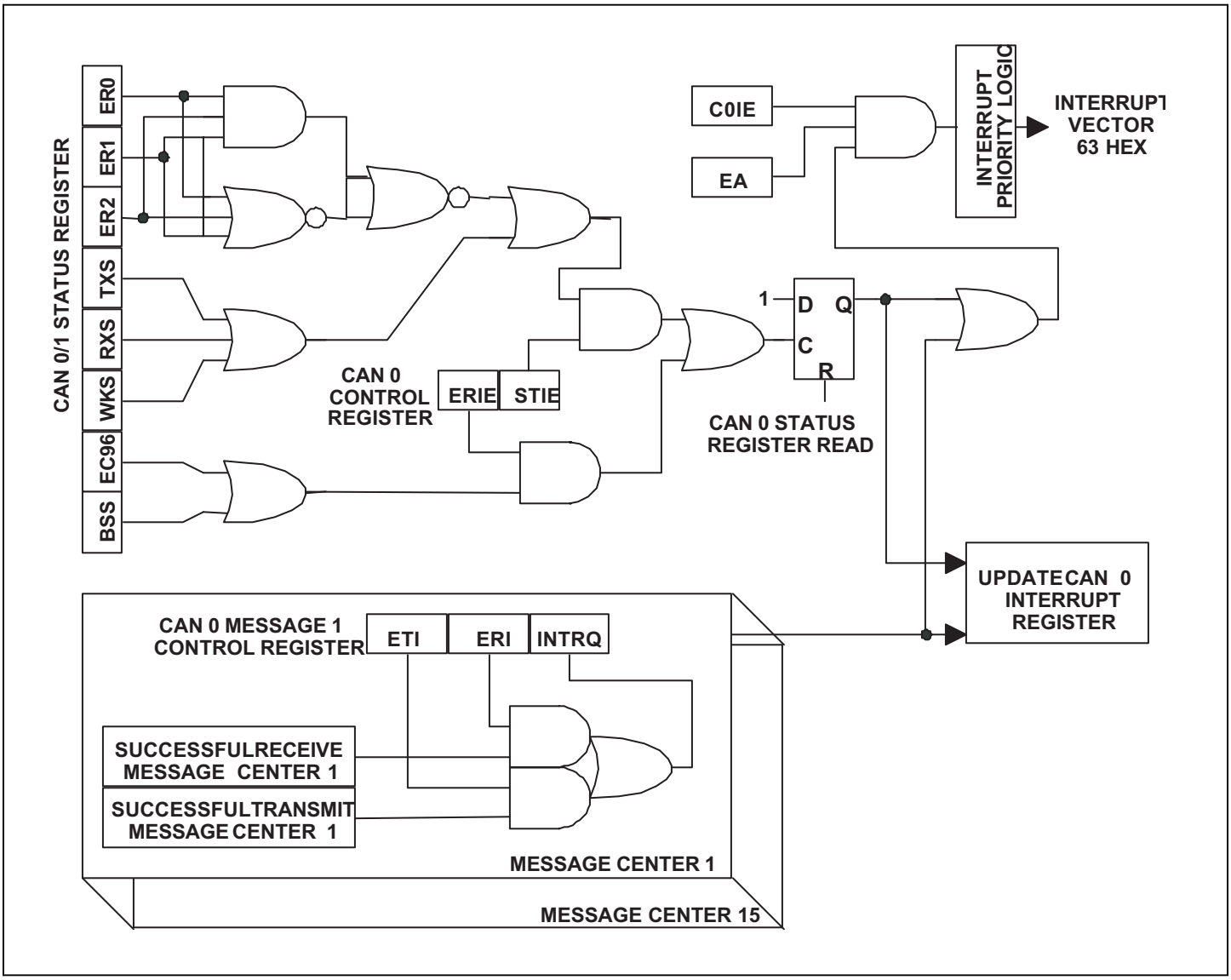


Figure 19-10. CAN Interrupt Logic

Arbitration/Masking Considerations

The CAN processor is designed to evaluate and determine if an incoming message is loaded into one of the 15 message centers. Acceptance of a message is determined by comparing the message's ID and/or data field against the corresponding arbitration information defined for each message center. Messages that contain bit errors or fail arbitration are discarded. The incoming message is tested in order against each enabled message center (enabled by the MSRDY bit in the CAN message control register) from 1 to 15. The first message center to successfully pass the test receives the incoming message and ends the testing, and the message is loaded into the respective message center.

The DS80C400 CAN module supports two types of arbitration: basic and media. Basic arbitration compares either the 29-bit (EX/$\overline{ST}$ = 1) or 11-bit (EX/$\overline{ST}$ = 0) incoming message ID against the corresponding bits in the message center CAN arbitration registers (C0MxAR0–3). This depends upon whether the message center has been configured for 29-bit or 11-bit operation. An optional mask-

ing feature can also be utilized in conjunction with basic arbitration. The format register (C0MxF) for each message center contains a message identification mask enable bit (MEME). If MEME is set, the CAN module factors in the standard global mask registers (C0SGM0–1) when EX/$\overline{ST}$ = 0, or the extended global mask registers (C0EGM0–3) when EX/$\overline{ST}$ = 1, when deciding if there is an ID match. A comparison between bits in the incoming message ID and arbitration register bits is only made for bit positions that correspond to a 1 in the appropriate mask register. Bits corresponding to 0 in the mask register are ignored, creating a "don't care" condition. Filling the mask register with all 0s while MEME = 1 causes the arbitration circuitry to automatically match all message IDs. Filling the mask register with all 1s while MEME = 1 requires an exact match between the incoming message ID and the arbitration registers, just as is the case when MEME = 0.

Media arbitration is an optional second arbitration performed if the media identification mask enable bit (MDME) is set in the C0MxF message center register. Media arbitration compares the first and second byte of the data field in each incoming message against two 8-bit media arbitration bytes (stored at locations C0MA0, C0MA1). Each media arbitration byte has an associated media identification mask: C0MID0 for C0MA0 and C0MID1 for C0MA1. Media byte comparison is made only for those bits corresponding to a 1 in the media identification mask registers. When MDME = 1, the first two data bytes of the incoming message must pass media byte arbitration as defined by C0MA0:1 and C0MID0:1 before being loaded into the respective message center. However, unlike the identification mask enable (MEME), when MDME = 0, no testing is performed of the first two bytes of the incoming data field.

MESSAGE CENTER 15

Message center 15 supports an additional set of masks to supplement basic arbitration. While this message center performs basic and media arbitration as per message centers 1–14, it also uses the C015M3–0 mask registers to perform an additional level of filtering during basic (i.e., not media) arbitration. When determining arbitration for message center 15, the contents of C015M3–0 are logically ANDed with either C0EGM3–0 (if EX/$\overline{ST}$ = 1 for message center 15) or C0SGM1-0 (if EX/$\overline{ST}$ = 0 for message center 15). This ANDed value is then used in place of C0EGM3–0 or C0SGM1-0 when performing basic arbitration as described previously. If the MDME bit is set, then the incoming message must also pass the media arbitration test.

Message center 15 has a buffered FIFO arrangement to allow up to two received messages to be received without being lost prior to the microcontroller reading of the first message. The first message received by message center 15 is stored in the normal MOVX memory location for message center 15, if the previous message has been already read by the microcontroller. If the first message has not been read, then the incoming message is buffered internally until the first message is read, at which time the second message is automatically loaded into the first (MOVX) message 15 slot, allowing software to then read the second message. The CAN module determines that the first message has been read (allowing the buffered message to be transferred) when software clears the DTUP and EXTRQ bits. If a third message arrives before the second message has been copied into the MOVX message 15 slot, then the third message writes over the second buffered message. Software should clear the INTRQ bit, as well as the DTUP and EXTRQ bits, after reading each message in the MOVX message 15 center. The WTOE bit associated with message center 15 has unique operating considerations, described later in the section.

Transmitting and Receiving Messages

All CAN data is sent and received through message centers. All CAN message centers are identical, with the exception of message center 15. Message center 15 has been designed as a receive-only center and is shadow-buffered to help prevent the loss of incoming messages when software is unable to read one message before the next one should be loaded. All message centers, with the exception of message center 15, are capable of four operations:

- Transmitting a data message
- Receiving a data message
- Transmitting a remote frame request
- Receiving a remote frame request

Transmitting Data Messages

Starting with the lowest-numbered message center (highest priority), each CAN module sequentially scans each message center until it finds a message center that is proper enabled for transmission (T/$\overline{R}$ = 1, TIH = 0, DTUP = 1, MSRDY = 1, and MTRQ = 1). The contents of the respective message center are then transferred to the transmit buffer, and the CAN module attempts to transmit the message. If successful, the appropriate MTRQ bit is cleared to 0, indicating that the message was successfully sent. Following a successful transmission, loss of arbitration, or an error condition, the CAN module again searches for a properly configured message center, starting with the lowest-numbered message center. This search relationship always allows the highest priority message center to be transmitted independent of the last successful (MTRQ = 0) or unsuccessful (MTRQ = 1) message transmission.

Receiving Data Messages

Each incoming data message is compared sequentially with each receive enabled (T/$\overline{R}$ = 0) message center starting with the lowest-numbered message center (highest priority) and proceeding to the highest-numbered message center. This testing continues until a match is found (incorporating masking functions as required), at which time the incoming message is stored in the respective message center. Higher numbered message centers that are not reviewed prior to the match are not evaluated during the current message test. When the WTOE = 1, the CAN module can overwrite receive message centers that have DTUP = 1, which, in turn, sets ROW = 1. When WTOE = 0, incoming messages do not overwrite receive message centers that have DTUP = 1.

Message center 15 is a special receive-only, FIFO-buffered message center designed to receive messages not accepted by the other message centers. The ROW bit in message center 15 is associated with the overwrite of the shadow buffer for message center 15. The EXTRQ and DTUP bits are shadow-buffered to allow the buffered message and the message center 15 values to take on different relationships. The EXTRQ and DTUP values read by the microcontroller are not those of the shadow buffer, as is the case with the ROW bit, but are the current values associated with message center 15. The shadow buffer is automatically loaded into message center 15 when both the DTUP bit and the EXTRQ bit are cleared. If either DTUP or EXTRQ are left set when clearing the other, any message in the shadow buffer is not transferred to the message 15 registers, and any incoming messages for message 15 are stored in the shadow buffer (if WTOE = 1) or are lost (if WTOE = 0).

Transmitting Remote Frame Requests

Starting with the lowest-numbered message center (highest priority), each CAN module sequentially scans each message center. When it finds a message center properly enabled to transmit a remote frame (T/$\overline{R}$ = 0, MSRDY = 1, and MTRQ = 1), the contents of the respective message center is then transferred to the transmit buffer and the CAN module attempts to transmit the message. If successful, the appropriate MTRQ bit is cleared to 0, indicating that the message was successfully sent. Following a successful transmission, loss of arbitration, or an error condition, the CAN module again searches for a properly configured message center, starting with the lowest-numbered message center. This search relationship always allows the highest priority message center to be transmitted, independent of the last successful (MTRQ = 0) or unsuccessful (MTRQ = 1) message transmission. The state of the TIH bit does not affect the transmission of a remote frame request.

Receiving/Responding to Remote Frame Requests

The remote frame request is handled like a data frame with data length zero and the EXTRQ and RXS bits are set. Each incoming remote frame request (RFR) message is compared sequentially with each enabled (MSRDY = 1) message center starting with the lowest-numbered message center (highest priority) and proceeding to the highest-numbered message center. Testing continues until a match is found (incorporating masking functions as required), at which time the incoming RFR message is stored in the respective message center, the DTBYC bits are updated to indicate the requested number of return bytes (DTBYC = 0 for a remote frame request), and EXTRQ and MTRQ are both set to 1. When the message is successfully received and stored, an interrupt of the corresponding message center is asserted, if enabled by the ERI bit. The EXTRQ bit can be left set if the message center is reconfigured to perform a transmit (T/$\overline{R}$ = 1) and used in the standard reply of a remote frame operating with transmit message centers. EXTRQ can also be cleared by software if the current message center is not being used to reply to the remote frame request. Higher-numbered message centers (lower priority) that are not reviewed prior to the match are not evaluated during the current message test. Depending on the state of the transmit/receive bit for that message center, the CAN module performs one of two responses.

If the microcontroller wants to request data from another node, it first clears the respective MSRDY bit to 0 and then writes the identifier and control bits in this message center, configures the message center as a receive message center (T/$\overline{R}$ = 0), and then sets the MTRQ bit. After a successful transmission, the CAN module clears MTRQ = 0 and sets TXS = 1. In addition to the TXS bit, if the ETI bit is set, the successful transmission also sets the corresponding INTRQ bit. Requesting data from another node is possible in message centers 1–14. As seen above, the CAN module sends a remote frame request and receives the data frame in the same message center that sent the request. Therefore, only one mailbox is necessary to do a remote request.

When a remote frame request is successfully received, message centers enabled for transmission (T/$\overline{R}$ = 1) set the EXTRQ and MTRQ bits in the corresponding message center to mark the message as a 'to be sent' message. The CAN module attempts to automatically transmit the requested message if the message center is fully enabled to do so (MSRDY = 1, TIH = 0, DTUP = 1). After the transmission, the TXS bit in the status register is set, the EXTRQ and MTRQ bits are reset to a 0, and a message center interrupt of the corresponding message center is asserted, if enabled by the respective ETI bit. If the transmit inhibit bit is set (TIH = 1), the message center receives the RFR, modifying the DTBYC and/or arbitration bits if necessary, but the return data is not transmitted until TIH = 0.

If software wants to modify the data in a message center configured for transmission of an answer to a remote request (EXTRQ set to a 1), the microcontroller must set the TIH = 1 and DTUP = 0. The microcontroller can then access the data byte registers 0–7, data byte count (DTBYC3–0), the extended or standard mode bit (EX/$\overline{ST}$), and the mask enables (MEME and MDME) of the message center to load the required settings. Following the setup, the software should reset TIH to a 0 and sets DTUP to a 1 bit to signal the CAN that the access is finished. Until the DTUP = 1 and TIH = 0, the transmission of this mailbox is not permitted. Thus, the CAN transmits

the newest data and resets EXTRQ = 0 after the transmission is complete. The message center must first be disabled to change the identifier or the direction control (T/$\overline{R}$).

Message centers enabled for reception (T/$\overline{R}$ = 0) do not automatically transmit the requested data. The remote frame request does, however, continue to store the requested number of return bytes in DTBYC and set EXTRQ = 1. No data bytes are received or stored from a remote frame request. The message center can then be configured by software to function as transmitter (T/$\overline{R}$ = 1) and transmit the requested data, or the microcontroller can configure another message center in a transmit mode (T/$\overline{R}$ = 1) to send the requested data. Note that, when T/$\overline{R}$ = 0, the MTRQ bit is not set upon loading of a matching remote frame request.

When a remote frame is received, the CAN module can be configured to either automatically transmit data back to the remote node or allow the microcontroller to intervene and establish the conditions for the transmission of the return message. The following examples outline various options to respond to remote frame requests.

Case 1: **Automatic reply.**
CAN controller receives a remote frame request (RFR) and automatically transmits data without additional software intervention.

1. Software sets T/$\overline{R}$= 1, MSRDY = 0, DTUP = 0, and TIH = 1.
2. Software loads data into respective message center.
3. Software sets MSRDY = 1, DTUP = 1, and TIH = 0 in same instruction.
   Note: Software does not change MTRQ = 0 from previously completed transmission
4. CAN does not transmit data (MTRQ = 0), but waits for RFR.
5. CAN successfully receives RFR.
6. CAN forces MTRQ = 1 and EXTRQ = 1.
7. CAN loads DTBYC from RFR and ID into arbitration registers.
8. CAN automatically transmits data in respective message center.
9. CAN clears EXTRQ = 0 and MTRQ = 0.

Case 2: **Software-initiated reply.** (Using TIH as gating control.)
CAN module wants to receive an RFR and wait for software to determine when and what is transmitted in reference to RFR.

1. Software sets T/$\overline{R}$ = 1, MSRDY = 0, DTUP = 0, and TIH = 1.
2. Software loads data into respective message center.
3. Software sets MSRDY = 1, DTUP = 1, and TIH = 1 in same instruction.
   Note: Software does not change MTRQ = 0 from previous completed transmission
4. CAN does not transmit data (MTRQ = 0), but waits for RFR.
5. CAN successfully receives RFR.
6. CAN forces MTRQ = 1 and EXTRQ = 1.
7. CAN loads DTBYC from RFR and ID into arbitration registers.
8. CAN waits for software to read message center and determine the fact that EXTRQ = 1.
9. Software can load data into message center (or it can already have the data established).
10. Software writes MSRDY = 1, DTUP = 1, and TIH = 0 in same instruction.
11. CAN automatically transmits data (as per RFR DTBYC) in respective message center.
12. CAN clears EXTRQ = 0 and MTRQ = 0.

Case 3: **Software-initiated reply**. (Reply through same message center, using TIH as gating control.)
CAN module wants to receive an RFR in a receive-configured (T/$\overline{R}$ = 0) message center. When the data is received, the message center is reconfigured send data back to the remote request node. This relationship is not possible for message center 15.

1. Software sets T/$\overline{R}$ = 0, MSRDY = 1, and DTUP = 0 and awaits either data frame or RFR.
   Note: Software does not change MTRQ = 0 from previous completed transmission.
2. CAN successfully receives RFR.
3. CAN forces EXTRQ = 1 and DTUP = 1.
4. MTRQ cannot be written to a 1 by the CAN when T/$\overline{R}$ = 0 and is left as MTRQ = 0.
5. CAN loads DTBYC from RFR and ID into arbitration registers.
6. CAN waits for software to read message center and to determine the fact that EXTRQ = 1.
7. Software disables message center and converts message center into transmit message center.

   Software clears MSRDY = 0 to disable message center. Software leaves EXTRQ = 1.

   Software forces message center to transmit mode, T/$\overline{R}$ = 1.
8. Software writes MSRDY = 0, DTUP = 0, and TIH = 1 in preparation to load data.
9. Software loads data into message center.
10. Software writes MSRDY = 1, MTRQ = 1, DTUP = 1, and TIH = 0 in same instruction.

    Note that software leaves EXTRQ = 1.
11. CAN automatically transmits data (as per RFR DTBYC) into the respective message center.
12. CAN clears EXTRQ = 0 and MTRQ = 0.

Case 4: **Software-initiated reply**. (Reply through different a message center, using TIH as a gating control.) CAN controller wants to receive an RFR in a message center (denoted MC1) configured to receive data (T/$\overline{R}$ = 0) and to wait for soft ware to select another message center (denoted MC2) to send data back to remote request node.

1. Software sets T/$\overline{R}$ = 0, MSRDY = 1, and DTUP = 0 in MC1 and awaits either data frame or RFR.
   Note: Software does not change MTRQ = 0 in MC1 from previously completed transmission.
2. CAN successfully receives RFR in MC1.
3. CAN forces EXTRQ = 1 and DTUP = 1 in MC1.
   MTRQ cannot be written to a 1 by the CAN when T/$\overline{R}$ = 0 and is left as MTRQ = 0.
4. CAN loads DTBYC from RFR and ID into arbitration registers in MC1.
5. CAN waits for software to read message center and determine the fact that EXTRQ = 1.
6. Software disables MC1 to transfer information to MC2.

   Software clears MSRDY = 0 to disable MC1. Software leaves EXTRQ = 1.

   Software clears MSRDY = 0 in MC2.
7. Software forces MC2 to transmit mode T/$\overline{R}$ = 1.
8. Software loads ID and DTBYC value from MC1 into ID and from MC2 into DTBYC.
9. Software writes MSRDY = 0, DTUP = 0, and TIH = 1 in MC2 in preparation to load data to MC2.
10. Software loads data into MC2.
11. Software writes MSRDY = 1, MTRQ = 1, EXTRQ = 0, DTUP = 1, and TIH = 0 in MC2 in same instruction. Note that CAN has not set EXTRQ in MC2, and is not required to be set for transmission of data from MC2.
12. CAN automatically transmits data (as per RFR requested DTBYC) in MC2.
13. CAN clears MTRQ = 0 (leaving previous EXTRQ = 0 cleared).
14. Software sets T/$\overline{R}$ = 0, MSRDY = 1, EXTRQ = 0, and DTUP = 0 in MC1 and awaits either next RFR or date frame

    Note that MTRQ is still cleared in MC1, since MC1 has not been set to a transmit mode.

Remote Frame Handling in Relation to the DTBYC Bits

The DTBYC bits function slightly differently when remote frames are used. The data length code currently programmed in message

center is overwritten by the data length code field of the incoming remote request frame so that the requested number of data bytes can be sent in response to the remote request. The following example demonstrates how the DTBYC bits are modified by a received remote frame request.

1. Assume the microcontroller has programmed the following into a message center:
   DTBYC = 5, data field = 75 AF 43 2E 12 78 90 00
   (Note that only the first through the fifth data bytes are recognized because DTBYC = 5.)
2. When the CAN module successfully receives a remote frame with the following data:
   Identifier = ID, DTBYC = 2, RTR = 1.
3. The incoming message overwrites the identifier and the data length code. The new data in the message center is:
   DTBYC = 2, data field = 75 AF 43 2E 12 78 90 00
   (Note that only the first and second data bytes are recognized because DTBYC is now 2.)
4. The outgoing response is a data frame containing the following information:
   DTBYC = 2, data field = 75 AF

Important Information Concerning ID Changes when Awaiting Data from a Previous Remote Frame Request:

The use of acceptance filtering (MEME = 1) in conjunction with remote frame requests can result in a modification of the message center arbitration registers. Suppose that, for example, a message center is configured to transmit a remote frame request (MTRQ = 1, EXTRQ = 0, T/$\overline{R}$ = 0, and MSRDY = 1). If arbitration masks are used, it is possible for a second frame request from an external node to modify this requesting node's arbitration register value prior to reception of the previously requested data. When a remote frame request is received, the message ID is loaded into that message center's arbitration registers. When message identification masking is not used (MEME = 0), the message ID always matches the arbitration value, so the process is transparent. If masking is used, however, the message ID ANDed with the appropriate mask is loaded into that message center's arbitration registers, resulting in a change of the arbitration values for that message center. To prevent this situation, acceptance filtering should be disabled (MEME = 0) for any message center configured for remote frame handling.

Overwrite Enable/Disable Feature

The write-over enable bit (WTOE) located in each message center (C0MxAR3.0) enables or disables the overwriting of unread messages in message centers 1–15. Programming WTOE = 1 following a system reset or CRST bit-enabled reset allows newly received messages that pass arbitration to overwrite unread (i.e., message centers with DTUP = 1) messages. When an overwrite occurs, the receive overwrite (ROW) bit in the respective CAN message control register is set. When WTOE = 0, message centers that have data waiting to be read (indicated by DTUP = 1) or transmitted (EXTRQ = 1) are not overwritten by incoming data.

Special care must be taken when reading data from a message center with the overwrite feature enabled (WTOE = 1). Caution is needed because the WTOE bit, when set, allows an incoming message to overwrite the message center. If an overwrite occurs at the same time that software is attempting to read several bytes from the message center (such as a multi-byte data field), it is possible that the read could return a mix of information from the old and overwriting messages. To avoid this situation, software should clear the DTUP bit to 0 prior to reading the message center, and then verify afterwards that the DTUP bit remained at 0. If DTUP remains cleared after the read, no overwrite occurred and the returned data was correct. If DTUP = 1 after the read, software should again clear DTUP = 0 and reread the message center, since a possible overwrite has occurred. The original message is lost (as planned since WTOE = 1), but a new message should be available on the next read.

One important use of the WTOE bit is to allow the microcontroller to program multiple message centers with the same ID when operating in the receive mode, with WTOE = 0. This allows the CAN module to store multiple incoming messages in a series of message centers, creating a large storage area for high-speed recovery of large amounts of data. The CPU is required to manage the use of these message centers to keep track of the incoming data, but the use of multiple message centers and disabling of their overwrite (WTOE = 0) function prevents the module from potentially losing data during a high-speed data transfer. In transmit mode, the WTOE bit prevents a message center ID from being overwritten by an incoming remote frame.

The following examples demonstrate the use of the WTOE and other bits when using multiple message centers with the same arbitration value. Case 2 illustrates the approach described above for configuring multiple message centers to capture a large amount of data at a relatively high rate.

Case 1: WTOE = 1 (Overwrites allowed)

1. Software configures message centers 1 and 2 with the same arbitration value (abbreviated AV).
2. Software configures message centers 1 and 2 to receive ($T/\overline{R}$= 0) and allow message overwrite (WTOE = 1).
3. The first message received that matches AV is stored in message center 1, DTUP = 1.
4. The second message that matches AV is stored in message center 1, DTUP = ROW = 1.
5. The third message that matches AV is stored in message center 1.
6. Etc.

Note that in this example message center 2 never receives a message and that, if software does not read message center 1 before the second message is received, the first message is lost.

Case 2: WTOE = 0 (Overwrites disabled)

1. Software configures message centers 1 and 2 with the same arbitration value (abbreviated AV).
2. Software configures message center, 1 and 2 to receive ($T/\overline{R}$ = 0) and to disable message overwrite (WTOE = 0).
3. The first message received that matches AV is stored in message center 1, DTUP = 1.
4. The second message received that matches AV is stored in message center 2, DTUP = 1
5. Software reads message center 1 and then programs message center 1, DTUP = 0.
6. The third message received that matches AV is stored into message center 1, DTUP = 1.
7. Software reads message center 2 and then programs message center 2, DTUP = 0.
8. The fourth message received that matches AV is stored into message center 2, DTUP = 1
9. Etc.

Note that, in this example, message center 1 or 2 is never overwritten. The user must ensure that the proper number of message centers be allocated to the same arbitration value when using this arrangement. If software fails to read the allocated message group, an incoming message can be lost without software realizing it (ROW is never set when WTOE = 0). To put a message center back into operation, software must force DTUP = 0 and EXTRQ = 0. This indicates that software has read the message center.

Special Considerations for Message Center 15

Message center 15 incorporates a shadow message center used to buffer incoming messages, in addition to the standard message center registers. When the message center is empty (DTUP = EXTRQ = 0), incoming messages are loaded directly into the message center registers. When the message center has unread data (DTUP = 1) or a pending remote frame request (EXTRQ = 1), incoming messages are loaded into the shadow message center. Unread contents of the shadow message center are automatically loaded into the message center when it becomes empty (DTUP = 0). An overwrite condition is possible when both the message center 15 and shadow message centers are full.

The response of message center 15 to the overwrite condition is dependent on the write-over enable (WTOE) bit. When overwrite is enabled (WTOE = 1) and there is unread data (DTUP = 1) or a pending remote frame request (EXTRQ = 1), successfully received messages are stored in the shadow message center, overwriting existing data. If the shadow message center contained previously unread data at the time of the overwrite, the message center 15 ROW bit is set. If the shadow message center was empty at the time, then the incoming message is simply loaded into the shadow buffer and ROW is not set to a 1. Note that the message center 15 ROW bit reflects only an overwrite of the shadow message center, not the message center registers (as with message centers 1–14).

When WTOE = 0, there is unread data (DTUP = 1) or a pending remote frame request (EXTRQ = 1) in message center 15, and there is already a message stored in the shadow buffer, incoming messages are not stored in either the message center or shadow buffer.

Using the Autobaud Feature

It is sometimes necessary to connect a CAN node to a network with an unknown baud rate. The autobaud feature of the DS80C400 provides a simple way for the CAN module to determine the baud rate of the network and reconfigure the DS80C400 to operate at that baud rate. Special hardware inside the CAN module allows it to interface to a fully functional CAN bus and perform the autobaud feature without disturbing other CAN nodes.

The theory behind the CAN autobaud feature is relatively simple. If a CAN module operating at a particular baud rate listens in on a CAN bus operating at a different baud rate, it see a random bit stream. Because the bit stream does not conform to the CAN 2.0B protocol, a large number of bus errors (bit 0 error, bit 1 error, bit stuff error, etc.) are seen by the "listening" CAN. These errors increment the CAN error counter register. With only a moderate amount of CAN traffic, enough errors quickly accumulate to set the CAN error count exceeded (EC96/128) bit in the CAN 0 status register (C0S; A4h). This can be used as an indicator that the DS80C400 is not operating at the same baud rate as the CAN bus. The DS80C400 would then adjust its baud rate and repeat the process.

If, after a period of time, only a small number of errors have accumulated (most likely due to normal transmission noise), then the DS80C400 is operating at the correct baud rate. The autobaud process is further simplified by the fact that most networks only operate at a small number of values. For example, DeviceNet operates at 125kbps, 250kbps, and 500kbps, so a device attempting to autobaud to a DeviceNet network would only have to test three baud rates.

The autobaud feature of the CAN module is enabled by setting the autobaud bit (C0C.2). Setting this bit activates a special loopback circuit within the CAN module that logically ANDs incoming network data received on the RX pin with the TX pin of the CAN module. While the autobaud bit is set, the CAN module disables its transmit output and places it in the recessive (high) state, so that error frames generated by the autobauding CAN module do not disturb other devices on the network during the procedure. Also, while in this state, messages are received, but not stored.

The following user-defined software procedure can be used in conjunction with the autobaud feature to determine the baud rate of the network.

1. Set CRST = 1 to disable bus activity. Setting this bit also sets the SWINT bit, enabling access to control/status registers, and also clearing the C0RE and C0TE registers.

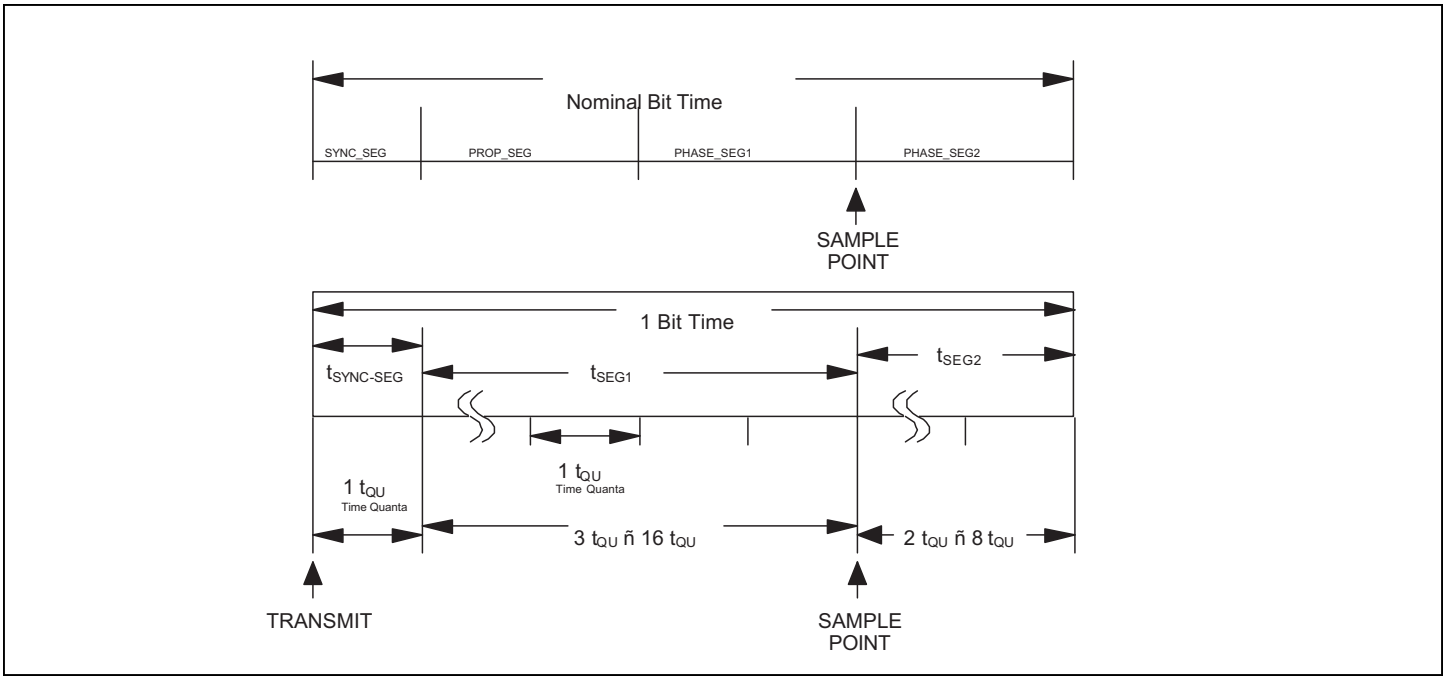


Figure 19-11. Bit Timing

2. Configure bus timing registers to set desired baud rate.
3. Set autobaud bit = 1.
4. Set SWINT = 0 to enable CAN module and begin listening for errors.
5. Delay approximately 500ms (allow enough time for >128 errors to occur).
6. If CAN error count exceeded (EC96/128) bit is set, baud rate is incorrect. Select a new baud rate and repeat procedure. If EC96/128 bit is not set, the DS80C400 CAN module is set to the correct baud rate.

Bus-Off/Bus-Off Recovery and Error Counter Operation

The CAN module contains two SFRs that allow software to monitor and modify (under controlled conditions) the error counts associat-

ed with the transmit- and receive-error counters in each CAN module. These registers can be read at any time. Writing the CAN transmit-error counter registers updates both the transmit-error counter registers and the receive-error counter registers with the same value. Details are given in the SFR description of these registers. These counters are incremented or decremented according to CAN specification version 2.0B, summarized in the following rules. The error counters are initialized by a CRST = 1 or a system reset to 00h. The error counters remain unchanged when the CAN module enters and exits from a low-power mode through the SIESTA or PDE bit.

Changes to the error counters are performed according to the following rules. This level of detail is not necessary for the average CAN user, and full information is provided in the CAN 2.0B specification. More than one rule can apply to a given message.

CONDITION	EFFECT ON ERROR COUNTERS
Error detected by receiver, unless the detected error was a bit error during the sending of an active error flag or an overload flag.	Receive-error counter incremented by 1.
Receiver detects a dominant bit as the first bit after sending an error flag.	Receive-error counter incremented by 8.
Transmitter sends an error flag. Note, however, that the transmit-error count does not change if: The transmitter is error passive and detects an acknowledgement error because of not detecting a dominant acknowledge, and does not detect a dominant bit while sending its passive error flag. Or, if the transmitter sends an error flag because a stuff error occurred during arbitration, and has been sent as recessive, but monitored as dominant.	Transmit-error counter incremented by 8.
Transmitter detects a bit error while sending an active error flag or an overload flag.	Transmit-error counter incremented by 8.
Receiver detects a bit error while sending an active error flag or an overload flag.	Receive-error counter incremented by 8.
Node detects the 14th consecutive dominant bit (in case of an active error flag or an overload flag), or detects the 8th consecutive dominant bit following a passive error flag, or after a sequence of additional eight consecutive dominant bits.	Transmit-error counter incremented by 8. Receive-error counter incremented by 8.
Message is successfully transmitted (acknowledge received and no error until end of frame is complete)	Transmit-error counter is decremented by 1 (unless it was already 0).
A message has been successfully received (reception without error up to the acknowledge slot and the successful sending of the acknowledge bit), and the receive-error count was between 1 and 127.	Receive-error counter decremented by 1.
A message has been successfully received (reception without error up to the acknowledge slot and the successful sending of the acknowledge bit), and the receive-error count was greater than 127.	Receive-error counter is set to a value between 119 and 127.

A node is error active when the transmit- and receive-error counters are less than 128. When in an error-active state, an error condition causes the node to send an error frame on the bus. A node is error-passive when the transmit-error count equals or exceeds 128, or when the receive-error count equals or exceeds 128. An error-passive node does not transmit an error frame on the bus. An error-passive node becomes error-active again when both the transmit- and receive-error counts are less than or equal to 127.

A node is bus off when the transmit-error count is greater than or equal to 256. A bus-off node becomes error active (no longer bus off) when its error counters are both set to 0 and after 128 occurrences of 11 consecutive recessive bits have been monitored on the bus.

After exceeding the error-passive limit (128), the receive-error counter is not increased further. When a message is received correctly, the counter is set again to a value between 119 and 127 (compare with CAN 2.0B specification). After reaching bus-off status, the transmit-error counter is undefined while the receive-error counter is cleared and changes its function. The receive-error counter is incremented after every 11 consecutive recessive bits on the bus. These 11 bits correspond to the gap between two messages on the bus. If the receive-error counter reaches count = 128 following the bus-off recovery sequence, the CAN module changes automatically back to the status of bus on and then sets SWINT = 1. After setting SWINT, all internal flags of the CAN module are reset and the error counters are cleared. A recovery from a bus-off condition does not alter the previously programmed MOVX memory values or SFR registers, apart from the transmit- and receive-error SFR registers and the error conditions displayed in CAN status register. The bus timing remains as previously programmed.

Bit Timing

Bit timing in the CAN 2.0B specification is based on a unit called the nominal bit time. The nominal bit time is further subdivided into four specific time periods.

1. The SYNC_SEG time segment is where an edge is expected when synchronizing to the CAN bus.
2. The PROP_SEG time segment is provided to compensate for the physical times associated with the CAN bus network
3 & 4. The PHASE_SEG1 and PHASE_SEG2 time segments compensate for edge phase errors. The PHASE_SEG1 and PHASE_SEG2 time segments can be lengthened or shorted through the use of the SJW1 and SJW0 bits in the CAN 0 bus timing register 0.

The CAN bus bit data is evaluated at a specific sample point. A time quantum (t_{QU}) is a unit of time derived from the division of the microcontroller system clock by both the baud-rate prescaler (programmed by the BPR7–BPR0 bits of the clock output register and CAN 0 bus timing register) and the system clock divider (programmed by the CD1:0 and 4X/$\overline{2X}$ bits of the PMR register). Combining the PROP_SEG and PHASE_SEG1 time segments into one time period termed t_{TSEG1}, and equating the SYNC_SEG time segment to t_{SYNC_SEG} and PHASE_SEG2 to t_{TSEG2}, provides the basis for the outlined time segments and the CAN bus timing SFR register descriptions. These are shown in the following figure.

The CAN 0 bus timing register 0 (C0BT0) contains the control bits for the PHASE_SEG1 and PHASE_SEG2 time segments, as well as the baud-rate prescaler (BPR5–0) bits. CAN 0 bus timing register 1 (C0BT1) controls the sampling rate, the time segment 2 bits that control the number of clock cycles assigned to the phase segment 2 portion, and the time segment 1 bits that determine the number of clock cycles assigned to the phase segment 1 portion. The value of both of the bus timing registers are automatically loaded into

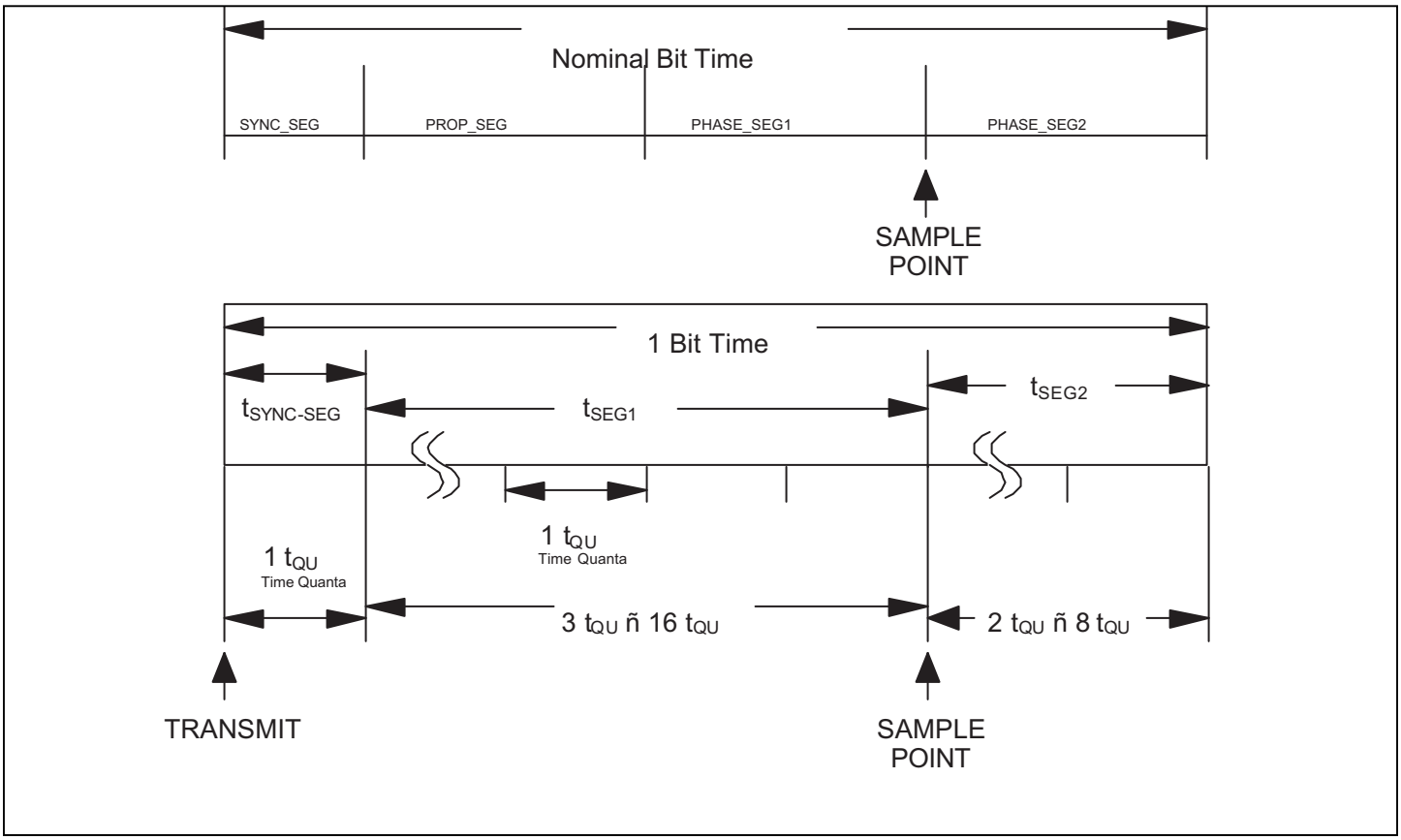


Figure 19-11. Bit Timing

the CAN module following each software change of the SWINT bit from a 1 to a 0 by the microcontroller. The bit timing parameters must be configured before starting operation of the CAN module. These registers can be modified only during a software initialization (SWINT = 1), when the CAN module is NOT in a bus-off mode, and after the removal of a system reset or a CAN reset. To avoid unpredictable behavior of the CAN module, the bus timing registers should never be written with all zeros. To prevent this, the SWINT is forced to 0 when TSEG1 = TSEG2 = 00h.

The timing of the various time segments is determined by using the following formulae. Most users never need to perform these cal-

culations, as other devices already attached to the network dictate the bus timing parameters.

$$t_{QU} = \frac{BRPV \times CCD}{F_{OSC}}$$
$$t_{SYNC\_SEG} = 1 \times t_{QU}$$
$$t_{TSEG1} = (TS1\_LEN) \times t_{QU}$$
$$t_{TSEG2} = (TS2\_LEN) \times t_{QU}$$
$$t_{SJW} = (SJW) \times t_{QU}$$
$$t_{QU\ PER\ BIT} = \frac{1}{BAUD\ RATE \times t_{QU}}$$

(Only integer values are permitted.)

Where BPRV is the CAN baud-rate prescaler value found in the earlier description of the C0BT0 register, F_{OSC} is the crystal or external oscillator frequency of the microprocessor, and TS1_LEN and TS2_LEN are listed in the description of the TSEG26-24 and TSEG13–10 bits in the CAN bus timing register 1. SJW is listed in the description of the SJW1-0 bits in the CAN bus timing register 0. The CAN clock divide value (CCD) is a factor connected to the current microcontroller system clock selection and can be referenced in the following table.

CD1	CD0	4X/$\overline{2X}$	CCD
0	0	1	0.5
0	0	0	1
1	0	x	2
1	1	x	512

The following restrictions apply to the above equations:

$$t_{TSEG1} \geq t_{TSEG2}$$
$$t_{TSEG2} \geq t_{SJW}$$
$$t_{SJW} < t_{TSEG1}$$
$$2 \leq TS1\_LEN \leq 16$$
$$2 \leq TS2\_LEN \leq 8$$
$$(TS1\_LEN + TS2\_LEN + 1) \leq 25$$

The nominal bit time applies when a synchronization edge falls within the t_{SYNC_SEG} period. The maximum bit time occurs when the synchronization edge falls outside of the t_{SYNC_SEG} period, and the synchronization jump width time is added to perform the resynchronization.

$$\text{Nominal bit time} = t_{SYNC\_SEG} + t_{TSEG1} + t_{TSEG2} = \frac{(BRPV)(CCD)[1+(TS1\_LEN)+(TS2\_LEN)]}{Fosc}$$
$$\text{Maximum bit time} = t_{SYNC\_SEG} + t_{TSEG1} + t_{TSEG2} + t_{SJW} = \frac{(BRPV)(CCD)[1+(TS1\_LEN)+(TS2\_LEN)+(SJW)]}{Fosc}$$
$$\text{CAN baud rate} = \frac{Fosc}{(BRPV)(CCD)[1+(TS1\_LEN)+(TS2\_LEN)}$$

Threefold Bit Sampling

The DS80C400 supports the ability to perform one or three samplings of each bit, based on the SMP bit (C0BT1.7). The single sample mode (SMP = 0) is available in all settings and takes one sample during each bit time. The triple sampling mode (SMP = 1) samples each bit three times for increased noise immunity. This mode can be used only when the baud-rate prescale value (BPRV) is greater than 3.

Bus Rate Timing Example

The following table shows a few example bit timing settings for common oscillator frequency and baud-rate selections. Because of the

large number of variables, there are many combinations not shown that can achieve a desired baud rate. There are a number of approaches to determining all the bit timing factors, but this uses the most common (i.e., the oscillator frequency and baud rate have already been determined by system constraints.)

Additional Bit Timing Examples

FOSC	CCD	BRPV	TQU	BAUD RATE	T_{QU} PER BIT	TS1_LEN	TS2_LEN	SJW	SMP = 1 PERMITTED?
40MHz	2	2	100ns	1Mbps	10	5	4	3	NO
	2	4	200ns	500kbps	10	5	4	3	YES
	2	5	250ns	250kbps	16	10	5	4	YES
	2	10	500ns	125kbps	16	10	5	4	YES
16MHz	0.5	2	125ns	1Mbps	16	10	5	4	NO
	1	1	125ns	1Mbps	16	10	5	4	NO
	2	1	125ns	1Mbps	8	4	3	2	NO
	2	1	125ns	500kbps	16	10	5	4	NO
	2	2	250ns	250kbps	16	10	5	4	NO
	2	4	500ns	125kbps	16	10	5	4	YES
8MHz	1	1	125ns	1Mbps	8	4	3	2	NO
	1	1	125ns	500kbps	16	10	5	4	NO
	1	1	250ns	250kbps	16	10	5	4	NO
	2	2	500ns	125kbps	16	10	5	4	NO

The following is an explanation of how the table row illustrating an oscillator frequency of 16MHz and a CAN baud rate of 125kbps is derived.

Various combinations of BRPV are selected until one is located that meets the t_{QU} per bit criteria, i.e., an integer value less than 25. Selecting BRPV = 4, the previously described equations state that there should be 16 t_{QU} per bit. That leaves 16–1 or 15 t_{QU} remaining for TS1_LEN and TS2_LEN, which are arbitrarily assigned as shown. Because BRPV > 3, the triple sampling feature (SMP = 1) can be used, if desired.

SECTION 20: ARITHMETIC ACCELERATOR

The DS80C400 incorporates an arithmetic accelerator that performs 32-bit and 16-bit calculations while maintaining 8051 software compatibility. Math operations are performed by sequentially loading three special registers. The mathematical operation is determined by the sequence in which three dedicated SFRs (MA, MB, and MCNT0) are accessed, eliminating the need for a special step to choose the operation. The arithmetic accelerator has four functions: multiply, divide, shift right/left, and normalize. The normalize function facilitates the conversion of 4-byte unsigned binary integers into floating point format. An integral 40-bit accumulator, described later, supports multiply-and-add and divide-and-add operations. The following table shows the operations supported by the math accelerator and their time of execution.

Table 20-1. Arithmetic Accelerator Execution Times

OPERATION	RESULT	EXECUTION TIME
32-bit by 16-bit divide	32-bit quotient, 16-bit remainder	36 t_{CLCL}
16-bit by 16-bit divide	16-bit quotient, 16-bit remainder	24 t_{CLCL}
16-bit by 16-bit multiply	32-bit product	24 t_{CLCL}
32-bit shift left/right	32-bit result	36 t_{CLCL}
32-bit normalize	32-bit mantissa, 5-bit exponent	36 t_{CLCL}

The following is a brief summary of the bits and registers used in conjunction with arithmetic acceleration operations. Please consult the SFR listing in Section 4 for a complete description of all these registers.

$\overline{\text{LSHIFT}}$
MCNT0.7
Left shift. This bit determines whether shift operations proceed from LSB to MSB, or vice-versa.

CSE
MCNT0.6
Circular shift enable. This bit determines whether shift operations wrap between the LSB and MSB.

SCE
MCNT0.5
Shift carry enable. This bit determines whether the arithmetic accelerator carry bit is included in the shift process.

MAS4–0
MCNT0.4–0
Multiplier register shift bits. When performing a shift operation, these bits determine how many shifts to perform. Following a normalize operation, these bits indicate the number shifts performed.

MST
MCNT1.7
Multiply/accumulate status flag. This bit serves as a busy flag for the arithmetic accumulator operations.

MOF
MCNT1.6
Multiply overflow flag. This bit is set when a divide-by-0 is attempted or when the result of a 16-bit by 16-bit multiplication exceeds FFFFh.

SCB
MCNT1.5
Shift carry bit. This bit serves as the carry bit during arithmetic accelerator shift operations when SCE = 1. This bit must be cleared or set by software as desired before each new shift operation.

CLM
MCNT1.4
Clear math accelerator registers. Setting this bit clears the MA, MB, and MC registers.

MA
MA.7–0
Multiplier A register. This register is used as both a source and result register for various arithmetic accelerator functions.

MB
MB.7–0
Multiplier B register. This register is used as both a source and result register for various arithmetic accelerator functions.

MC
MC.7–0
Multiplier C register. This register serves as the 40-bit accumulator of the arithmetic accelerator.

The following procedures illustrate how to use the arithmetic accelerator. The MA and MB registers must be loaded and read in the order shown for proper operation, although accesses to any other registers can be performed between accesses to the MA or MB registers. An access to the MA, MB, or MC registers out of sequence corrupts the operation, requiring the software to clear the MST bit to restart the math accelerator-state machine.

Divide (32-bit by 16-bit or 16-bit by 16-bit)

The divide operation utilizes a 32-bit or 16-bit dividend and a 16-bit divisor. The dividend is loaded into MA (4 bytes in the case of a 32-bit dividend, 2 bytes for a 16-bit dividend), and the 16-bit divisor is loaded into MB. The quotient is stored in MA and the remainder in MB. The optional test of the MOF bit can be performed to detect a divide-by-0 operation, if software has not previously checked for a nonzero divisor.

1. Load MA with dividend LSB.
2. *Load MA with dividend LSB + 1*.*
3. *Load MA with dividend LSB + 2*.*
4. Load MA with dividend MSB.
5. Load MB with divisor LSB.
6. Load MB with divisor MSB.
7. Poll the MST bit until cleared (nine machine cycles for 32 by 16 divide, six machine cycles for 16 by 16 divide).
8. Check MOF bit (MCNT1.6) to see if divide-by-0 occurred (optional).
9. Read MA to retrieve the quotient MSB.
10. *Read MA to retrieve the quotient LSB + 2*.*
11. *Read MA to retrieve the quotient LSB + 1*.*
12. Read MA to retrieve the quotient LSB.
13. Read MB to retrieve the remainder MSB.
14. Read MB to retrieve the remainder LSB.

*Steps 2, 3, 10, and 11 not performed for 16-bit dividend.

Multiply (16-bit by 16-bit)

This function multiplies two 16-bit values in MA and MB and places the 32-bit product into MA. If the product exceeds FFFFh, then the multiply overflow flag (MOF) is set.

1. Load MB with multiplier LSB.
2. Load MB with multiplier MSB.
3. Load MA with multiplicand LSB.
4. Load MA with multiplicand MSB.
5. Poll the MST bit until cleared (for six machine cycles).
6. Read MA for product MSB.
7. Read MA for product LSB + 2.
8. Read MA for product LSB + 1.
9. Read MA for product LSB.
10. Check MOF bit (MCNT1.6) to see if product exceeded FFFFh (optional).

Shift right/left

The shift function rotates the 32 bits of the MA register as directed by the control bits of the MCNT0 register. MA contains the shifted results following the operation. Note that the multiplier register shift bits (MCNT.4–0) must be set to a nonzero value, or the normalize function is performed instead of the desired shift operation.

1. Load MA with data LSB.
2. Load MA with data LSB + 1.
3. Load MA with data LSB + 2.
4. Load MA with data MSB.
5. Configure MCNT0 register as required.

6. Poll the MST bit until cleared (for nine machine cycles).
7. Read MA for result MSB.
8. Read MA for result LSB + 2.
9. Read MA for result LSB + 1.
10. Read MA for result LSB.

Normalize

The normalize function is used to convert four byte-unsigned binary integers into floating point format by removing all leading zeros through shift left operations. Following the operation, MA contains the normalized value (mantissa) and the MAS4–0 bits contain the number of shifts performed (characteristic).

1. Load MA with data LSB.
2. Load MA with data LSB + 1.
3. Load MA with data LSB + 2.
4. Load MA with data MSB.
5. Write 00000b to the MAS4–0 bits in the MCNT0 register.
6. Poll the MST bit until cleared (for nine machine cycles).
7. Read MA for mantissa MSB.
8. Read MA for mantissa LSB + 2.
9. Read MA for mantissa LSB + 1.
10. Read MA for mantissa LSB.
11. Read MAS4–0 to determine the number of shifts performed.

40-BIT ACCUMULATOR

The accelerator also incorporates an automatic accumulator function, permitting the implementation of multiply-and-accumulate and divide-and-accumulate functions without any additional delay. Each time the accelerator is used for a multiply or divide operation, the result is transparently added to a 40-bit accumulator. This can greatly increase speed of DSP and other high-level math operations.

The accumulator can be accessed anytime the multiply/accumulate status flag (MCNT1.7) is cleared. The accumulator is initialized by performing five writes to the multiplier C register (MC: D5h), LSB first. The 40-bit accumulator can be read by performing five reads of the multiplier C register, MSB first.

SECTION 21: 1-WIRE BUS MASTER

The 1-Wire master contained within the DS80C400 was designed to offload the task of 1-Wire communications from the microcontroller. Its main target is a network between the DS80C400 and a small number of local 1-Wire devices. This would include any 1-Wire chips or permanently attached ¡Buttons® located on the same PC board as the DS80C400. A minimum of external devices must be connected to the designated 1-Wire master pins in order to obtain proper function. This section discusses the hardware setup required, as well as the minimum steps necessary, to bring the 1-Wire master up and running.

HARDWARE SETUP

The 1-Wire system typically requires the slave devices to obtain their power parasitically from the master or, more specifically, from the 1-Wire (OW) line coming from the master. This I/O line is historically rated to function between 2.8V and 5.25V. The DS80C400, however, is a low-voltage microcontroller with a nominal I/O supply voltage (V_{CC3}) of 3.3V, leaving little operational headroom. This requires the parasitic power supplied by 1-Wire line to come from an external source, namely a pullup resistor to an external supply. Some devices, however, need a more direct connection to the parasite power source. This is necessary to obtain the large amounts of current they require without dropping below the minimum input voltage. The direct connection is obtained through a strong pullup bypass transistor.

The two pads on the DS80C400 involving the 1-Wire master both require external pullup resistors. Each pad should be pulled up by the same power-supply voltage that is used to power the slave devices. $\overline{\text{OWSTP}}$ is the control signal for the gate of the strong pullup transistor. A 10kΩ resistor should pull up the $\overline{\text{OWSTP}}$ output to (a recommended) 5V. The drain of the transistor should be connected to the OW line. In parallel with the strong pullup transistor is the OW weak pullup resistor. This resistor, typically being around 2.2kΩ, provides the power supply to the slaves most of the time. The following figure shows the described circuit.

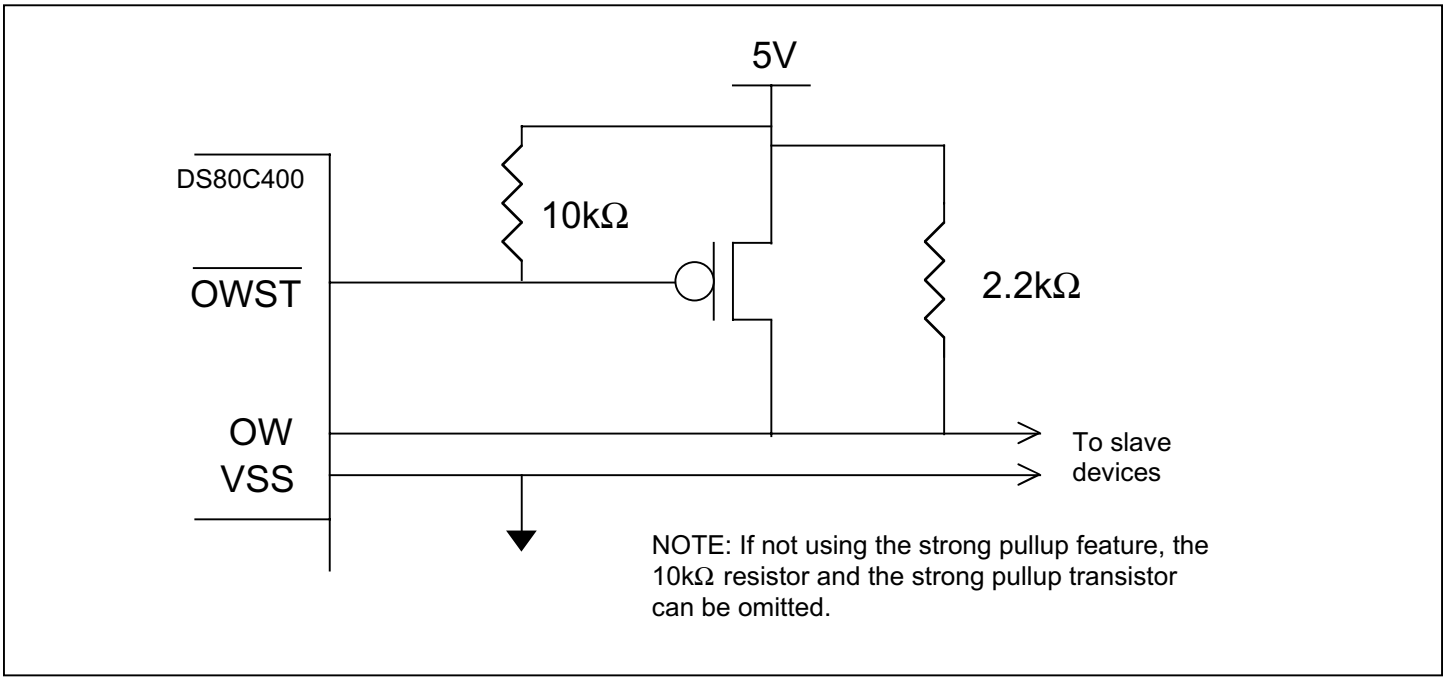


Figure 21-1. Typical 1-Wire External Hardware Configuration

SETTING UP AND USING THE 1-WIRE MASTER

The default state of the 1-Wire master is to remain completely shut down until accessed. When an application calls for the master's features, certain steps must be taken to bring it out of this shutdown state. The following is a quick guide to show the minimum setup requirements for the master to start communications followed by some basic communication code.

¡Button is a registered trademark of Dallas Semiconductor.

SETTING UP THE 1-WIRE MASTER

The first step is to determine the input crystal frequency to the DS80C400. Next, look up the appropriate divider ratio for that frequency in DS80C400 data sheet. This value, along with a logic 1 for the most significant bit, must be input into the clock divisor register that, in turn, enables the input clock and divides it down to the proper frequency. The following code demonstrates enabling the clock divider for a frequency of 10MHz.

```
       MOV    OWMAD, #004h ; Setup the address pointer
       MOV    OWMDR, #086h ; Set CLK_EN, and set divisor to 10MHz
```

SENDING A 1-WIRE RESET

The following code sends out a 1-Wire reset, waits for an interrupt to signal it has finished, and then checks the results to see if a part was located.

```
       ORG 43h
       LJMP Interrupt
       ORG 100h
       .
       .
Owrst:
       MOV    OWMAD, #003h ; Set the address pointer to the Interrupt Enable register
       MOV    OWMDR, #001h ; Enable the PD interrupt
       MOV    OWMAD, #005h ; Set the address pointer to the Control register
       MOV    OWMDR, #080h ; Set EOWMI to enable the OW interrupts
       MOV    OWMAD, #000h ; Set the address pointer to the Command register
       MOV    OWMDR, #001h ; Enable a 1-Wire Reset
       MOV IE, #080h; Enable the 400's interrupts
       MOV    EIE, #01h    ; Enable external interrupts 2-5 as well as OWMI
       LCALL Wait4int      ; Call the wait routine
       MOV    OWMAD, #002h ; Set the address pointer to the Interrupt register
       MOV    A, OWMDR     ; Read the value of the Interrupt register
       ANL    A, #002h     ; Mask off all bits but the PD Result bit
       CJNE A, #02h, Pdloop      ; Check to see if a part was found, loop again if not
       LJMP Cont           ; Continue on with your program
Pdloop:
       LJMP Owrst          ; perform 1-Wire Reset again
Cont:
       .
       .
Wait4int:
       MOV    R1, #000h           ; Clear flag
Waitloop:
       MOV    A, R1               ; Obtain the value in R1
       CJNE A, #01h, Waitloop;    ; Check flag and keep looping until set
       RET
Interrupt:
       MOV    EXIF, #08h   ; Turn off INT5 flag
       MOV R1, #001h; Set loop flag
       RETI
       END
```

SENDING A BYTE

The following code can be used in conjunction with the 1-Wire reset code to send out a byte across the 1-Wire line. In order to read a byte, the same program is used, but the byte value to send should be 0FFh. This, in turn, produces eight write 1's, which are the same as eight read-time slots.

```
       MOV    OWMAD, #003h ; Set the address pointer to the Interrupt Enable register
       MOV    OWMDR, #010h ; Enable the RBF interrupt
       MOV    OWMAD, #001h ; Set the address pointer to the TX/RX register
       MOV    OWMDR, #0AAh ; Load up the byte to be transmit
       LCALL Wait4int      ; Loop until the byte has been sent
```

```
MOV A, OWMDR ; Read the byte received to check against the value sent
```

As stated before, if performing a READ function, the byte transmitted should be 0FFh, and then the value read from the TX/RX register would have reflected the value sent by the slave(s).

SEARCH ROM ACCELERATOR

The 1-Wire bus master supports a search ROM-accelerator mode to expedite learning of ROM IDs for those devices connected to the bus. The bus master must determine the ROM IDs of the slave devices on the 1-Wire bus before it can address each slave device individually.

The search ROM command (F0h) is used by the bus master to signal external 1-Wire devices that a ROM ID search will be conducted. The search ROM command can be issued immediately following a reset sequence initiated by the master. Once the search ROM command has been issued by the bus master, slave devices simultaneously transmit, bit-by-bit, their unique ROM IDs. Listed below are the three 1-Wire bus time slots associated with each ROM ID bit acquistion.

Time Slots for Each ROM ID Bit Acquisition:

1) Read time slot—each slave transmits a single bit of its ROM ID (LSB first).
2) Read time slot—each slave transmits a complementary bit to that transmitted in 1.
3) Write time slot—bus master transmits discrepancy decision bit if needed.

The ROM ID acquisition and selection process listed above starts with the least significant bit of each slave device. If the ROM ID bits match for all currently selected slave devices, the two read time slots reflect complementary data and the bus master does not need to deselect or remove any slave devices from the selection process. The bus master simply repeats the time slot 1 read data as its write data for time slot 3, and continues to the next higher ROM ID bit acquistion period. Since it is expected that all 1-Wire devices have unique ROM IDs, time slots 1 and 2 above inevitably result in conflicting data being driven on the bus for at least one bit position when multiple slaves are connected. When this occurs, the wired-AND line state yields a 0 for time slots 1 and 2. At this point, the master has to send a bit value 1 or 0 to select the devices that remain in the search process. All deselected devices are idle until they receive a reset pulse. The four possible scenarios for slave ROM ID read time slots are shown in the table.

Table 21-1. ROM ID Read Time Slot Possibilities

READ TIME SLOT 1 (SLAVE)	READ TIME SLOT 2 (SLAVE)	WRITE TIME SLOT (MASTER)	FUNCTION
0	1	0	All slave devices remaining in the selection process have a 0 in this ROM ID bit position.
1	0	1	All slave devices remaining in the selection process have a 1 in this ROM ID bit position.
0	0	0 or 1	ID discrepancy—slave devices remaining in the selection process have both 0 and 1 in this ROM ID bit position. The bus master write time slot dictates which devices remain in the selection process.
1	1	1	Error—no slave devices responded during the read time slots.

The general principle of this search process is to deselect slave devices at every conflicting bit position. At the end of each ROM search process, the master has learned another ROM ID. A pass of search process takes 64 reading/selection cycles for the master to learn one device's ROM ID. Each reading/selection cycle, as noted above, consists of two read time slots and a write time slot. Subsequent search passes are performed identically to the last up until the point of the last decision. For details about search ROM algorithm in the 1-Wire system, refer to The Book of iButton Standards.

To speed up this ROM ID search process, the 1-Wire bus master incorporates a search ROM accelerator. To enable the search ROM accelerator, the SRA bit in the command register must be set immediately following the reset sequence and issuance of the search ROM command. After the bus master is placed in search ROM accelerator mode, each byte loaded into the transmit buffer contains one nibble (4 bits) worth of discrepancy decision data. The two slave read time slots are automatically generated by the bus master as a part of the transmit sequence. After four reading/selection cycles, the receive buffer data reflects four newly acquired bits of the ROM ID and four corresponding bits flagging whether a discrepancy existed in a given bit position. The format for the transmit and receive data (when in search ROM accelerator mode) is detailed in Table 21-2.

Table 21-2. Transmit/Receive Byte Sequence

BYTE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Byte 1								
Transmit Buffer	r_3	x	r_2	x	r_1	x	r_0	x
Receive Buffer	ID_3	d_3	ID_2	d_2	ID_1	d_1	ID_0	d_0
Byte 2								
Transmit Buffer	r_7	x	r_6	x	r_5	x	r_4	x
Receive Buffer	ID_7	d_7	ID_6	d_6	ID_5	d_5	ID_4	d_4
				....				
Byte 16								
Transmit Buffer	r_{63}	x	r_{62}	x	r_{61}	x	r_{60}	x
Receive Buffer	ID_{63}	d_{63}	ID_{62}	d_{62}	ID_{61}	d_{61}	ID_{60}	d_{60}

r_n = decision discrepancy data (write time slot selection data if ID discrepancy)
ID_n = selected ROM ID bit (rn if discrepancy occurred, otherwise read time slot 1)
d_n = discrepancy detected flag (ID discrepancy or no response)
x = don't care data

The CPU must send and receive 16 bytes of data to complete a single search ROM pass on the 1-Wire bus. To perform a search ROM sequence one starts with all decision discrepancy bits (rn) being 0. In case of bus error, all subsequent response bits IDn are 1s until the search accelerator is deactivated by clearing the SRA bit in the command register. Thus, if ID63 and d63 are both 1, an error has occurred during the search process and the last sequence has to be repeated. Otherwise, ID63:0 is the ROM code of the device that has been found and addressed. For the next search ROM sequence one reuses the previous set rn (for n = 0:63), changing to 1 only that bit position where the highest discrepancy was detected (dn flags). This process is repeated until the highest discrepancy occurs in the same bit position for two passes, then the next lower discrepancy flag is used for next search. When the search ROM process is completed, the SRA bit should be cleared in order to release the 1-Wire master from search ROM Accelerator mode.

Accelerated ROM Search Example

The following example should provide a better understanding of how the search ROM Accelerator functionality allows the 1-Wire master to identify four different devices on the 1-Wire bus. The ROM contents of the devices is as shown (LSB first):

```
ROM1 = 00110101....
ROM2 = 10101010....
ROM3 = 11110101....
ROM4 = 00010001....
```

1) The host issues a reset pulse by writing 01h to the command register. All slave devices respond simultaneously with a presence detect.

2) The host issues a search ROM command by writing 0Fh to the transmit buffer.

3) The host places the 1-Wire master in search ROM Accelerator mode by writing 02h to the command register.

4) The host writes 00h to transmit buffer and reads the return data from the receive buffer. This process is repeated for total of 16 bytes. The data read contains ROM4 in the ID bit locations and the discrepancy flags d0 and d2 are set. This can easily be seen by examining the ROM IDs bit by bit. The first discrepancy occurs in bit position 0 (d0). The bus master write time slot contains a 0, thus deselecting ROM2 and ROM3. A discrepancy between ROM1 and ROM4 then occurs in bit position 2 (d2), leaving only ROM4 in the search. The receive data is as follows: (d0ID0 d1ID1 d2ID2 d3ID3 d4ID4 d5ID5 d6ID6 d7ID7 ….):

5) Receive Data = 10 00 10 01 00 00 00 01....

6) The host then deinterleaves the data to arrive at a ROM ID of 00010001... and discrepancy data (bold) of 10100000…. with the last discrepancy at location d2.

7) The host writes 0x00h to the command register to exit accelerator mode. The host is now free to send a command or read data directly from this device.

8) Steps 1 to 6 are now repeated to find the next device on the bus. The 16 bytes of data transmitted this time are identical to ROM4 up until the last discrepancy flag (d2 in this case), which is inverted and all higher order decision discrepancy data

bits are set to 0 as shown: r0r1r2r3r4r5…. = 001000….. . For this search iteration, the receive data contains ROM1 in the ID bit locations, again with discrepancy flags d0 and d2 set. Receive Data = 10 00 11 01 00 01 00 01….

9) Since the most significant discrepancy (d2) did not change, the next highest discrepancy (d0) is used for the next search r0r1r2r3r4r5…. = 100000…..
Receive Data = 11 10 01 00 01 00 01 00....

10) Deinterleaving yields a ROM ID of 10101010.. (ROM2) and discrepancy flags of 11000000.. (d1 is the most significant flag).

11) The next search uses the ROM ID acquired in the previous search up until the most significant discrepancy: r0r1r2r3r4r5…. = 110000…

12) Receive Data = 11 11 01 01 00 01 00 01....

13) Deinterleaving yields a ROM ID of 11110101.. (ROM3) and discrepancy flags of 11000000.. (d1 is the most significant flag).

14) At this point, the most significant discrepancy (d1) did not change so the next highest discrepancy (d0) should be used. However, d0 has now been reached for the second time and since there are no lesser significant discrepancies possible, the search is completed and all four devices are identified.

SECTION 22: ETHERNET CONTROLLER

The DS80C400 incorporates a 10/100Mbps Ethernet controller, which supports the protocol requirements for operating an Ethernet/IEEE802.3-compliant PHY device. It provides receive, transmit, and flow control mechanisms through a media-independent interface (MII), including a serial management bus to allow external PHY configuration. A block diagram of the on-chip Ethernet controller can be seen in Figure 22-1.

Central to the Ethernet controller are the command/status (CSR) registers, which serve to define the operational behavior. Through the CSR registers, one is allowed to do such things as define the physical MAC address, set half- or full-duplex mode control, configure address-checking/filtering mechanisms, and operate the serial PHY management bus. The CSR registers are accessible through the SFR combination of BCUC (E7h), CSRA (E4h), and CSRD (E3h). Definitions for individual bits of each CSR register and the prescribed method for writing/reading CSR registers are contained in the DS80C400 data sheet.

In addition to configuring the Ethernet controller and the external PHY through the CSR registers, an adjustable on-chip 8kB transmit/receive data buffer, shared by the CPU and the Ethernet controller, is configurable by the Ethernet buffer size (EBS: E5h) SFR. The logical address location for this 8kB data memory is determined by the setting of the IDM1:0 bits of the MCON register.

The DS80C400 implements two Ethernet interrupt sources: Ethernet power mode interrupt and Ethernet activity interrupt. Once the Ethernet controller, external PHY, and packet buffer memory have been configured as desired, these two interrupt sources can be used to minimize CPU interaction with the Ethernet controller, thereby allowing the CPU more time to execute other tasks. By using interrupts, the CPU can conveniently manage Ethernet transmit and receive traffic using only the BCUC and BCUD SFR interface and the 8kB data buffer memory.

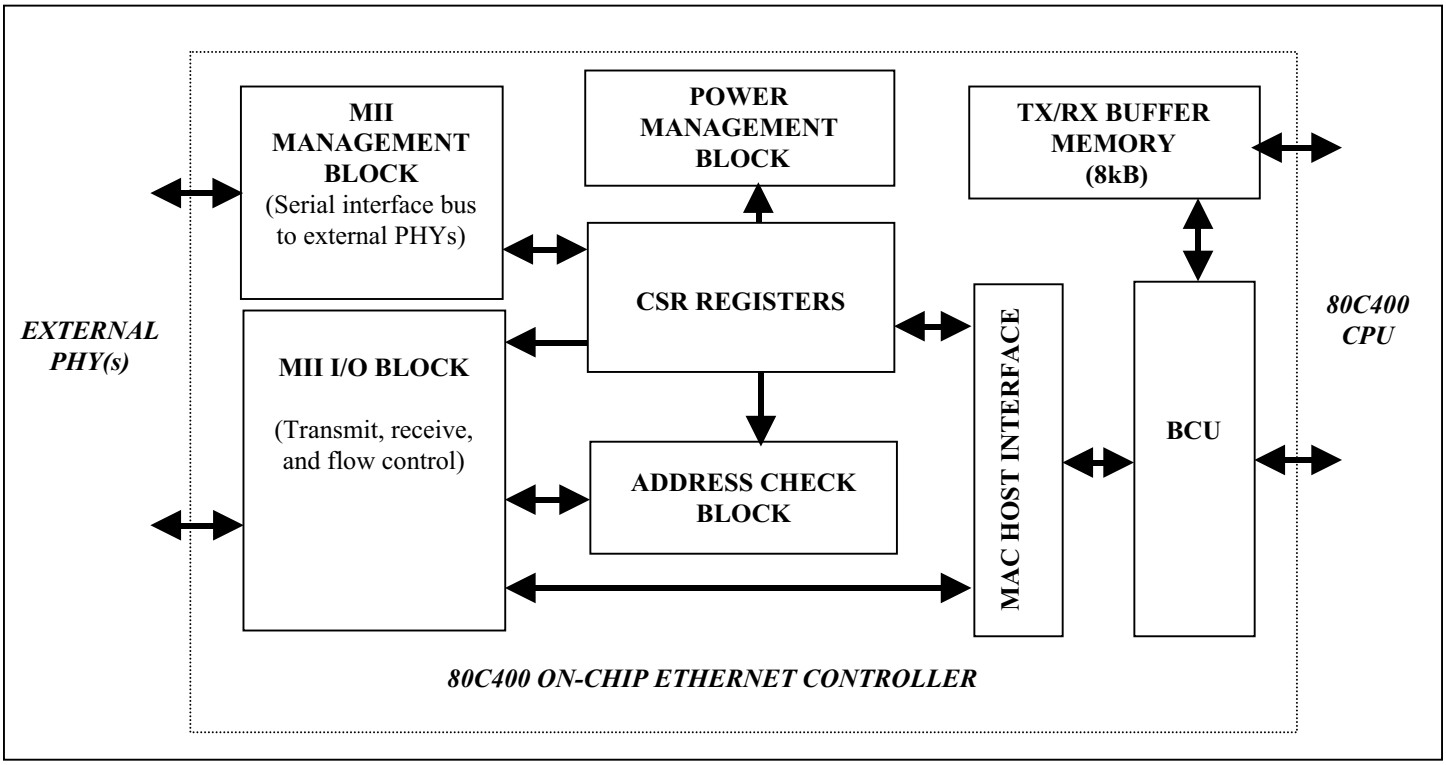


Figure 22-1. Ethernet Controller Block Diagram

Assigning a Physical MAC Address

The 48-bit physical MAC address for the Ethernet controller should be programmed in the MAC address high (04h) and MAC address low (08h) CSR registers. These on-chip registers are volatile memory locations and, following any reset, automatically default to the state: MAC address high = 00_00_FF_FFh, MAC address low = FF_FF_FF_FFh. This coincides with a physical MAC address of FF-FF-FF-FF-FF-FF or the broadcast address. The first 24 bits of the 48-bit physical MAC address are referred to as organization-unique identifiers (OUIs). OUIs are vendor specific and are assigned by IEEE. The last 24 bits of the physical MAC address are uniquely assigned by the individual hardware vendor. Generally, once a physical MAC address has been assigned to the network-enabled end product, it is thought to be nonvolatile.

To support this general expectation and automate the physical MAC address assignment, the embedded DS80C400 silicon software, when enabled, always executes initialization code to acquire and program the physical MAC address. This automatic NetBoot initialization of the physical MAC address assignment requires that the 48-bit address be preprogrammed into an externally connected 1-Wire device. Details of the NetBoot procedure and a list of acceptable 1-Wire devices for holding the 48-bit physical MAC address are found in the TINI400 embedded ROM section of this user's guide.

Alternatively, if the DS80C400 silicon software and/or NetBoot function are not used, user application code is responsible for programming the desired physical MAC address accordingly.

CONFIGURING THE MAC OPERATIONAL MODE

The DS80C400 media access controller works autonomously based upon the settings that have been programmed into various command/status (CSR) registers. The following table summarizes the MAC control (00h) CSR register bits for full-duplex, half-duplex, and loopback modes of operation, ordering the bits according to function as related to PHY interface, address/packet filtering control, CSMA/CD functionality, and data packet/buffer handling, respectively. Each mode of operation and associated function is covered in detail later in this section.

Table 22-1. MAC Control Register Bit Summary

OPERATIONAL MODE ==>	HALF-DUPLEX	FULL-DUPLEX	LOOPBACK
Bit 27: Port select	0 = MII, 1 = ENDEC		
Bit 28: Heartbeat disable	User-selectable ENDEC only	Invalid–set to 1	Invalid–set to 1
Bit 23: Disable receive own	**1**	**0**	**0**
Bits 22-21: Loopback operating mode	**00**	**00**	**01 or 10**
Bit 20: Full-duplex mode	**0**	**1**	**1**
Bit 3: Transmitter enable	MAC transmitter enable		
Bit 2: Receiver enable	MAC receiver enable		
Bit 31: Receive all	Address-filtering control		
Bit 19: Pass all multicast	Address-filtering control		
Bit 18: Promiscuous	Address-filtering control		
Bit 17: Inverse filtering	Address-filtering control		
Bit 15: Hash only	Address-filtering control		
Bit 13: Hash/perfect filtering	Address-filtering control		
Bit 16: Pass bad frames	Packet-filtering control		
Bit 11: Disable broadcast frames	Packet-filtering control		
Bit 12: Late collision control	CSMA/CD	Invalid–set to 0	Invalid–set to 0
Bit 10: Disable retry	CSMA/CD	Invalid–set to 0	Invalid–set to 0
Bits 7-6: Back-off limit	CSMA/CD	Invalid–set to 00	Invalid–set to 00
Bit 5: Deferral check	CSMA/CD	Invalid–set to 0	Invalid–set to 0
Bit 30: Endian mode	User-selectable	User-selectable	User-selectable
Bit 8: Automatic pad stripping	User-selectable	User-selectable	User-selectable

MEDIA INDEPENDENT INTERFACE (MII)

The DS80C400 fully supports the media-independent interface according to the IEEE802.3 standard. The MII interface provides independent transmit and receive datapaths, as well as input signals for monitoring network status. All standard PHY controller chips can use this default 4-bit parallel interface for connection to the Ethernet cable.

The transmit interface is comprised of TXCLK, TX_EN, and TXD[3:0]. The TXCLK input is the transmit clock provided by the PHY. For 10Mbps operation, the transmit clock (TXCLK) should be run at 2.5MHz. For 100Mbps, TXCLK should be run at 25MHz. The TXD[3:0] outputs provide the 4-bit (nibble) data bus for transmitting frame data to the external PHY. Each transmission begins when the TX_EN output is driven active-high, indicating to the PHY that valid data is present on the TXD[3:0] bus.

The receive interface is comprised of RXCLK, RX_DV, RX_ER, and RXD[3:0]. The RXCLK input is the receive clock provided by the external PHY. This clock (RXCLK) should be run at 2.5MHz for 10Mbps operation and at 25MHz for 100Mbps operation. The RXD[3:0] inputs serve as the 4-bit (nibble) data bus for receiving frame data from the external PHY. The reception begins when the external PHY drives the RX_DV input high, signaling that valid data is present on the RXD[3:0] bus. During reception of a frame (RX_DV = 1), the RX_ER input indicates whether the external PHY has detected an error in the current frame. The RX_ER input is ignored when not receiving a frame (RX_DV = 0).

The MII also monitors two network status signals that are provided by the external PHY. The carrier sense (CRS) input is used to assess when the physical media is idle. The collision detect (COL) input is required for half-duplex operation to signal when a collision has occurred on the physical media. Following is diagram showing a full MII interface. Data transactions between the MAC and the external PHY are done least significant nibble first as shown in Figure 22-2.

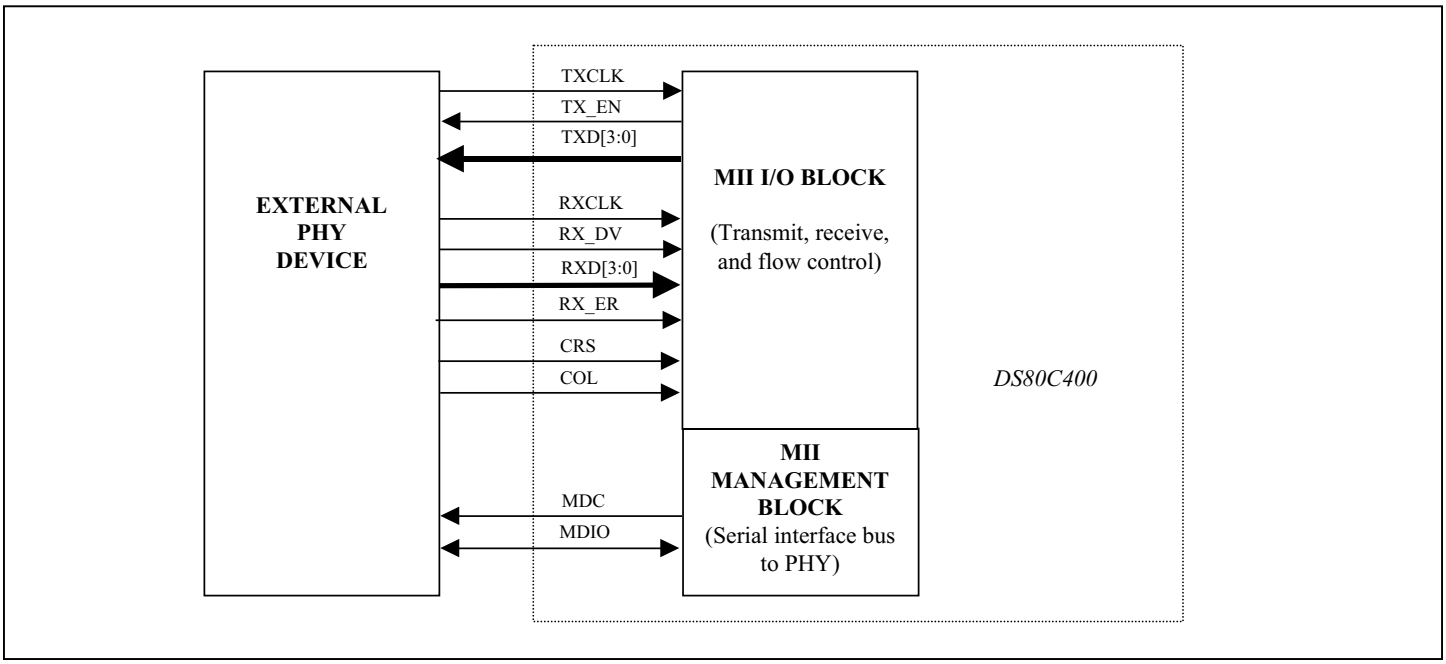


Figure 22-2. MII Signal Diagram

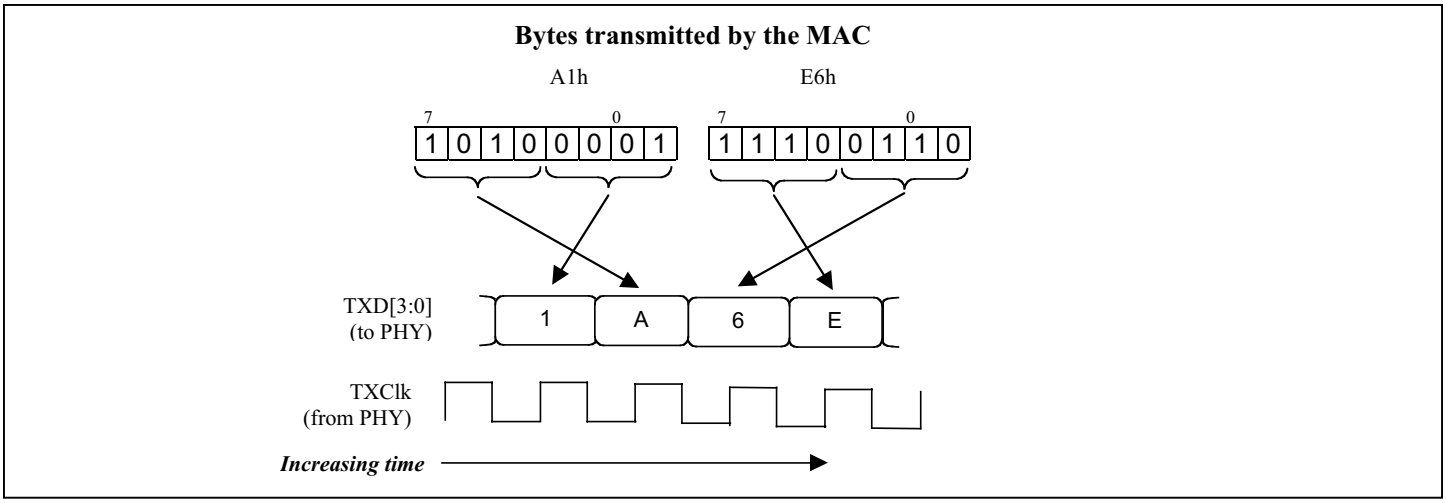


Figure 22-3. MII MODE–Byte/Bit Transmit and Receive Order

ENDEC OPERATION

The DS80C400 supports a serial ENDEC mode of operation, which is a subset of the MII mode of operation. The ENDEC mode can be used to support communication through GPSI (general-purpose serial interface) or SNI (serial network interface). The ENDEC mode of operation is selected by setting the port select bit (bit 27) of the MAC control CSR register. When ENDEC mode has been selected, only the lowest bit of each RXD3:0 and TXD3:0 nibble, RXD.0 and TXD.0, respectively, are used for data transactions. The only outputs generated by the DS80C400 are the TXEN and TXD.0 signals. All other signals are sourced from the PHY, including the TXCLK and RXCLK clocks, which must run at 10MHz to provide 10Mbps bandwidth. The RX_ER input signal is not used for ENDEC mode and should be connected in the inactive state (logic low). Figure 22-4 shows an example ENDEC interface. Serial data transactions conducted between the DS80C400 MAC and the external PHY over the TXD.0 and RXD.0 lines are done least significant bit first, as shown in Figure 22-4. The MII serial management bus (MDC, MDIO pins) operates no differently in ENDEC mode than MII mode, and can still be used for external PHY configuration.

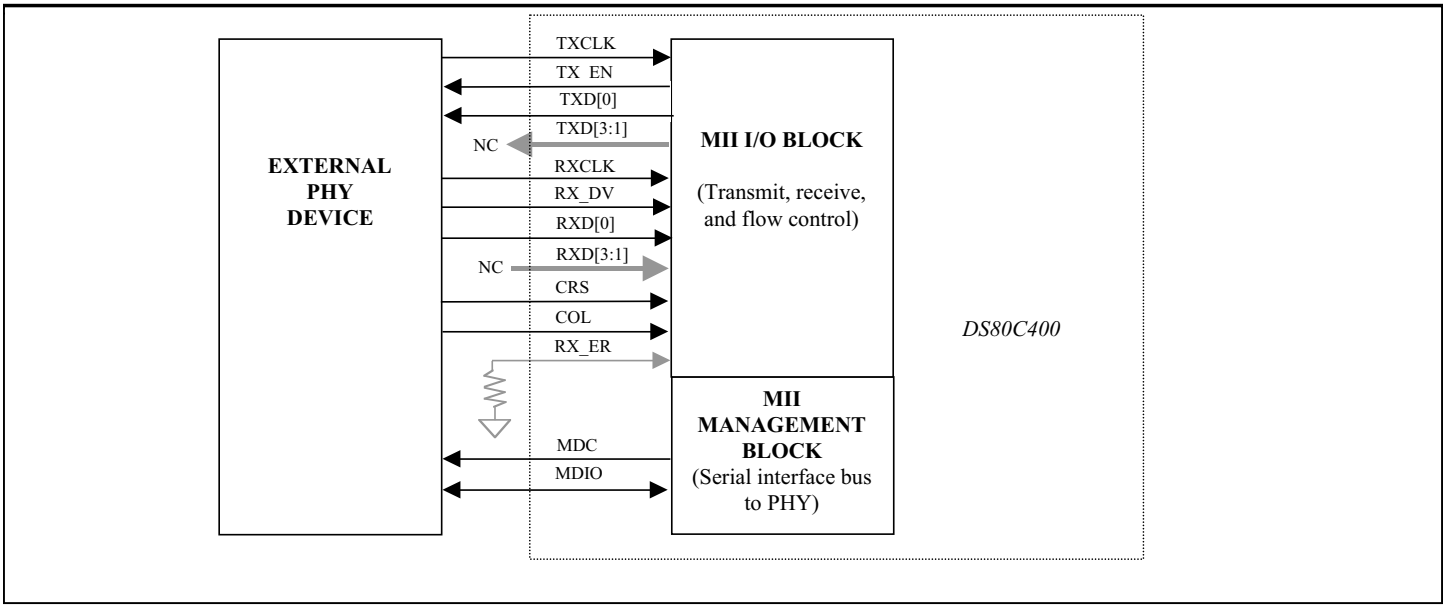


Figure 22-4. ENDEC Signal Diagram

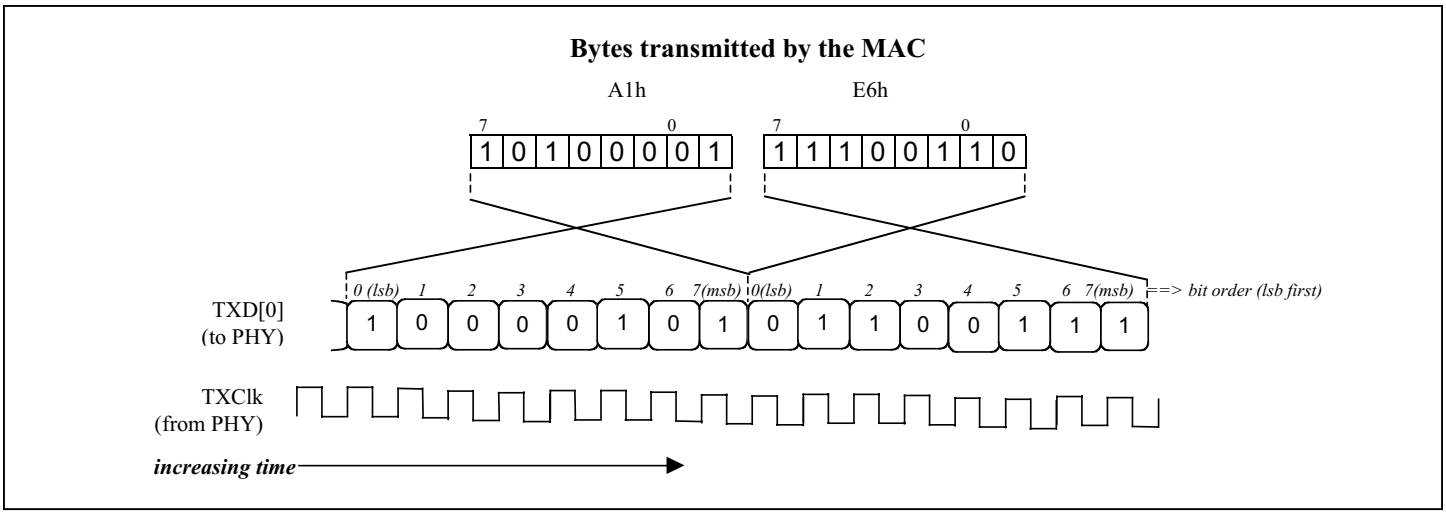


Figure 22-5. Serial ENDEC Mode–Byte/Bit Transmit and Receive Order

ENDEC MODE—HEARTBEAT SIGNAL QUALITY GENERATOR

When operating in ENDEC mode, the heartbeat signal quality generator (SQE) heartbeat function can optionally be enabled by clearing (= 0) the heartbeat disable bit (bit 28) of the MAC control CSR register. When the SQE function has been enabled, after each transmission, the MAC expects to receive a pulse on the COL pin after the TX_EN signal goes inactive. Failure to receive this SQE pulse from the PHY results in the heartbeat fail bit of the transmit status word being set for the packet. In order to use the SQE function, the PHY must support the SQE function and the function must be enabled.

MAC PRIMARY FUNCTIONS—PACKET FILTERING

The DS80C400 media access controller (MAC) provides programmable features designed to minimize host supervision and interaction. The MAC independently handles all necessary Ethernet framing and error checking requirements.

For transmission, the MAC automatically generates preamble and start-of-frame delimiter bytes. If the minimum frame length is not met, the MAC can automatically append zero-padding to the data field such that the frame length exceeds the required minimum length (46 bytes). The CPU can optionally request, when submitting to the BCU, that the automatic zero-padding not be added to the transmit packet. The MAC dynamically generates and appends the FCS to each transmit packet. Again, the CPU has the option of requesting that the FCS not be appended by the MAC. The MAC monitors the CRS and COL network status signals and, according to programmable MAC control register bit settings, can defer transmission temporarily or indefinitely, can automatically abort or retry collid-

ed frames, and can attempt retransmission according to a variable 1-bit to 10-bit back-off counter.

For reception, the MAC automatically synchronizes on the preamble and start-of-frame delimiter bytes. The MAC distinguishes among broadcast, multicast, and unicast frames. The MAC performs automatic minimum/maximum frame length and FCS checking on incoming frames. The MAC interprets the type/length field for each frame and can extend the maximum frame length for VLAN1 and VLAN2 tagged frames. The MAC can optionally be configured to strip zero-padding and FCS information before receive packet data is transferred (by the BCU) to data buffer memory. Additionally, the MAC can detect unsupported control frames, dribbling bit errors, runt frames, and any other errors signaled by the RX_ER input from the PHY. Bit 30 of the receive status word, the packet filter (PF) bit, serves as an indication of whether the MAC detected any errors in the receive frame. The receive status word contains other status bits such that the application can further discern what type of error was detected. One exception to this generalization is when the disable broadcast frames (DBF) bit of the MAC control register has been set to 1. When broadcast frames have been disabled (DBF = 1), the PF bit is returned as 0 in the receive status word for any broadcast frame received.

USING THE MII SERIAL MANAGEMENT BUS

The DS80C400 provides a 2-wire, serial MII management interface for communication with external PHY devices. This interface is comprised of the MDC clock signal and the bidirectional MDIO data line. The MDC clock rate is derived from the system clock frequency, and data is valid on the rising edge of the clock (MDC). The AC timing section of the DS80C400 data sheet contains detailed timing information relative to the MDC, MDIO signals. The MII management frame format, as defined by clause 22 of the IEEE802.3 standard, is shown in Figure 22-6. The CSR registers MII address (14h) and MII data (18h) allow the CPU to execute read/write operations over the two-wire serial bus. The MII address register supports a 5-bit address field and 5-bit register pointer field and therefore can address up to 32 registers in as many as 32 PHY devices. The MII data register holds the returned 16-bit data following a read operation and should be loaded with the desired 16-bit data prior to a write operation. The MII serial management bus operates identically for full-duplex MII, half-duplex MII, and serial ENDEC mode.

Figure 22-6. MII Management Frame Format

	Preamble (32 bits)	Start (2 bits)	Op code (2 bits)	PHY Address (5 bits)	PHY Register (5 bits)	Turn Around (2 bits)	Data (16 bits)	Idle (1 bit)
Read	111…111	01	10	PHYA [4:0]	PHYR[4:0]	ZZ*	ZZ….ZZ*	Z
Write	111…111	01	01	PHYA [4:0]	PHYR[4:0]	10	PHYD[15:	Z

** During a read operation, the external PHY drives the MDIO line low for the second bit of the turnaround field to indicate proper synchronization, and then drives the 16 bits of read data requested.*

HALF-DUPLEX OPERATION–CSMA/CD AND FLOW CONTROL

The CSMA/CD protocol specifies that each Ethernet station must wait until there is no signal on the channel. Then it can begin transmitting. If there is a signal (carrier) on the channel, all other stations must wait until carrier ceases before trying to transmit. The protocol supports multiple accesses, allowing all Ethernet stations The equal ability to send frames onto the network. Because signals take a finite time to travel from one end of an Ethernet system to the other, the first bits of a transmitted frame do not reach all parts of the network simultaneously. Therefore, it is possible for more than one station to sense that the network is idle and to start transmitting their frames simultaneously. When this happens, the Ethernet system has to be able to sense the "collision" of signals, stop the transmission, and resend the frame.

The CSMA/CD protocol is designed to provide fair access to the shared channel so that all stations get a chance to use the network. After every packet transmission, all stations use the CSMA/CD protocol to determine which station gets to use the Ethernet channel next. A collision occurs if more than one station transmits on the Ethernet channel at the same moment. The stations are notified of this event, and instantly reschedule their transmission using a specially designed back-off algorithm. As part of this algorithm, the stations involved choose a random time interval to schedule the retransmission of the frame, which keeps the nodes from making retransmission attempts simultaneously. Collisions are normal and expected events on an Ethernet. As more stations are added to a given Ethernet, and as the traffic level increases, more collisions occur as part of the normal operation of an Ethernet. A normal collision does not result in lost data. In the event of a collision, the Ethernet interface backs off for microseconds and then automatically retransmits the data. On a network with heavy traffic loads, multiple collisions can occur for a given frame transmission attempt. If repeated collisions occur for a given transmission attempt, then the stations involved begin expanding the set of potential back-off times (truncated binary exponential back-off), from which they choose their random retransmission time. This provides an automatic method for Ethernet controllers to adjust to traffic conditions on the network. Only after 16 consecutive collisions for a given transmission attempt does the station finally discard the Ethernet packet. This can happen only if the Ethernet channel is overloaded for long period of time or is broken in some way.

DEFERRAL CHECK

When a transmit request is queued, the MAC monitors the CRS line to determine when the physical carrier becomes idle. If the physical carrier is not initially idle, the MAC defers transmission until the carrier becomes idle. The deferred state persists as long as the physical carrier remains busy (CRS = 1). Optionally, the MAC can be configured to abort a transmit request if deferred excessively. Setting the deferral check (DC) bit, bit 5 of the MAC control (00h) CSR register, forces the MAC to abort a transmit attempt that has been deferred for more than 24,288 bit times. The deferral timer resets once the MAC is able to begin the transmit attempt. The deferral (DFR) bit of the transmit status word is set if the MAC must defer during the transmit attempt. The excessive deferral (XDFR) bit of the transmit status word is set if a packet is aborted because of excessive deferral.

DISABLE RETRY

Once transmission begins, the MAC monitors the COL pin to detect when a collision occurs. If a collision occurs within the first 512-bit time normal collision window, the MAC sends a jamming signal and, by default, waits some random number of timeslots (according to an internal back-off counter) before attempting the transmission again. The MAC attempts to transmit the packet as many as 16 times before aborting the transmit request because of excessive collisions. The MAC can optionally be configured to abort the transmit packet if a collision occurs on the first attempt. The disable retry bit, bit 10 of the MAC control register, forces the MAC to abort the packet if the COL pin is asserted (= 1) at any time during the transmission. The excessive collisions (XCOL) bit of the transmit status word is set if the MAC aborts the transmit attempt because of excessive collisions (DTRY = 1 for 1 collision; DRTY = 0 for 16 collisions).

BACK-OFF LIMIT

Whenever a collision occurs during transmission, the MAC sends a jamming signal and backs off for some amount of time before attempting the transmission again (given that DRTY = 0). An internal 10-bit pseudorandom counter is used to generate the back-off delay. The back-off limit bits, BOLMT1:0, define how many bits of the counter are used in determining the back-off delay. The four settings for the back-off delay counter are given in the DS80C400 data sheet.

LATE COLLISION CONTROL

The maximum round-trip network delay gives a 512-bit time window within which normal network collisions can be expected to occur. By default, the DS80C400 MAC aborts a transmit packet if a collision is detected beyond this normal collision window. Optionally, the MAC can be configured to react to a late collision in a similar fashion as it would to a normal collision. By setting the late-collision control (LCC) bit of the MAC control register, the MAC attempts retransmission of the packet just as if the collision had occurred within the normal collision window. The observed late collision (OLTCOL) bit of the transmit status word is set when a collision is observed beyond the normal 512-bit time collision window. The late collision (LTCOL) bit of the transmit status word is set if the transmit packet was aborted because of a late collision.

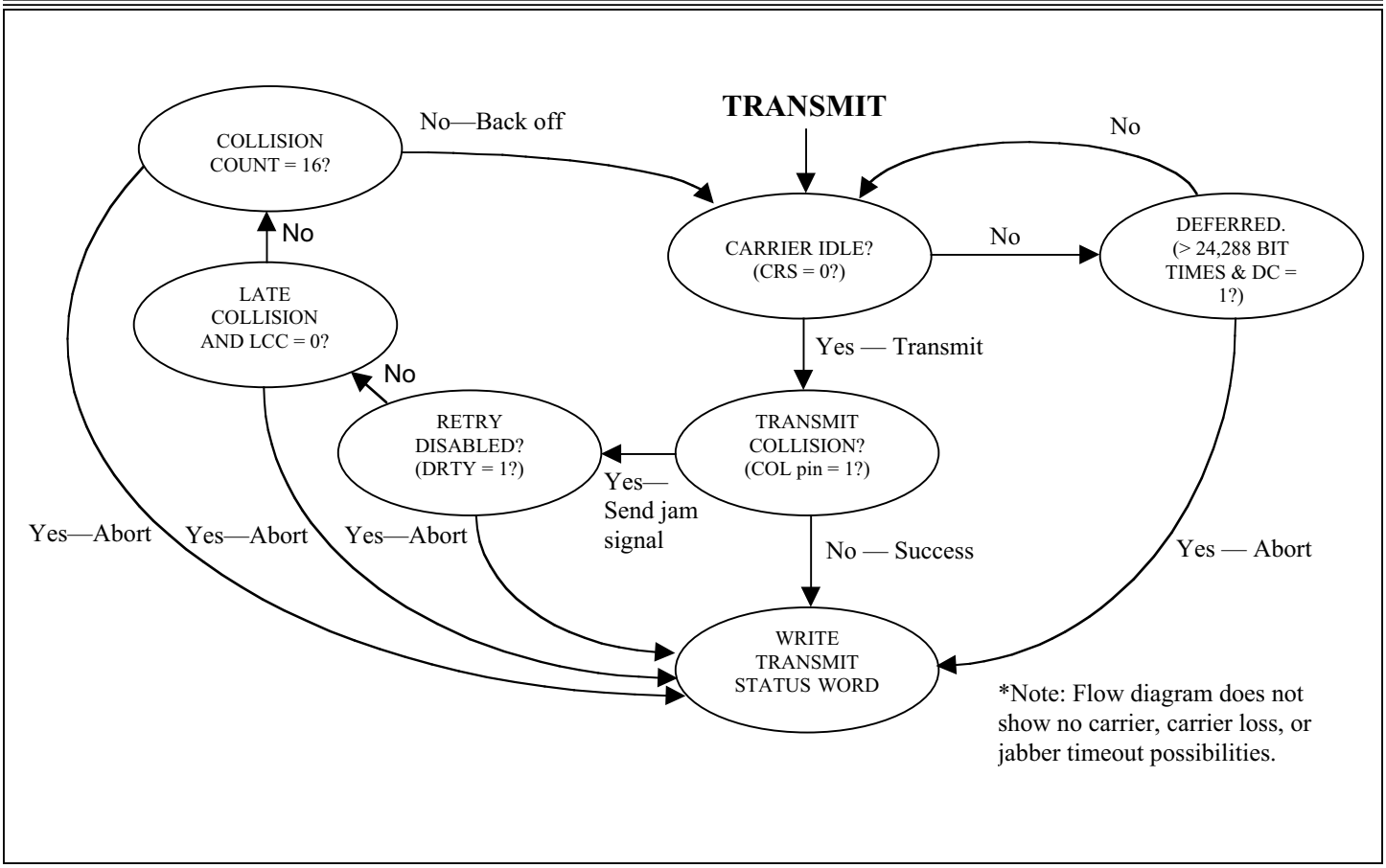


Figure 22-7. Half-Duplex Transmit Deferral/Collision Handling

FLOW CONTROL

In half-duplex mode, the MAC supports receive side flow control through back pressure. The DS80C400 asserts back pressure when the receive buffer has reached the threshold level of five or fewer available pages. Back pressure is asserted by transmitting a jamming signal of four to 6 random bytes on to the bus. The jamming signal is intended to cause collisions for other nodes on the bus to allow the DS80C400 to free additional receive buffer pages. Back pressure occurs only if the flow control enable (FCE) bit of the flow control (1Ch) register is set to 1. Back pressure is not asserted if triggered during reception of a frame, but occurs for any subsequent receive frames, assuming that the receive buffer is still below the threshold limit.

FULL-DUPLEX OPERATION

The DS80C400 supports full-duplex Ethernet operation, allowing simultaneous transmit and receive operation over the media independent interface. When using full-duplex Ethernet, the physical media is not shared and therefore does not require CSMA/CD. For this reason, the COL pin is ignored by the MAC and the interpacket gap (IPG) timer for the transmit side is based solely upon the TX_EN signal instead of the CRS input. For receive side flow control, the PAUSE control frame is used.

PAUSE CONTROL FRAME

The pause control frame is used to inhibit transmission of data frames for a specified time (within the pause frame). A pause control frame consists of the globally assigned multicast address 01-80-C2-00-00-01, the pause op code, and a parameter specifying the quanta of slot time (512 bit times/slot) to inhibit data transmission. The pause parameter can range from 0 to 65,536 slot times. When the MAC receives a frame with the reserved multicast address and pause op code, the MAC inhibits data frame transmission for the length of time as indicated by the pause parameter. If a pause request is received while a transmission is in progress, then the pause takes effect after the current transmission is completed. Pause control frames are automatically received and processed by the MAC, and can be passed on to the application (PF bit of the receive status word is set) when the pass-pause control frame bit in the flow control register is set.

The MAC also can transmit a pause control frame on the request from the application. To initiate a pause control frame, the desired pause time [15:0] interval should first be written to the flow control register. The application must then write a 1 to the BUSY bit of the flow control register to trigger the transmission. The MAC constructs a pause control frame using these values and transmits the frame to the MII interface. The transmission of the pause control frame does not affect, and is not affected by, the state of the pause timer, which could be running because a previously received pause control frame. Upon the completion of the pause control frame transmission, the FCB bit is cleared. Note that the user should check the state of the FCB bit before writing to the flow control register when initiating a pause control frame. When the FCB is set to a 1, it signifies a pause control frame transmission is still in progress.

LOOPBACK MODES

The DS80C400 implements two diagnostic loopback modes: internal loopback through the MII, and external loopback through the PHY. To support either of the loopback modes, the MAC must be configured identically to the normal full-duplex operating mode, except for the bits used to select the loopback mode. The loopback mode is controlled by the loopback operating mode bits (OM[1:0]) of the MAC control (00h) CSR register. The OM[1:0] bits default to the 00b state for normal operation with no loopback. Internal loopback through the MII is selected by configuring the OM[1:0] bits to the 01b state. In the internal loopback mode, TXD[3:0] is looped back internally to RXD[3:0], TX_EN is looped back internally to RX_DV and CRS, and TXCLK (sourced from the PHY) is looped back internally to RXCLK. RX_ER is internally sampled low by the MAC, and COL is ignored due to full-duplex operation. The internal loopback mode maps the MII interface pins as diagrammed in Figure 22-8.

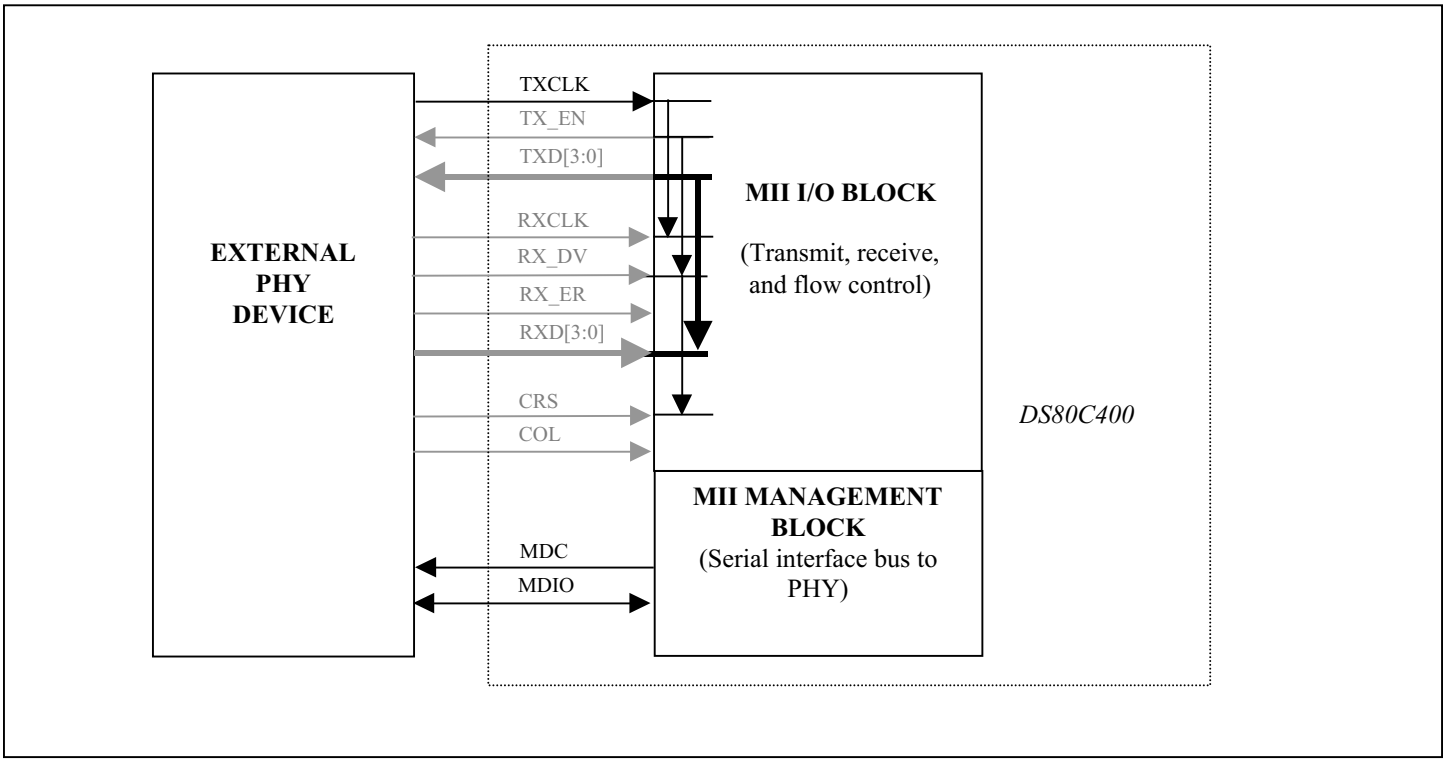


Figure 22-8. Internal Loopback Mode (MAC Control OM1:0 = 01b)

The external loopback mode is selected by configuring the OM[1:0] bits to the 10b state. The external loopback mode first requires that the serial MII management bus (MDC, MDIO) be used to configure the external PHY for local loopback operation. Once the PHY is placed into the loopback mode, it disregards activity on the physical layer but forces transmissions made by the DS80C400 transmit interface to be looped back to the receive interface. In external loopback mode, the DS80C400 MAC behaves exactly as if it were in the normal full duplex mode, requiring that the external PHY perform the signal loopback function. The external loopback mode is diagrammed in the following figure.

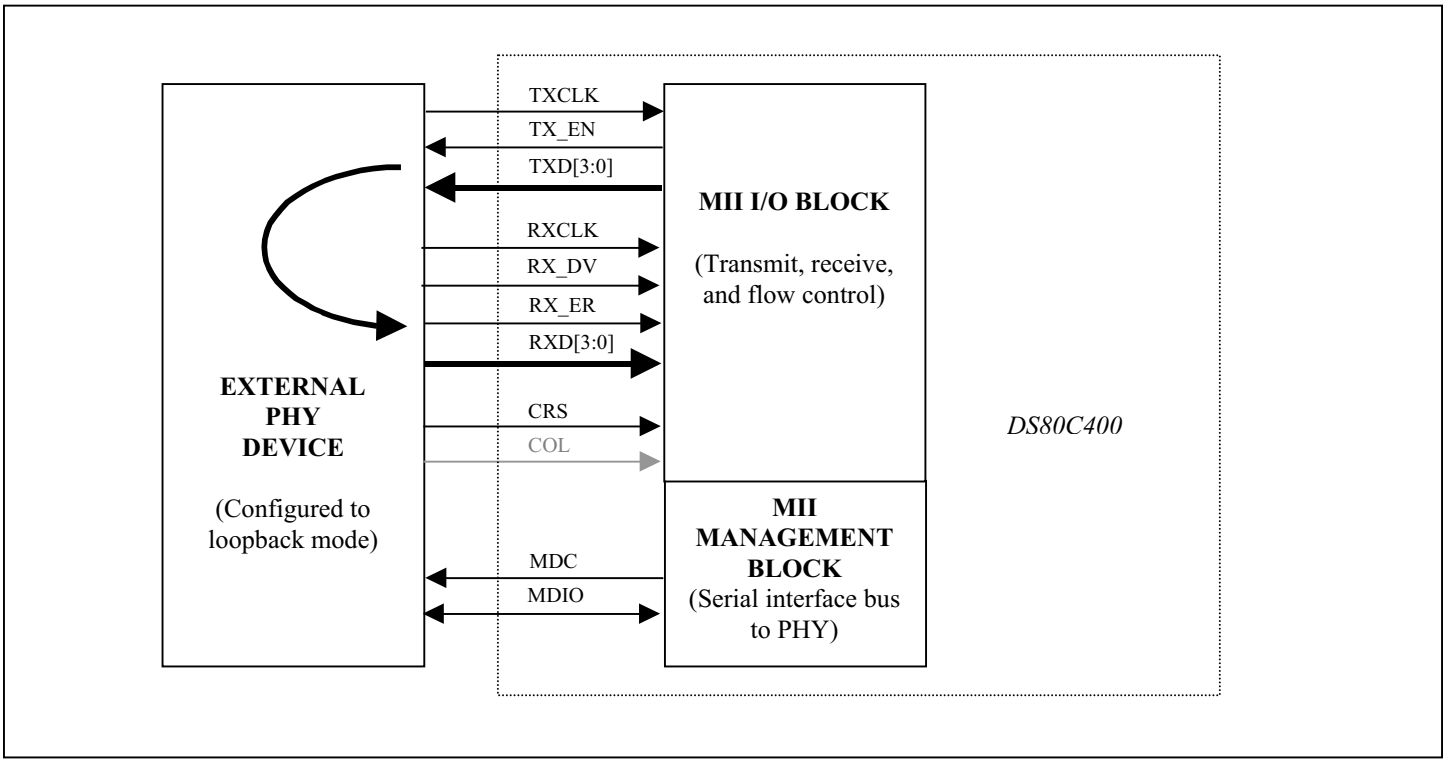


Figure 22-9. External Loopback Mode (MAC Control OM1:0 = 10b)

ADDRESS FILTERING CONTROL

The destination address of each Ethernet packet received by the MAC is examined by the address check block of the Ethernet controller. The address check block and associated MAC control register bits that define the destination address-filtering mode are covered in the DS80C400 data sheet. The destination address is tested against the currently defined address filter, and the pass/fail status is reported (by the BCU) to the filter fail (FF) bit of the receive status word. Additional status bits indicating whether the destination address is a multicast or broadcast frame are also written by the BCU to the receive status word. Each receive packet, whether it passes or fails the destination address filter, normally is written to the receive data buffer. In the same respect, as the PF bit is normally used to indicate whether a frame was error-free, the FF bit normally is used to assess whether it passed the address filter, allowing the CPU to quickly determine whether the packet should be processed or discarded. The following table summarizes the possible receive frames and resulting PF, FF bit states reported in the receive status word.

Table 22-2. Packet Filter and Filter Fail Bit Status for Various Received Frames

PF, FF STATUS BITS	RECEIVED FRAME				SPECIAL FILTERING CONDITION ENABLED?
	NO ERRORS	WITH ERRORS	PASSED ADDRESS FILTER	FAILED ADDRESS FILTER	
00		X	X		
00	X		don't care	don't care	Promiscuous mode (PM = 1)
00	don't care	don't care	X		Broadcast frame AND DBF = 1
01		X		X	
10	X		X		
10	X		don't care	don't care	Promiscuous mode (PM = 1)
10		X	X		Pass bad frames (PB = 1)
11	X			X	
11		X		X	Pass bad frames (PB = 1)

One way to prevent receive packets from always being stored to the receive data buffer, and thus prevent needless interruption of the

CPU, is to use the flush filter failed-packet enable function. The flush filter failed-packet enable (FPE: EBS.7) bit of the EBS SFR is provided as a simple means to automatically manage consumption of the receive data buffer. The FPE bit enables the BCU to flush a receive packet as soon as it fails the destination address filter. Setting the FPE bit to 1 causes the BCU to ignore any receive data once the destination address filter has failed. Thus, no pages in the receive data buffer are consumed, the receive FIFO is not updated, and the CPU is not interrupted. Note that the promiscuous mode (PM = 1) results in the FF bit always being returned clear (= 0), effectively disabling the flush filter packet enable function. Setting the receive all (RA) bit of the MAC control CSR register to 1 overrides the FPE bit but allows the FF bit to be set according to the current destination address filter. Clearing the FPE bit to 0 results in the BCU receiving all incoming packets into the receive buffer, regardless of the destination address filter.

USING THE HASH TABLE

The MAC control (00h) CSR register contains two bits, hash only (HO, bit 15) and hash/perfect (HP, bit 13), that can be configured to initiate hash filtering of destination addresses. These two filter mode control bits and the resulting address filtering mode are described in the DS80C400 data sheet. Once hash filtering has been selected, the destination address is passed through the CRC-32 generator circuitry in order to produce an index into the hash table formed by the multicast address high (0Ch) and multicast address low (10h) CSR registers. The most significant bit of the resultant CRC-32 is used to select one of the two CSR registers just mentioned. The next five most significant bits are used to select one of 32 programmable bits in the respective CSR register. If the selected bit per the 6-bit indexing process just described is set to 1, the destination address passes the hash address filter. If the indexed bit is 0, the destination address fails the hash address filter. Remember that each bit in the hash table corresponds to many addresses, so the application must perform additional checking in order to make sure that the address matches with one that it wants to pass.

VLAN SUPPORT

The DS80C400 supports one-level and two-level tagged VLAN frames. A VLAN-tagged frame contains a tag protocol ID (TPID) in the 13th and 14th bytes (those bytes normally occupied by the type/length field). These 2 bytes are compared to the values programmed in the VLAN1 (20h) and VLAN2 (24h) CSR registers. If a nonzero match occurs with the VLAN1 register, the maximum frame length is extended by four bytes, two bytes for the TPID already received, plus 2 bytes for the tag control information that immediately follows the TPID. If the 13th and 14th bytes match the VLAN2 register settings, the maximum frame length is extended by 20 bytes, 2 bytes for the TPID already received, plus 18 bytes for any tag control information. The DS80C400 data sheet diagrams the VLAN1 and VLAN2 tagged frames. The absolute requirements for virtual LAN implementations are not rigidly defined by IEEE, thus the DS80C400s VLAN1 and VLAN2 register options provide the system administrator with the flexibility to accommodate different VLAN schemes.

PARTITIONING THE 8kB ETHERNET DATA BUFFER MEMORY

An on-chip 8kB SRAM is provided for Ethernet transmit/receive data packet buffering. The address location of this 8kB data memory is determined by the setting of the IDM1:0 bits of the MCON (C6h) SFR. The 8kB data buffer memory is logically configured as 32 256-byte pages. These 32 pages are partitioned between receive and transmit data buffer memory per the setting of the buffer size (BS4:0) bits of the EBS (E5h) SFR. The BS4:0 bits can be programmed to any value *n* between 0 and 31, inclusive. The receive buffer occupies the first *n* pages of the 8kB memory, while the transmit buffer occupies the remaining (32-*n*) pages. Changing the BS4:0 bits automatically flushes the contents of the receive buffer and receive FIFO. The Ethernet buffer control unit directly accesses the 8kB data buffer memory through its 32-bit wide datapath. The CPU must access the 8kB data buffer memory one byte at a time using MOVX instructions. Each of the data buffers, receive and transmit, operate in a circular fashion. Following is a diagram showing how one could partition the 8kB buffer memory for 8 receive pages and 24 transmit pages.

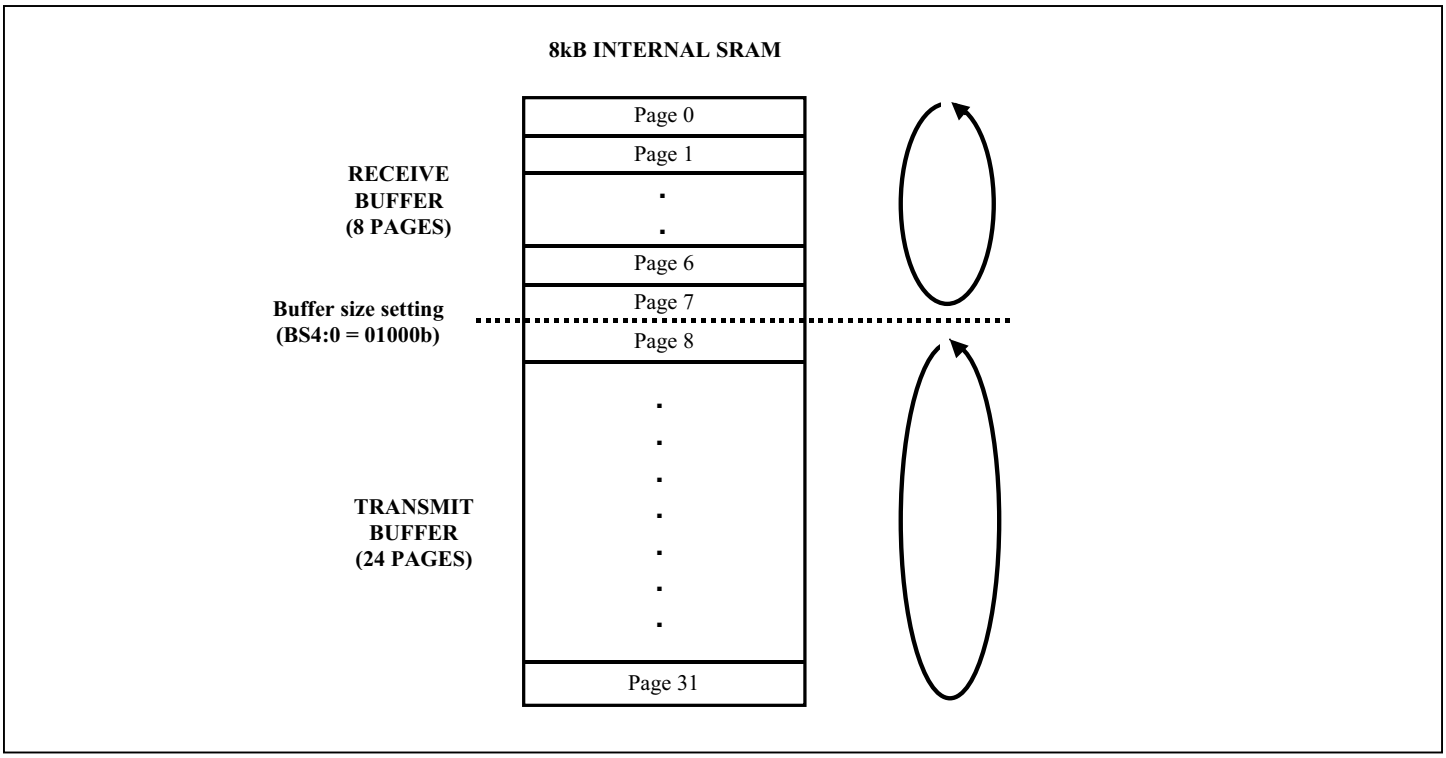


Figure 22-10. Example 8kB Data Memory Partition

TRANSMIT/RECEIVE DATA BUFFER WORD ORIENTATION: ENDIANESS

The big/little-endian (BLE) bit of the MAC control (00h) CSR register defines the endianess with which the MAC handles each 32-bit word transaction made by the BCU to/from the 8kB data buffer memory. The BLE bit defaults to 0, causing the MAC to consider each 32-bit word to be represented in little-endian byte order. If BLE is set to 1, the MAC considers each 32-bit word to be represented in big-endian byte order. The endianess defined by the BLE bit applies to all word transactions made by the BCU between the MAC and buffer memory, including transmit and receive status words reports made to the respective data buffers. Since the DS80C400 CPU can only access the data buffer memory one byte at a time using the MOVX operation, little-endian byte ordering (BLE = 0) generally allows for the most efficient buffer handling routines by the CPU. The following figure illustrates the two endian alternatives.

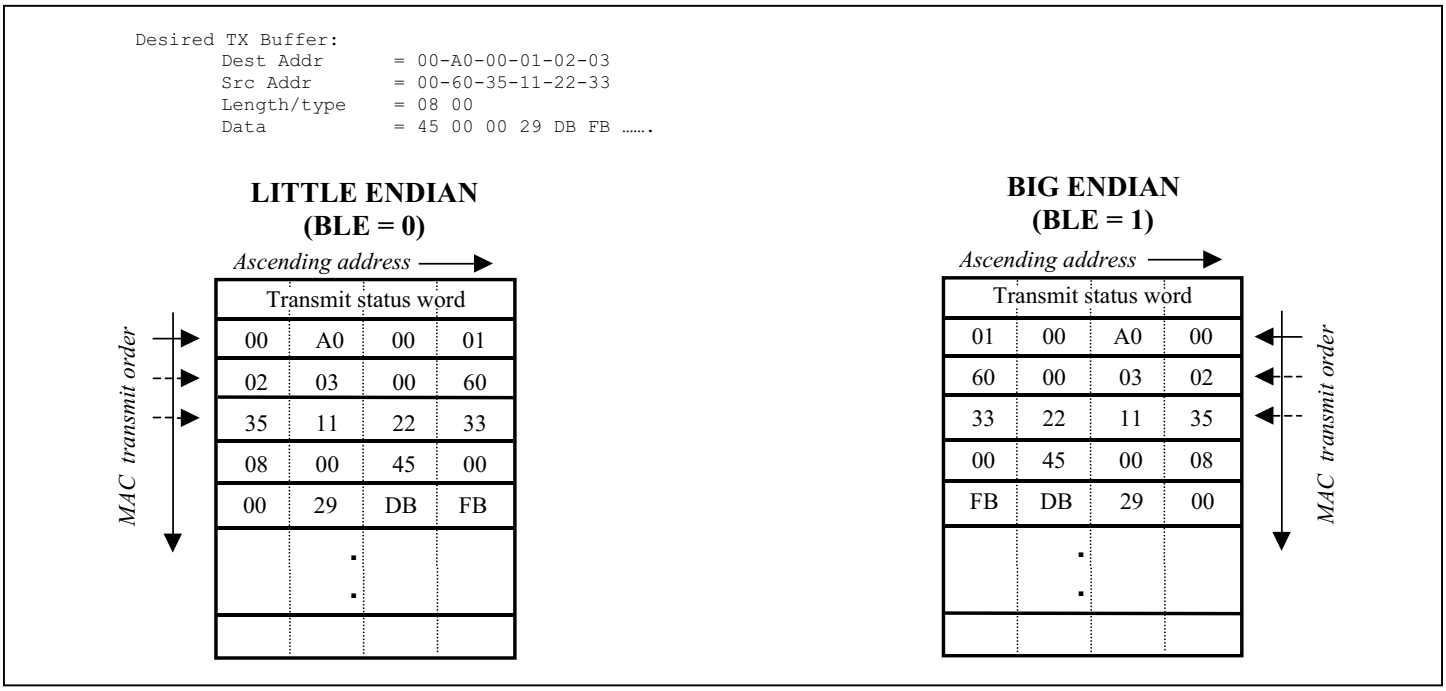


Figure 22-11. Big/Little ENDIAN Data Buffers

TRANSMITTING DATA

To transmit data over the MII or ENDEC interface, the CPU needs only to load the data buffer, supply size, and location of the buffer, and submit a transmit request to the buffer control unit (BCU) of the Ethernet controller. Data buffer memory should first be loaded with the desired transmit data. After the data buffer has been loaded, the BCUD (E6h) and BCUC (E7h) SFR pair are used to communicate size/location information and submit a transmit request to the BCU. The BCUD SFR needs to be written with the following information about the data to be transmitted, in the following order: 1) MSByte of the length of the data, 2) LSByte of the length of data, and 3) starting page for the data. Once this information has been written, the buffer command BC3:0 bits of the BCUC register can be written with one of the three possible transmit requests: 0100b = normal, 0101b = disable automatic padding, or 0110b = disable FCS field generation. Once the transmit request is submitted to the BCU, additional transmit requests submitted by writes to the BCUD, BCUC registers are intentionally ignored (dropped) by the BCU until the current transmission completes or aborts. Data buffer memory remains accessible during transmission so that additional transmit/receive data buffers can be loaded or unloaded. When the transmission has completed or has been aborted, the BCU returns a transmit status word for the given data buffer in the first word (32 bits) of the starting page originally specified for the buffer. The transmit interrupt flag (TIF) is set after the transmit status word has been written and an interrupt request to the CPU is generated, if the Ethernet activity interrupt source is enabled. A simple flow diagram for the transmit process is provided in Figure 22-12.

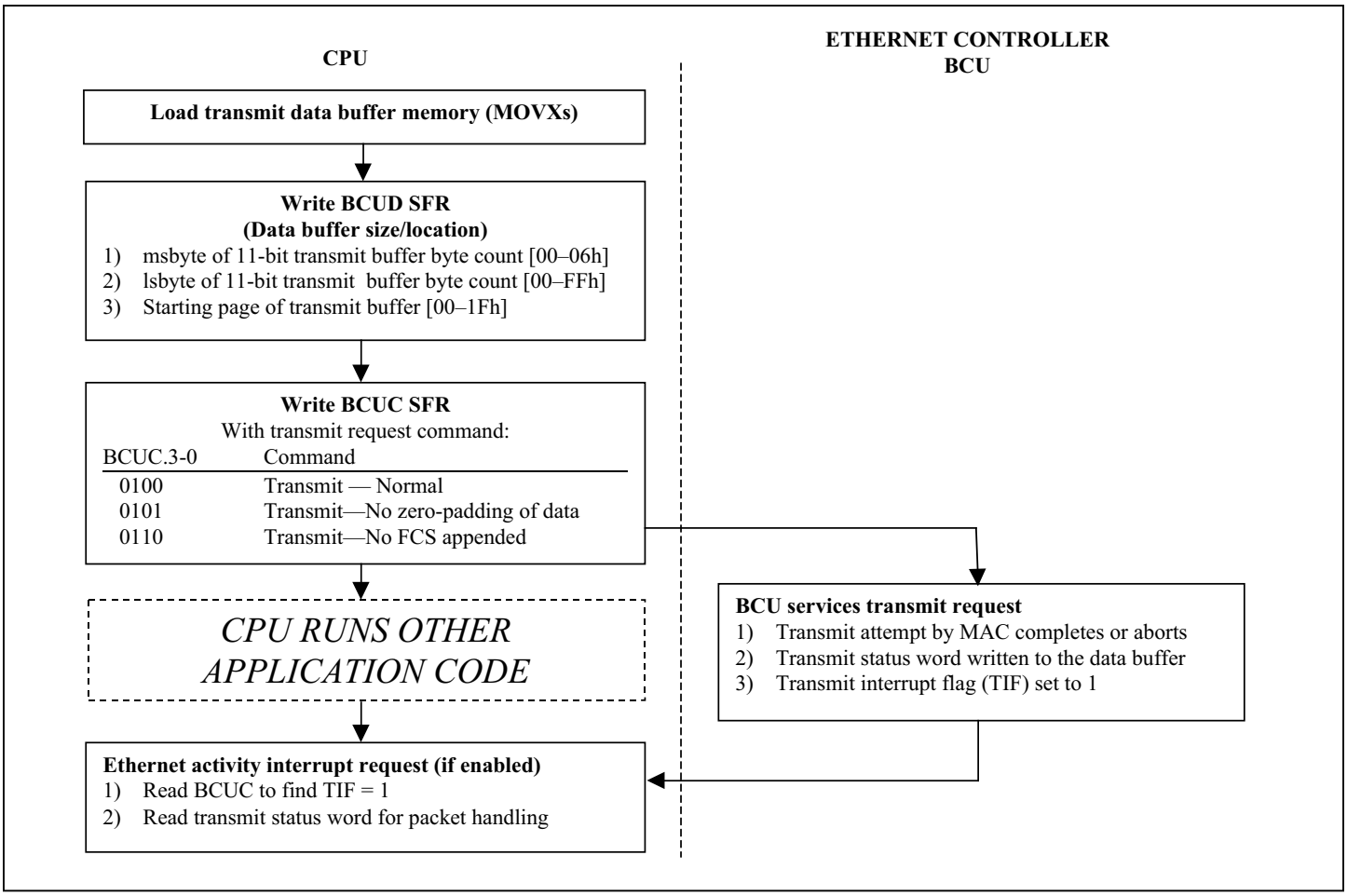


Figure 22-12. Transmit Flow Diagram

RECEIVING DATA

After configuring the Ethernet MAC and the defining the receive buffer size, reception, and storage of data from the MII or ENDEC, the interface does not require CPU intervention. The MAC operates per the settings specified in the CSR registers. The BCU automatically stores received data and receive status words to the receive data buffer when open pages are available and updates the receive FIFO. The CPU needs only to process receive data buffer entries and maintain sufficient open pages in the receive data buffer for additional packet storage by the BCU. From the CPUs perspective, the receive operation begins when the receive interrupt flag (RIF) is set by the BCU. If the Ethernet activity interrupt source is enabled, this event generates an interrupt request to the CPU. If the Ethernet activity interrupt source is not enabled, some polling scheme must be used to determine when an Ethernet packet has been received. The CPU then reads the BCUD SFR to acquire packet size/location information from the receive FIFO. Using this information, the CPU can locate the receive status word for the packet and process the packet accordingly. Once the CPU finishes processing the receive data buffer, it must invalidate the current receive packet in order to release the associated receive buffer pages for future use. Once the current receive packet has been invalidated, the next entry in the receive FIFO (if not empty) can then be accessed through the BCUD SFR. Note that the RIF flag serves as an indicator as to whether the receive FIFO is empty and therefore should always be cleared before invalidating the current receive packet to prevent missing an RIF = 1 condition. As an alternate means to free receive data memory, the receive FIFO and all receive buffer pages can be flushed by issuing the flush receive buffer command. A simple flow diagram for the receive process is provided in the following figure.

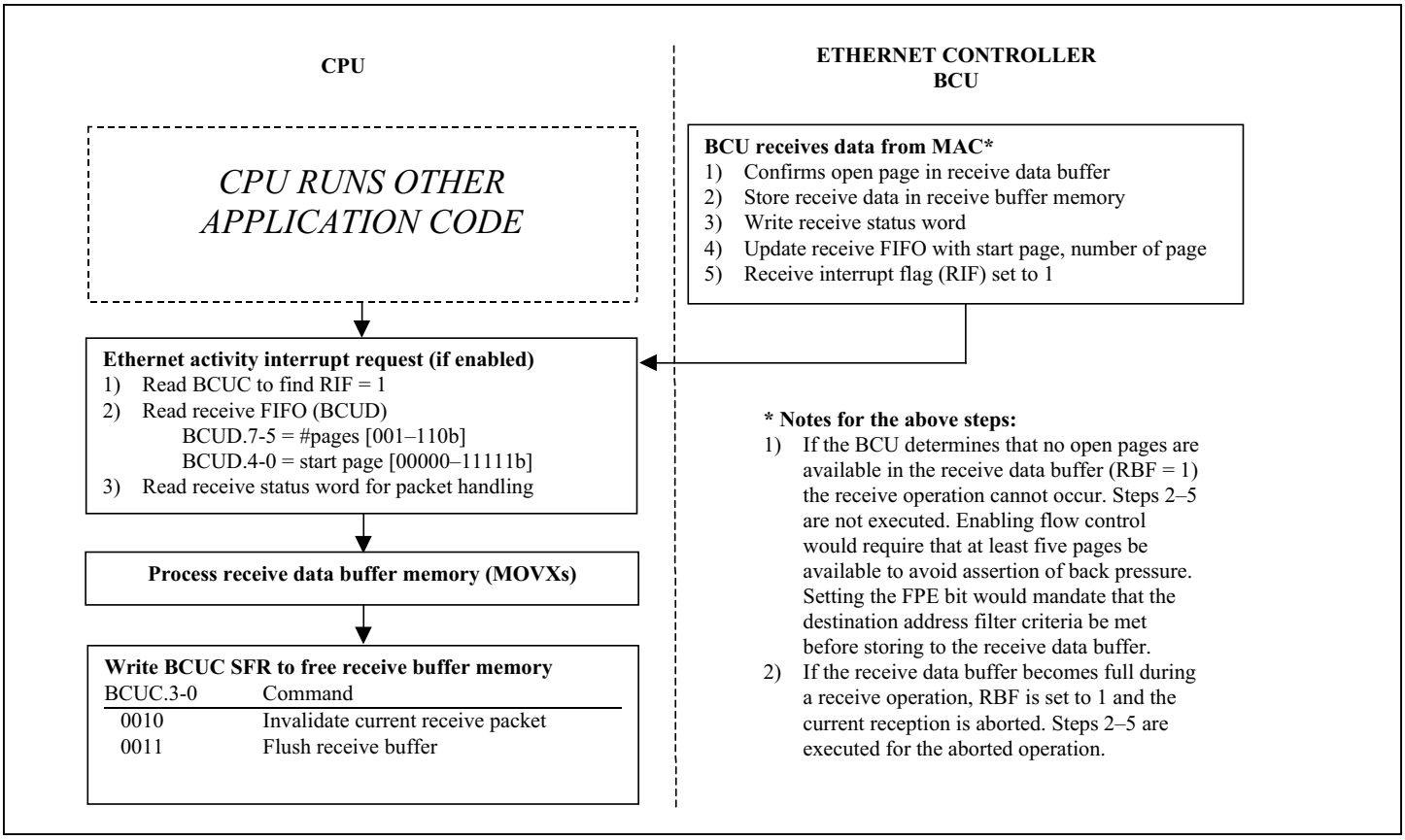


Figure 22-13. Receive Flow Diagram

USING WAKE-UP FRAMES

As discussed in the DS80C400 data sheet, the Ethernet controller can be placed into a low-power sleep mode such that it can be awakened by a user-programmable network wake-up frame and/or magic packet. The magic packet must conform to a specific predefined frame format, while the user-programmable network wake-up frame gives the application flexibility in defining the wake-up frame.

MAGIC PACKET MODE

When put into the magic packet mode, the power-management block constantly monitors each frame addressed to the device for a

specific magic packet pattern. Only packets that pass the current destination address filter are checked against the magic packet requirement. Each frame received is checked for an FFFF_FFFF_FFFFh pattern after the destination and source address fields. This pattern must be followed by 16 repetitions of physical MAC address without any breaks or interruptions. In case of a break in the 16 repetitions of the MAC physical address, the FFFF_FFFF_FFFFh pattern is scanned for again in the frame. The 16 repetitions can be anywhere in the frame, but they must be preceded by the synchronization stream.

For example, if the MAC address of a device were 00-11-22-33-44-55, then the MAC would scan for the following frame data sequence:

[DestinationAddress][SourceAddress] .....FFFF FFFF FFFF 0011 2233 4455 0011 2233 4455

0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455

0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455 0011 2233 4455

0011 2233 4455 0011 2233 4455 .....CRC

NETWORK WAKE-UP FRAME

Before putting the MAC into sleep mode, the host provides a list of sample frames and the corresponding byte masks to the wake-up frame filter register in the power-management block through the wake-up frame filter (28h) register. In order to know which bytes of each receive frame should be included in the CRC-16 calculation, the power-management block uses a programmable byte mask and a programmable pattern offset for each of the supported filters. In order to load the wake-up frame filter register, the host has to perform eight writes to the wake-up frame filter register. The wake-up frame filter structure is illustrated in the DS80C400 data sheet. The first write loads the filter 0 byte mask, the second write loads the filter 1 byte mask and so on. The fifth write loads the filter command nibble for each frame filter. Bit 3 of each filter [0,1,2,3] command nibble, wake source (WS), is used in conjunction with the global unicast (GU) bit of the wake-up events control and status (2Ch) CSR register to define what type of frames are compared to the wake-up frame filters. Bit 0 of each filter [0,1,2,3] command nibble enables the associated wake-up frame filter. The following table summarizes the possible WS, GU bit combinations. Note that a broadcast frame is always compared versus the enabled wake-up frame filters. One option to easily allow any broadcast frame to serve as a network wake-up frame is to disable all filter byte mask bits (= 00_00_00_00h) and designate that the required filter CRC-16 be 0000h (default CRC-16 value). Following the table is an example sequence for programming wake-up filter 0 and an accompanying diagram showing the resultant filter.

Table 22-3. Network Wake-Up Frame Patterns

GU, WS	NETWORK WAKE-UP FRAME CRITERIA (IF FRAME FILTER ENABLED)
00	Unicast, pass destination address filter, pass wake-up frame filter
01	Multicast, pass destination address filter, pass wake-up frame filter
10	Unicast, pass wake-up frame filter
11	Multicast, pass wake-up frame filter
xx	Broadcast, pass wake-up frame filter

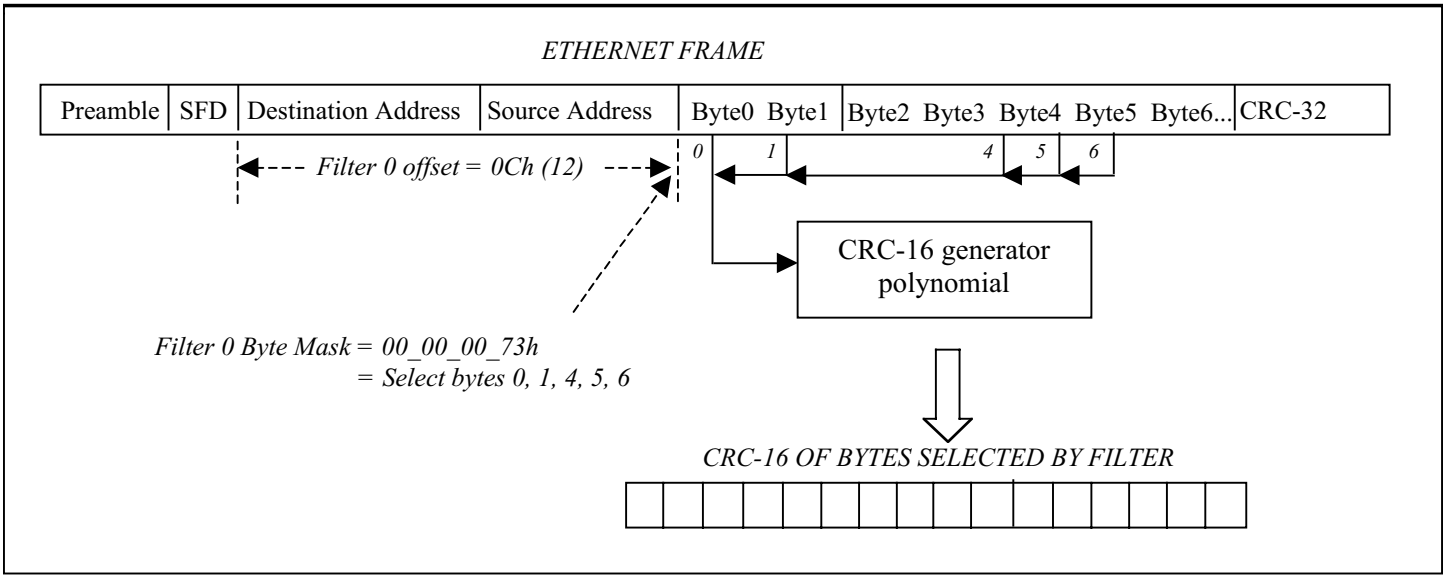


Figure 22-14. Wake-Up Frame Filter 0 Programming Example

1.	CSR write register (CSRA = 28h, CSRD = 00_00_00_73h)	*Filter 0 byte mask*
2.	CSR write register (CSRA = 28h, CSRD = xx_xx_xx_xxh)	*don't care*
3.	CSR write register (CSRA = 28h, CSRD = xx_xx_xx_xxh)	*don't care*
4.	CSR write register (CSRA = 28h, CSRD = xx_xx_xx_xxh)	*don't care*
5.	CSR write register (CSRA = 28h, CSRD = x0_x0_x0_x1h)	*Filter 0–enabled, unicast only*
6.	CSR write register (CSRA = 28h, CSRD = xx_xx_xx_0Ch)	*Filter 0 offset = 12*
7.	CSR write register (CSRA = 28h, CSRD = xx_xx_[CRC-16]h)	*Filter 0 CRC-16*
8.	CSR write register (CSRA = 28h, CSRD = xx_xx_xx_xxh)	*don't care*

SECTION 23: EMBEDDED DS80C400 SILICON SOFTWARE

The DS80C400 silicon software has been designed and embedded into the DS80C400 to give developers a means to quickly and cost-effectively network-enable any given application. The DS80C400 ROM firmware implements three major components: full TCP/IP IPv4/v6 stack with industry standard/Berkeley socket interface, preemptive task scheduler, and NetBoot functionality. A basic summary of these components and a high-level overview of the DS80C400 silicon software capabilities can be found in the *DS80C400 data sheet*. Section 23 provides more detailed information on usage of the functions provided by the DS80C400 silicon software.

SERIAL LOADER

The DS80C400 silicon software contains a serial bootloader to support in-system programming of external memory. The serial loader can be invoked following any reset event, provided that certain conditions are met. First, the $\overline{EA}$ pin must be pulled high externally in order to enable internal program memory. Next, the bypass ROM (BROM) bit must be set to 0, such that the ROM code is executed and not bypassed. This condition is always true for a power-on reset, but otherwise depends on the previous (prior to reset) state of BROM, since the bit is unaffected by other reset sources. The BROM bit can serve to trigger a reset when changed from 0 to 1 by the application code. When the application writes BROM from 0 to 1, the application expressly wants to generate a reset and bypass the ROM code and, therefore, the serial loader is not accessible. When both of the previous conditions are true following a reset, $\overline{EA}$ = 1 and BROM = 0, the ROM code examines the state of port pin P1.7. If P1.7 is found to be a logic 0, the DS80C400 silicon software skips the serial loader code, and proceeds to its check of the NetBoot (P5.3) pin. If P1.7 is in the logic 1 state, the DS80C400 silicon software code enters the serial loader autobaud-rate detection code.

AUTOBAUD-RATE DETECTION

The serial loader can automatically detect certain external baud rates and configure itself to that speed. The auto-baud routine operates serial port 0 in asynchronous serial mode 1 (8-N-1 or 8 data bits, no parity, 1 stop bit) and monitors the receive pin (RXD0) for a <CR> character (0Dh) at a supported baud rate. The auto-baud routine uses 16-bit Timer 1 to measure the duration of the (0Dh) received data bits. This measured duration is translated into a corresponding reload value for timer 2, which is used by the ROM for serial port timing generation. The functionality was designed to for external clock rates from 3.680MHz to 75.000MHz and baud rates from 2400bps to 115,200bps. The auto-baud routine monitors for serial activity for at least 2 seconds. This monitoriing period depends upon the external clock rate and is longer for clock rates <75MHz. If the auto-baud routine does not detect any serial activity during this period, it aborts the auto-baud process and proceeds to the next decision point in the ROM code flow. If the auto-baud routine is successful, the TINI ROM transmits a startup banner and displays a command prompt similar to what is shown below.

```
DS80C400 Silicon Software—Copyright (C) 2002 Maxim Integrated Products
Detailed product information available at http://www.maxim-ic.com

Welcome to the TINI DS80C400 Auto Boot Loader 1.0.1
```

COMMAND LINE INTERFACE

The serial loader supports an easy-to-use, ASCII command line interface that responds to the commands that are detailed later. Some commands require arguments and some commands have optional arguments. In all cases, the arguments are expected to be hexadecimal numbers. The serial loader manages memory in 64kB blocks (or banks). The most significant 8 bits of the 24-bit memory address are used to define the bank index. Most commands apply to the selected bank.

COMMAND SUMMARY

B *bank*

Bank Select—Selects the given *bank* of memory as the target for subsequent serial loader operations.
Example: **B C0** (selects the 64kB bank of memory C00000h–CFFFFFh)

C *[begin address [length]]*

Calculates the CRC-16 (cyclic redundancy check) of *length* bytes within the currently selected memory bank, starting at *begin address*.
Examples: **C** (calculates the CRC over the full 64kB bank xx0000h–xxFFFFh)
C 1000 (calculates the CRC for the range xx1000h–xxFFFFh)
C 1000 200 (calculates the CRC for the range xx1000h–xx1200h)

D *[begin address [length]]*

Dumps the selected memory range from the currently selected bank in hex format.
Examples: **D** (dumps memory for the full 64kB bank xx0000h–xxFFFFh)

D 1000 (dumps memory for the range xx1000h–xxFFFFh)
D 1000 200 (dumps memory for the range xx1000h–xx1200h)

E

Exit the serial loader and proceed to the find user code routine.

F *byte [begin address [length]]*

Fills the selected memory range in the currently selected bank with *byte*. Fill works for SRAM banks only (not flash banks).

Examples: **F 00** (fills the full 64kB bank xx0000h–xxFFFFh with 00h data)
F FF 1000 (fills the range xx1000h–xxFFFFh with FFh data)

G

Go—Starts executing at address 0000h of the currently selected memory bank.

H, ?

Help—Prints loader version and selected bank.

L

Load hex—Load standard ASCII Intel hex formatted data into the currently selected bank of external memory.

N

NetBoot—Transfers execution to the NetBoot code.

V

Verify—Verifies the incoming hex records versus the memory contents of the currently selected bank.

X *[offset]*

Execute—Executes the user code starting at *offset* in the currently selected bank. When the optional offset argument is not provided, a default offset of 0h is used. When an invalid offset is provided, the command is ignored.

Examples: **X** (executes code starting at address xx0000h)
X 1000 (executes code starting at address xx1000h)

Z *bank*

Zap—Erases the specified bank of flash memory. Zap should only be used on flash banks (not SRAM banks).

EXPORTED ROM FUNCTIONS

The DS80C400 makes many of its embedded ROM functions accessible to the user application. In the following pages, the exported ROM functions have been grouped and are described in the following order:

- Utility
- Memory manager
- Socket
- DHCP
- TFTP
- Task scheduler and hooks
- 1-Wire master
- Initialization
- Other

A complete listing of the exported ROM functions and associated function index numbers is located in the *DS80C400 data sheet*.

UTILITY FUNCTIONS

crc16

Description: **int crc16**(
int *crc*, /* initial CRC value */
unsigned char *value*); /* value to include in the CRC calculation */

The **crc16** function computes the CRC-16 of a byte given an initial CRC value.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*value*	R1:R0	CRC-16 return value
R1:R0	*crc*		

```
Example:  MOV R1, #0
          MOV R0, #0
          MOV DPTR, #bytelocation
          MOVX A, @DPTR
          ROMCALL crc16
```

mem_clear

Description: **void mem_clear(**
void ****target*,** /* pointer to start of *target* memory to clear */
int *length*); /* *length* (in number of bytes) to clear */

The **mem_clear** function clears a block of memory of *length* beginning at starting address specified by **target*.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
B:ACC	*length*	-	
DPTR0	**target*		

```
Example:  MOV B, #high(MEMLENGTH)
          MOV A, #low(MEMLENGTH)
          MOV DPTR, #MEMTOCLEAR
          ROMCALL mem_clear
```

mem_copy

Description: **void mem_copy(**
**void ****source,* /* pointer to the start of the *source* buffer */
void **target*, /* pointer to the start of the *target* buffer*/
int *length*); /* *length* of data to be copied */

The **mem_copy** function copies a block of memory. If the *source* and *target* memory buffers overlap, there is no guarantee that the *source* bytes are not overwritten prior to being copied to the *target.*

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
B:ACC	*length*	DPTR0	**source+length*
DPTR0	**source*	DPTR1	**target+length*
DPTR1	**target*		

```
Example:  MOV B, #high(MEMLENGTH)
          MOV A, #low(MEMLENGTH)
          MOV DPTR, #SOURCE
          INC DPS
          MOV DPTR, #TARGET
          INC DPS
          ROMCALL mem_copy
```

mem_compare

Description: **int mem_compare(**
void **block0,* /* pointer to the start of block0 */
void **block1,* /* pointer to the start of block1 */
int *length*); /* *length* of data to be compared */

The **mem_compare** function compares two blocks of memory, *block0* and *block1*, for *length* bytes. This function returns 0 in the accumulator if the two memory blocks are identical, nonzero otherwise.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
B:ACC	*length*		
DPTR0	**block0*	ACC	Return value (= 0 if equal)
DPTR1	**block1*		

```
Example:  MOV B, #high(MEMLENGTH)
          MOV A, #low(MEMLENGTH)
          MOV DPTR, #BLOCK0
          INC DPS
          MOV DPTR, #BLOCK1
          INC DPS
          ROMCALL mem_compare
```

add_dptr0/add_dptr1

Descriptions:

```
void add_dptr0(
void *dptr0,          /* dptr0 pointer */
int value);           /* value to be added to pointer */
void add_dptr1(
void *dptr1,          /* dptr1 pointer */
int value);           /* value to be added to pointer */
```

The **add_dptr0/add_dptr1** function adds a *value* to current *dptr0* or *dptr1*, depending upon which function is called.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
B:ACC	*value*		
DPTR0 or	**dptr0* (for **add_dptr0**) or	DPTR0 or	Pointer return value
DPTR1	**dptr1* (for **add_dptr1**)	DPTR1	

```
Example:  MOV B, #01h
          CLR A
          MOV DPTR, #STARTPTR
          ROMCALL add_dptr0
```

sub_dptr0/sub_dptr1

Descriptions:

```
void sub_dptr0(
void *dptr0,          /* dptr0 pointer */
int value);           /* value to be subtracted from pointer */
void sub_dptr1(
void *dptr1,          /* dptr1 pointer */
int value);           /* value to be subtracted from pointer */
```

The **sub_dptr0/sub_dptr1** function subtracts a *value* from current *dptr0* or *dptr1*, depending upon which function is called.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
B:ACC	*value*		
DPTR0 or	**dptr0* (for **sub_dptr0**) or	DPTR0 or	Pointer return value
DPTR1	**dptr1* (for **sub_dptr1**)	DPTR1	

```
Example:  MOV B, #01h
          CLR A
          MOV DPTR, #ENDPTR
          ROMCALL sub_dptr0
```

getpseudorandom

Description: **unsigned char getpseudorandom(void);**

The **getpseudorandom** function gets a pseudorandom byte from a CRC function.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
—		ACC	Return value

```
Example:  ROMCALL getpseudorandom
```

MEMORY MANAGER FUNCTIONS

rom_kernelmalloc

Description: **void rom_kernelmalloc(**
int *blocksize*); /* requested memory *blocksize* */

The **rom_kernelmalloc** function allocates fast kernel memory. It allocates a block from the kernel memory pool without incurring the overhead of the regular memory manager. This function returns 0 in the accumulator if successful, nonzero otherwise. It is exported by the DS80C400 silicon software, and also serves as the default **kernelmalloc** in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R3:R2	*blocksize*	ACC R3:R2 DPTR0	Return value (= 0 for success) Memory block handle Pointer to memory block

```
Example: MOV R3, #high(MEMLENGTH)
         MOV R2, #low(MEMLENGTH)
         ROMCALL rom_kernelmalloc
```

rom_kernelfree

Description: **void rom_kernelfree**(
int blockhandle); /* blockhandle of memory to free. */

The **rom_kernelfree** function frees fast kernel memory. With blockhandle, this function frees a block of memory that had previously been allocated by kernelmalloc. It returns 0 in the accumulator if successful, nonzero otherwise. This function is exported by the DS80C400 silicon software, and also serves as the default kernelfree in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R3:R2	*blockhandle*	ACC	Return value (= 0 for success)

```
Example: MOV R3, #high(MEMHANDLE)
         MOV R2, #low(MEMHANDLE)
         ROMCALL rom_kernelfree
```

rom_malloc

Description: **void rom_malloc(**
int *blocksize*); /* requested memory *blocksize* */

The **rom_malloc** function allocates memory from the heap and clears the allocated memory. It returns 0 in the accumulator if successful, nonzero otherwise. This function is exported by the DS80C400 silicon software, and also serves as the default **malloc** in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R3:R2	*blocksize*	ACC R3:R2 DPTR0	Return value (= 0 for success) Memory block handle Pointer to memory block

```
Example: MOV R3, #high(MEMLENGTH)
         MOV R2, #low(MEMLENGTH)
         ROMCALL rom_malloc
```

rom_malloc_dirty

Description: **void rom_malloc_dirty(**
int *blocksize*); /* requested memory *blocksize* */

The **rom_malloc_dirty** function allocates memory from the heap, but does not clear the allocated memory. It returns 0 in the accumulator if successful, nonzero otherwise. This function is exported by the DS80C400 silicon software, and also serves as the default **mallocdirty** in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R3:R2	*blocksize*	ACC R3:R2 DPTR0	Return value (= 0 for success) Memory block handle Pointer to memory block

Example:	MOV R3, #high(MEMLENGTH)
	MOV R2, #low(MEMLENGTH)
	ROMCALL rom_malloc_dirty		

rom_free

Description: **void rom_free(**
int *blockhandle*); /* *blockhandle* of memory to free. */

The **rom_free** function frees a block of memory with a *blockhandle* that had previously been allocated with **rom_malloc** or **rom_malloc_dirty**. This function returns 0 in the accumulator if successful, nonzero otherwise. It is exported by the DS80C400 silicon software, and also serves as the default free in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R3:R2	*blockhandle*	ACC	Return value (= 0 for success)

Example:	MOV R3, #high(MEMHANDLE)
	MOV R2, #low(MEMHANDLE)
	ROMCALL rom_free		

rom_deref

Description: **void rom_deref(**
int *blockhandle*); /* *blockhandle* of memory to derefence */

The **rom_deref** function dereferences a memory *blockhandle* into an absolute address pointer. It returns 0 in the accumulator if successful, nonzero otherwise. This function is exported by the DS80C400 silicon software, and also serves as the default **deref** in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R3:R2	*blockhandle*	ACC DPTR0	Return value (= 0 for success) Pointer to memory block

Example:	MOV R3, #high(MEMHANDLE)
	MOV R2, #low(MEMHANDLE)
	ROMCALL rom_deref		

rom_getfreeram

Description: **void rom_getfreeram(void);**

The **rom_getfreeram** function returns the amount of memory that is still available in the heap. This function returns 0 in the accumulator if successful, nonzero otherwise. It is exported by the DS80C400 silicon software, and also serves as the default **getfreeram** in the function redirect table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC R3:R0	Return value (= 0 for success) Amount of free memory
Example:	ROMCALL rom_getfreeram		

SOCKET FUNCTION CALLING CONVENTIONS

The DS80C400 silicon software socket functions conform to the TINI Native Library calling (NatLib) conventions. The NatLib calling conventions are described as follows.

Input Parameter Buffer:

The input parameter buffer consists of up of six parameters (or arguments). Each parameter is 4 bytes wide; thus, a parameter buffer spans 24 total bytes. Parameter 0 would be at offset 0 in the parameter buffer, parameter 1 at offset 4, etc., as shown here:

Parameter buffer:

Parameter0 (4 bytes) ⟹ Param0[0], Param0[1], Param0[2], Param0[3]

Parameter1 (4 bytes) ⟹ ….

Parameter2 (4 bytes) ⟹ ….

Parameter3 (4 bytes) ⟹ ….

Parameter4 (4 bytes) ⟹ ….

Parameter5 (4 bytes) ⟹ Param5[0], Param5[1], Param5[2], Param5[3]

Each argument passed to a socket function, no matter whether it is an 8-bit byte or 24-bit pointer, occupies a single parameter of the parameter buffer as described:

Integer representation. 8-bit, 16-bit, and 32-bit values are stored LSB first in the corresponding parameter.

Pointer representation. Pointers are stored LSB first in bytes 0, 1, and 2 of the corresponding parameter. When a pointer to an array is given, that data array should be of the format: type (one byte: *ignored*), array length (two bytes: LSB, MSB), and array data (MSB: LSB bytes).

Prior to calling a given socket function, a parameter buffer should be loaded accordingly and a pointer to that parameter buffer should be placed in working registers R7, R6, and R5 of working register bank 2 (R7_B2:R5_B2).

Output Return Values:

Every socket function returns a success/failure indication in the accumulator (ACC). Success is signaled by ACC = 0. Socket functions that return 8-bit, 16-bit, or 32-bit values store this value/pointer most significant byte first in working registers R3, R2, R1, and R0 of working register bank 0 (R3_B0:R0_B0). The following registers potentially are used by the socket functions: ACC, B, DPTR0 (DPX:DPH:DPL), DPTR1 (DPX1:DPH1:DPL1), DPTR2 (DPX2:DPH2:DPL2), DPS, PSW, and register banks 0, 1, 2, and 3 (i.e., the user must take care to save these registers across socket function calls when necessary).

SOCKET FUNCTIONS/POINTERS

PARAMBUFFER

The DS80C400 silicon software exports a pointer the six-argument parameter buffer (PARAMBUFFER) that is used by the DHCP task for its socket function calls. The user application is free to use this parameter buffer, but should recognize that, if more than one task needs to use a parameter buffer, separate buffers must be declared for each task or the PARAMBUFFER must be protected from concurrent access.

socket

Description: **int socket**(

int *domain,* /* currently ignored */

int *type*, /* *type* of socket (UDP = 00h or TCP = 01h) */

int *protocol*); /* currently ignored */

The **socket** function creates a network socket (a local endpoint) for TCP or UDP communication. The *type* can either be SOCK_STREAM for TCP sockets or SOCK_DGRAM for UDP sockets. The *domain* and *protocol* parameters are ignored. The **socket** function returns a socket handle (i.e., an identifier for the new socket), but has no specific local address assigned to it. To use it as a server socket, the use of **bind** is required. To use a streaming (TCP) socket, the socket must be connected using either **connect** or **listen/accept**. To destroy/free a socket, use **closesocket**. This is the only function that returns a socket number. All other socket functions require the socket number to be passed to them to access the correct socket.
Note: The **socket** function calls gettaskID through the function redirect table to get the current task ID.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
domain	Param0–currently ignored		
type	Param1[0]–00h = UDP 01h = TCP	ACC R0	Return value (= 0 for success) Socket handle
protocol	Param2–currently ignored		

closesocket

Description: **int closesocket(**
int *s*); /* closes socket with handle = *s* */

The **closesocket** function closes a socket. It closes the socket having *handle*, which was created by the socket call and returns a success/failure code in the accumulator.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #	ACC	Return value (= 0 for success)

sendto

Description: **int sendto(**
int *s*, /* send to socket with handle *s* */
void ****buf*,** /* datagram data array at **buf* */
int *len*, /* size of datagram to be sent */
int *flags*, /* currently ignored */
struct sockaddr **addr*, /* target address */
int *addrlen*); /* size of address structure */

The **sendto** function sends a UDP datagram to the specified address. The target address is specified at **addr; addrlen* is the size of the sockaddr structure. The datagram itself is referenced by **buf*, and *len* is the size of the datagram. The *flags* parameter is ignored. The **sendto** function is unable to detect whether the datagram has successfully reached the destination, and only returns a failure code on local errors. Use **bind** to specify a local port number. Without **bind, sendto** chooses a random local port.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**buf*	Param1[0:2]–pointer to UDP data		
len	Param2[0:1]–size of datagram	ACC	Return value (= 0 for success)
flags	Param3–currently ignored		
**addr*	Param4[0:2]–pointer to target addr		
addrlen	Param5[0]–size of addr structure		

recvfrom

Description: **int recvfrom(**
int *s*, /* receive on socket with handle *s* */
void **buf*, /* datagram data array at **buf* */
int *len*, /* size of datagram received */
int *flags*, /* currently ignored */
struct sockaddr **addr*, /* remote address */

int *addrlen*); /* size of address structure */

The **recvfrom** function receives a UDP datagram. This function receives a message on socket *s*, storing the message at **buf*. *len* is the size of *buf*. If addr is not NULL, the remote address is filled in at **addr*. The fla*gs* parameter is ignored. **recvfrom** returns the number of bytes read. If no data is available on the socket, **recvfrom** blocks for the amount of time specified with the **setsockopt/SO_TIMEOUT** call. **Note:** It is generally required to use **bind** in order to first assign a local port to the socket.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**buf*	Param1[0:2]–pointer to UDP data		Return value (= 0 for success)
len	Param2[0:1]–size of datagram	ACC	Number of bytes received
flags	Param3–currently ignored	R1:R0	
**addr*	Param4[0:2]–pointer to target addr		
addrlen	Param5[0]–size of addr structure		

connect

Description: **int connect**(
int *s*, /* socket to connect to a specific address */
struct sockaddr **addr*, /* address to which socket *s* should be connected */
int *addrlen*); /* size of address structure */

The **connect** function connects a TCP socket to the specified address. This function connects the sockets to the remote address specified by the addr structure and returns a success/failure code in the accumulator. connect can only be used once with each socket.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**addr*	Param1[0:2]–pointer to target addr	ACC	Return value (= 0 for success)
addrlen	Param2[0]–size of addr structure		

bind

Description: **int bind**(
int *s*, /* socket to bind to specified address/port */
struct sockaddr **addr*, /* address to which socket *s* should be bound */
int *addrlen*); /* length of *addr* structure */

The **bind** function binds a socket to the specified address. This function assigns a local address and port at **addr* to the socket *s*. The combination of IP address and port is often referred to as "name". Binding a socket is necessary for server sockets. For client sockets, use **bind** if a specific source port is requested. *s* is the socket to be assigned a local name, *addr* contains the local address (IP and port values). **bind** returns a success/failure code in the accumulator.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**addr*	Param1[0:2]–pointer to target addr	ACC	Return value (= 0 for success)
addrlen	Param2[0]–size of addr structure		

listen

Description: **int listen**(
int *s*, /* socket on which to listen for connections */
int *backlog*); /* maximum queue length for pending connections */

The **listen** function listens for connections on the specified socket. It creates a queue of length *backlog*; *backlog* is the maximum number of pending new incoming connections (max.16). This function returns a success/failure code. To move an incoming connection request from the queue to an established state, **accept** must be called. It is generally required to use **bind** to assign a local name to a socket before invoking **listen**.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #	ACC	Return value (= 0 for success)
backlog	Param1[0]–max queue backlog		

accept

Description: **int accept**(

int *s*, /* socket on which to accept a pending connection */

struct sockaddr **addr*, /* address for new socket connection */
int *addrlen*); /* length of *addr* structure */

The **accept** function accepts a TCP connection on the specified socket. This function moves the first pending new incoming connection request from the listen queue into the established state. It assigns a new local socket (connection endpoint) to the connection and returns its socket handle. **Accept** blocks if there are no pending new incoming connection requests. The **socket** *s* must first be created by the **socket** call, bound to an address-using bind, and a listen queue must be created for it using **listen**. If *addr* is not NULL, the remote address is filled in.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #	ACC	Return value (= 0 for success)
**addr*	Param1[0:2]–pointer to addr	R0	New socket handle #
addrlen	Param2[0]–size of addr structure		

recv

Description: **int recv**(

int *s*, /* socket from which to receive data */
void **buf*, /* pointer to receive data buffer */
int *len*, /* maximum number of bytes to receive */
int *flags*); /* currently ignored */

The **recv** function reads data from a connection-oriented (TCP) socket. This function reads up to *len* bytes from the socket *s*, which must be in a connected state, into the buffer *buf*. It returns the number of bytes read. The *flags* parameter is ignored. If there is no data on the socket, recv blocks infinitely unless a socket timeout is set using **setsockopt**.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**buf*	Param1[0:2]–pointer to TCP data	ACC	Return value (= 0 for success)
len	Param2[0:1]–max bytes to read	R1:R0	Number of bytes read
flags	Param3–currently ignored		

send

Description: **int send**(

int *s*, /* socket on which to send data */
void **buf*, /* pointer to transmit data buffer */
int *len*, /* number of bytes to send */
int *flags*); /* currently ignored */

The **send** function writes data to a connection-oriented (TCP) socket. It writes *len* bytes from the buffer *buf* to the socket s, which must be in a connected state. The *flags* parameter is ignored. **Send** returns only a local success/failure code and does not necessarily detect transmission errors.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**buf*	Param1[0:2]–pointer to TCP data	ACC	Return value (= 0 for success)
len	Param2[0:1]–# bytes to send		
flags	Param3–currently ignored		

getsockopt/setsockopt

Descriptions: **int getsockopt(**

int *s*, /* socket for which to get option */
int *level*, /* currently ignored */
int *name*, /* option to get */
void **buf*, /* buffer to which socket option data is written */
int *len*); /* length of *buf* */

```
int setsockopt(
int s,              /* socket for which to set option */
int level,          /* currently ignored */
int name,           /* option to set */
void *buf,          /* buffer containing socket option data to set */
int len);           /* length of buf */
```

The **getsockopt** and **setsockopt** functions get and set socket options. These functions get/set various options of a socket*s*. The option to be read/written is specified by the *name* parameter. The **buf* parameter points to the buffer to be filled (get operation) or the option value to be written (set operation). *len* is the size of the buffer and modified to the size of the filled-in data by **getsockopt**. *buf* can be NULL if a socket option needs no data. The *level* parameter is ignored. **getsockopt** and **setsockopt** return a success/failure code in the accumulator.

The following option names are supported:

NAME	VALUE	DESCRIPTION
TCP_NODELAY	0	Gets/sets the TCP Nagle parameter
SO_LINGER	1	(ignored)
SO_TIMEOUT	2	Gets/sets the socket timeout
SO_BINDADDR	3	Gets the local socket IP (get operation only)

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
level	Param1–currently ignored		
name	Param2[0]–option to get/set	ACC	Return value (= 0 for success)
**buf*	Param3[0:2]–pointer to buf		
len	Param4–length of buf		

getsockname

Description:

```
int getsockname(
int s,              /* socket for which to get local IP address and port */
struct sockaddr *addr,  /* address where IP address and port should be stored */
int addrlen);       /* size of addr structure */
```

The **function** getsockname returns the local IP and port of the socket *s* and stores it in the *addr* structure. It returns a success/failure code in the accumulator.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**addr*	Param1[0:2]–pointer to addr	ACC	Return value (= 0 for success)
addrlen	Param2[0]–size of addr structure		

getpeername

Description:

```
int getpeername(
int s,              /* socket for which to get the remote IP address */
struct sockaddr *addr,  /* address where IP address should be stored */
int addrlen);       /* length of addr structure */
```

The **getpeername** function returns the remote address of a connection-oriented (TCP) socket. If socket *s* is connected, **getpeername** stores the remote address into the *addr* structure. This function returns a success/failure code in the accumulator.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**addr*	Param1[0:2]–pointer to addr	ACC	Return value (= 0 for success)
addrlen	Param2[0]–size of addr structure		

cleanup

Description:

```
int cleanup(
int pid);           /* task ID of terminated task */
```

The **cleanup** function closes all sockets associated with a task ID. The user's task manager should call this function when-

ever a task dies so as to ensure all associated resources are freed by the socket layer. *pid* is the task ID of the terminated task. cleanup returns a success/failure code in the accumulator. **Note:** The DS80C400 silicon software task scheduler does not call this function. The user should call **cleanup** after each **task_kill** call.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
pid	Param0[0]–task ID for which associated sockets should be closed	ACC	Return value (= 0 for success)

avail

Description:
```
int avail(
int s);                          /* socket to check availability of received bytes */
```
The **avail** function reports the number of bytes available for **recv** on a connection-oriented (TCP) socket.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #	ACC R1:R0	Return value (= 0 for success) Bytes available for reading

join/leave

Descriptions:
```
int join(
int s,                           /* socket to add to the multicast group */
struct sockaddr *addr,           /* address of specified multicast group */
int addrlen);                    /* length of addr structure */
int leave(
int s,                           /* socket to remove from the multicast group */
struct sockaddr *addr,           /* address of specified multicast group */
int addrlen);                    /* length of addr structure */
```
The **join and leave** functions add or remove, respectively, socket *s* to/from the specified multicast group. The multicast group name is specified by the *addr* structure. **Join** and **leave** return a success/failure code in the accumulator. **Note:** The current implementation does not support IPv6 multicasting.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
s	Param0[0]–socket handle #		
**addr*	Param1[0:2]–pointer to addr	ACC	Return value (= 0 for success)
addrlen	Param2[0]–size of addr structure		

ping

Description:
```
int ping(
struct sockaddr *addr,           /* address to ping */
int addrlen,                     /* size of addr structure */
int TTL,                         /* time-to-live */
unsigned char *response);        /* data returned in response to ping */
```
The **ping** function pings the specified address and returns the result. It sends an ICMP echo request, or ping, to a remote host specified by *addr*. The packets sent by **ping** have the specified time-to-live *(TTL*). **Ping** returns the response time and the buffer pointed to by **response* gets filled in with the returned data.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
**addr*	Param0[0:2]–pointer to addr		
addrlen	Param1[0]–size of addr structure	ACC	Return value (= 0 for success)
TTL	Param2[0]–time-to-live		
**response*	Param3[0:2]–pointer to return data		

getnetworkparams/setnetworkparams

Descriptions:
```
int setnetworkparams(
void *parameters);          /* pointer to network parameters buffer */
int getnetworkparams(
void *parameters);          /* pointer to network parameters buffer */
```

The **setnetworkparams** and **getnetworkparams** functions set/get the IPv4 address and configuration parameters. These functions allow the user to set/get the IPv4 portion of the network configuration (**Note:** the IPv6 address is autoconfigured). To autoconfigure IPv4, use the DHCP functionality instead of **setnetworkparams.** The address configured by DHCP can be read using **getnetworkparams.** Both functions return a success/failure code in the accumulator. Network *parameters* is a buffer containing the following data:

PARAMETER	OFFSET	LENGTH	DESCRIPTION
(zero)	0	12	Must be 0
IP4ADDR	12	4	IP address
IP4SUBNET	16	4	Subnet mask
IP4PREFIX	20	1	Number of 1 bits in subnet mask
(zero)	21	12	Must be 0
IP4GATEWAY	33	4	IP address of default gateway

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
**parameters*	Param0[0:2]–Pointer to *parameters* buffer	ACC	Return value (= 0 for success)

getipv6params

Descriptions:
```
getipv6params(
void *parameters);          /* pointer to IPv6 parameters buffer */
```

The **getipv6params** function returns the IPv6 address of the Ethernet interface to the *parameters* buffer. The *parameters* buffer contain the following data following the **getipv6params** function:

PARAMETER	OFFSET	LENGTH	DESCRIPTION
IP6ADDR	0	16	IP address
IP6PREFIX	16	1	IP prefix length

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
**parameters*	Param0[0:2]–Pointer to *parameters* buffer	ACC	Return value (= 0 for success)

getethernetstatus

Description: **int getethernetstatus(void)**;
The **getethernetstatus** function returns the Ethernet status (link). The return value is a bit-wise OR of the following flags:

FLAG*	VALUE	DESCRIPTION
ETH_STATUS_LNK	01h	Ethernet link status

*(*No other flags are currently defined.*)

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Return value (= 0 for success)
		R0	Return value (bit-wise OR of flags)

gettftpserver/settftpserver

Descriptions:
```
int gettftpserver(
struct sockaddr *addr,        /* address of TFTP server */
int *addrlen);                /* length of addr structure */
int settftpserver(
struct sockaddr *addr,        /* address of TFTP server */
int *addrlen);                /* length of addr structure */
```

The **gettftpserver** and **settftpserve**r functions get/set the IP address of the TFTP server. **gettftpserver** stores the address of the TFTP server into the *addr* structure. **settftpserver** sets the TFTP server address to the value supplied in *addr*. The **settftpserver** function must be used if the address of the TFTP server is not acquired by DHCP or 1-Wire. Both functions return a success/failure code in the accumulator.

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
*addr	Param0[0:2]–Pointer to addr	ACC	Return value (= 0 for success)
addrlen	Param1[0]–Size of addr structure		

DHCP FUNCTIONS

dhcp_init

Description: **int dhcp_init(void);**

The **dhcp_init** function initializes the DHCP client. **dhcp_init** starts a DHCP client task and returns to the caller. This function returns 0 in the accumulator if successful, nonzero otherwise. To read the address DHCP has config ured (only valid if DHCP is in bound, renewing or rebinding state—see **dhcp_status**), use the socket layer function **getnetworkparams.**

DHCP is implemented for IPv4 only. The IPv6 portion of the network stack uses neighbor discovery.

The DHCP client calls DHCPnotify from the function redirect table when it acquires or loses an IP. The default DHCPnotify routine supplied in the function redirect table (when ROM_redirect_init is executed by the DS80C400 silicon software) is **rom_dhcp_notify.**

INPUT	PARAMETER#–DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Return value (= 0 for success)
		R0	Return value (bit-wise OR of flags)
Example:	`ROMCALL dhcp_init`		

dhcp_setup

Description: The **dhcp_setup** function is used by **dhcp_init** and **dhcp_stop.**

dhcp_startup

Description: The **dhcp_startup** function is used by **dhcp_init.**

dhcp_run

Description: The **dhcp_run** function is used by **dhcp_init.**

dhcp_status

Description: **int dhcp_status(void)**;

The dhcp_status function returns the DHCP state. The possible DHCP state return values are listed as follows. See the RFC2131 for a description of these DHCP states.

DHCP STATE	RETURN VALUE
INIT	0
SELECTING	1
REQUESTING	2
INITREBOOT	3
REBOOTING	4
BOUND	5
RENEWING	6
REBINDING	7

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Return value
Example:	`ROMCALL dhcp_status`		

dhcp_stop

Description: **void dhcp_stop(void);**

The **dhcp_stop** function disables the DHCP functionality and kills the DHCP client task.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		-	
Example:	`ROMCALL dhcp_stop`		

rom_dhcp_notify

Description: **int rom_dhcp_notify(void)**

This function notifies of a DHCP state change and sends the TASK_DHCPSLEEP signal to the task originally called **dhcp_init()**. **rom_dhcp_notify()** is exported by the DS80C400 silicon software and serves as the default DHCPnotify in the function redirect table. The DHCP client calls **DHCPnotify** from the function redirect table when it acquires or loses an IP.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	New DHCP state	—	
Example: ROMCALL rom_dhcp_notify			

TFTP FUNCTIONS/POINTERS

TFTP_MSG

The DS80C400 silicon software exports a pointer to the data buffer that is configured by the tftp_first task, which is used by the TFTP session.

tftp_init

Description: **int tftp_init(void);**

The **tftp_init** function initializes the TFTP client. It sets up the data structures required for the TFTP client, most importantly the TFTP_MSG data buffer. This function returns 0 in the accumulator if successful, nonzero otherwise.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Return value (= 0 for success)
Example:	`ROMCALL tftp_init`		

tftp_first

Description: **int tftp_first(**

char **filename); /* pointer to *filename* for TFTP request */

The **tftp_first** function requests the first TFTP data block and waits for data. This function requests the file name specified at **filename* from the TFTP server and returns the number of bytes read along with a success/failure indication in the accumulator. If this number is less than 512, the TFTP transfer has ended.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
DPTR0	**filename*	ACC R1:R0	Return value (= 0 for success) #bytes read
Example:	`MOV DPTR, #FILENAME` `ROMCALL tftp_first`		

tftp_next

Description: **int tftp_next(**

int *ack_only); /* flag to signal when *ack_only* should be sent */

The **tftp_next** function acknowledges a TFTP data block and waits for data. **tftp_next**(0) returns subsequent data blocks (until the returned length is less than 512). Use *ack_only* <>0 (i.e., **tftp_next**(1)) to acknowledge the last data block without waiting for additional data.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*ack_only*	ACC R1:R0	Return value (= 0 for success) #bytes read
Example:	`MOV A, #0h` `ROMCALL tftp_next`		

tftp_close

Description: **int tftp_close**(void);

This function closes a tftp socket allocated by **tftp_first()**. **tftp_close()**. It frees the socket handle allocated by **tftp_first()** and should be called when a tftp data transfer has finished (or when a tftp data transfer has failed).

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	return value (=0 for success)
Example: ROMCALL tftp_close			

TASK SCHEDULER FUNCTIONS

task_genesis

Description: **void task_genesis(**

int *savesize);* /* buffer of *savesize* allocated to hold task state */

The **task_genesis** function sets up primordial and idle threads. This function creates the running task list and assigns the task ID 1 to the current flow of execution. It calls **malloc** to allocate a buffer of *savesize*, which is used to save and restore the task state. **Note:** This function does not change or enable the timer interrupt. Interrupt handlers must be installed and corresponding interrupts must be enabled before calling this function.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R1:R0	*savesize*	-	

```
Example:  MOV R1, #high(SAVESIZE)
          MOV R0, #low(SAVESIZE)
          ROMCALL task_genesis
```

task_getcurrent

Description: **int task_getcurrent(void);**

The **task_getcurrent** function returns the current task's ID.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Task ID

```
Example:  ROMCALL task_getcurrent
```

task_getpriority

Description: **int task_getpriority(**

int *id);* /* *id* of the task for which we want to get priority */

The **task_getpriority** function returns the priority of a task. It returns the priority of the task with the given *id*. The current task always has ID = 0. This function returns a success/failure code in the accumulator (ACC) SFR and the task priority in the B SFR.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*id*	ACC B	Success (= 00h) or failure code task priority

```
Example:  MOV A, #…
          ROMCALL task_getpriority
```

task_setpriority

Description: **int task_setpriority(**

int *id*, /* *id* of the task for which we want to set priority */

int *priority);* /* *priority* that we wish to assign to task with *id* */

The **task_setpriority** function changes a task priority. ID 0 means current task. The priority is a value in the range MIN_PRI ORITY MAX_PRIORITY, with an idle task running at MIN_PRIORITY, a regular task running at NORM_PRIORITY. This function returns a success/failure code in the accumulator (ACC) SFR.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC B	*id* *priority*	ACC	Success (= 00h) or failure code

```
Example:  CLR A
          MOV B, #NORM_PRIORITY-1
          ROMCALL task_setpriority
```

task_fork

Description: **int task_fork(**

int *priority,* /* priority that we wish to assign to the task */

int *savesize); /* buffer of savesize allocated */

The **task_fork** function creates and executes a new task. It creates a new task and links it into running task list with the given *priority.* This function creates a duplicate of the current task and assigns it a new task id. The new task returns with a zero id, the parent gets the child id as a return value. This function also returns a success/failure code in the accumulator (ACC). It calls **malloc** to allocate a buffer of *savesiz*e, which is used to save and restore the task's state. *priority* is a user-assigned value in the range MIN_PRIORITY...MAX_PRIORITY.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*priority*	ACC	Success (= 00h) or failure code
R1:R0	*savesize*	R0	Child task ID or 0h if child

Example:

```
; Create a DHCP task
MOV R1, #high(SAVESIZE)
MOV R0, #low(SAVESIZE)
MOV A, #NORM_PRIORITY
ROMCALL task_fork
JNZ dhcp_init_exit
; this is a child task - run the FSM
AJMP dhcp_run
Dhcp_init_parent:
;record the DHCP task id
MOV DPTR, #DHCP_TASKID
MOVX A, @DPTR
```

task_kill

Description: **int task_kill(**

int *id); /* *id* of the task to be killed */

The **task_kill** function kills a task. This function destroys a task and frees its state buffer. ID 0 means current task. It returns a success/failure code in the accumulator (ACC). Note: This function does not interact with the socket code. Call the socket function cleanup() to close all associated sockets.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*id*	ACC	Success (= 00h) or failure code

Example:

```
MOV DPTR, #DHCP_TASKID
MOVX A, @DPTR
ROMCALL task_kill
```

task_suspend

Description: **int task_suspend(**

int *id, /* *id* of task to be suspended */

int *eventmask; /* *eventmask* to be satisfied before awakening */

The **task_suspend** function suspends a task. It suspends a task until all events in *eventmask* have been generated. ID 0 means current task. This function returns a success/failure code in the accumulator. Before task suspension, the switch_out() hook is called. To wake up a task, use task_signal().

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*id*	ACC	Success (= 00h) or failure code
B	*eventmask*		

Example:

```
CLR A
MOV B, #TASK_DHCPSLEEP
ROMCALL task_suspend
```

task_sleep

Description: **int task_sleep(**

int *id, /* *id* of the task to be put to sleep */

int *eventmask; /* *eventmask* to be satisfied before awakening */

int *milliseconds);* /* number of *milliseconds* the task must sleep */

The **task_sleep** function puts a task to sleep. It suspends a task until at least *milliseconds* milliseconds have elapsed *or* the *eventmask* has occurred (use *eventmask* = 0 for regular sleep). ID 0 means current task. Before suspension, the **switch_out()** hook is called. The function returns a success/failure code in the accumulator (ACC).

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*id*		
B	*eventmask*	ACC	Success (= 00h) or failure code
R3:R0	*milliseconds*		

Example:
```
CLR A
MOV B, A  ;regular sleep for 10 seconds = 10000ms
MOV R3, #0
MOV R2, #1
MOV R1, #0
MOV R0, #0
ROMCALL task_sleep
```

task_signal

Description: **int task_signal(**

int *id,* /* *id* of task to send signal */

int *eventmask);* /* *eventmask* containing event(s) to signal */

The **task_signal** function signals a task. This function sends event(s) in *eventmask* to a task. If the task is waiting for no other events, it wakes up and is electable to be run by the task switcher. ID 0 means current task. This function returns a suc cess/failure code in the accumulator (ACC). It wakes up tasks that have been suspended by task_suspend().

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	*id*		
B	*eventmask*	ACC	Success (= 00h) or failure code

Example:
```
MOV A, …  ;regular sleep for 10 seconds = 10000ms
MOV B, #TASK_USER0 ;user defined event
ROMCALL task_signal
```

TASK SCHEDULER USER HOOKS

To allow the user to expand upon implemented task scheduler i,e., to save and restore additional properties of a task the task scheduler calls hook functions. The user must make sure to preserve all of the parameters passed across the hook function call.

task_create

The **task_create** function is called when the very first task block is created by **task_genesis**. The DS80C400 silicon software implementation of this function does nothing. The parameters automatically passed to **task_create** by **task_genesis** are defined as follows.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R1: R0	*savesize*		
DPTR0	Newly allocated TCB	User-defined	

task_duplicate

The **task_duplicate** function is called by the **task_fork** function. The DS80C400 silicon software implementation of this function does nothing. The parameters automatically passed to **task_duplicate** by **task_fork** are defined as follows.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R1: R0	*savesize*		
R7	*priority*	User-defined	
DPTR0	Newly allocated TCB		

task_destroy

The **task_destroy** function is called when a task is destroyed by **task_kill**. The DS80C400 silicon software implementation of this func-

tion does nothing. The parameters automatically passed to **task_destroy** by **task_kill** are defined as follows.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
DPTR0	TCB of task to be removed	User-defined	

rom_task_switch_in

The **rom_task_switch_in** function is called by the scheduler before a task is switched in. The DS80C400 silicon software implementation of this function restores the state buffer (stack and SFRs). This function is exported by the DS80C400 silicon software, and also serves as the default taskswitchin in the function redirect table.

Note: This function does not return. However, it returns to the caller of **rom_task_switch_out** with ACC = 0. The parameters passed to **rom_task_switch_in** are defined as follows.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
DPTR0	TCB of task to be switched in	User-defined	

rom_task_switch_out

The **rom_task_switch_out** function is called before a task is suspended (either voluntarily because it waits for an event or because its timeslice is over). The DS80C400 silicon software implementation of this function saves the task's state (stack and SFRs) to the state buffer. This function is exported by the DS80C400 silicon software, and also serves as the default **taskswitchout** in the function redirect table. **Note:** It returns ACC ! = 0. **rom_task_switch_in** returns to the caller of this function with ACC = 0. The parameters passed to **rom_task_switch_in** are defined as follows.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
DPTR0	TCB of task to be switched in	User-defined	

1-WIRE MASTER

OWM_First

The **OWM_First** function searches for the first 1-Wire device on the network and, if found, places the 8-byte unique ROM ID in the **OWM_ROMID** buffer. A pointer to the **OWM_ROMID** buffer is provided in the ROM export table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
—		ACC	device found (=01h) otherwise (=00h)

OWM_Next

The **OWM_Next** function searches for the next 1-Wire device on the network based upon the last known search ROM discrepancy and, if found, places the 8-byte unique ROM ID in the **OWM_ROMID** buffer. A pointer to the **OWM_ROMID** buffer is provided in the ROM export table.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
—		ACC	next device found (=01h) otherwise (=00h)

OWM_Reset

The **OWM_Reset** function performs a 1-Wire reset on the network and returns device presence and line-short conditions.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
—		ACC	line shorted (=00h) presence detected (=01h) no presence detected (=03h)

OWM_Byte

The **OWM_Byte** function performs eight 1-Wire bit timeslots on the network and returns the response.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	byte to transmit	ACC	byte received

OWM_ROMID

OWM_ROMID is a memory buffer where 8-byte address (unique ROM ID) is stored after successful **OWM_First** or **OWM_Next** function call. A pointer to **OWM_ROMID** buffer is provided in the ROM export table. The 8-byte returned address is stored as follows:

@OWM_ROMID+0:	Family Code
@OWM_ROMID+1:	LSByte of 6-byte serial number
@OWM_ROMID+2:	LSByte+1 of 6-byte serial number
@OWM_ROMID+3:	LSByte+2 of 6-byte serial number
@OWM_ROMID+4:	LSByte+3 of 6-byte serial number
@OWM_ROMID+5:	LSByte+4 of 6-byte serial number
@OWM_ROMID+6:	MSByte of 6-byte serial number
@OWM_ROMID+7:	CRC

INITIALIZATION FUNCTIONS

Before using any DS80C400 silicon software function, the system must have a valid interrupt vector table and function redirect table, and all memory must be cleared (directs, RAM). Additionally, other initialization functions for the memory manager, scheduler, and networking hardware/software drivers must be executed. The **rom_init** call takes care of required initialization. It is expected that using the **rom_init** call is the preferred method for initializing the system, therefore, only high-level descriptions are given for the functions called by **rom_init**.

rom_init

The rom_init function performs the initialization that provides functionality to the exported DS80C400 silicon software functions. Supplying a lower address bound (R2:R0) of 000000h results in a default heap-memory allocation that afterward can be assessed by examining the BOOT_MEMBEGIN and BOOT_MEMEND parameters. **rom_init** is automatically executed with default heap-memory allocation when NetBoot is selected. Otherwise, the user code should supply the lower and upper address bounds for heap-memory allocation when calling **rom_init**. The basic **rom_init** execution flow is given following the input/output parameter definition.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
R2:R0	Lower address bound for heap memory allocation	ACC	return value (0 = success)
R5:R3	Upper address bound for heap memory allocation		

Init Function	Description*
1. **rom_copyivt**	Copies a default interrupt vector table from ROM to external memory starting at 000000h. The default interrupt vector table executes a RETI for all interrupts except for the following: Timer 0 (000Bh)—WOS_tick routine Power-Fail (0033h)—Loops until the PFI bit can be cleared, signaling that power has returned to a good state. Once PFI can be cleared, a jump is made to the start of ROM as if power had been lost. Ethernet Act (0073h)—ETH_ProcessInterrupt routine
2. **rom_redirect_init**	Copies the redirect call table from ROM to external memory 0100h–017Fh.
3. ----	SETB EPFI instruction enables power-fail interrupt
4. ----	Prints part of the opening message "DS80C400 Silicon Software "
5. ----	Clears internal direct memory (scratchpad) 00h–FFh
6. ----	Clears external system xdata memory between 0180h–(start address of BLOB - 1) that is used for network and task manager data structures.
7. **mm_init**	Initialize heap memory according to upper/lower bound addresses passed to rom_init. If the default heap-memory allocation is used, the address range is ~(start address of BLOB +1000h) – FFFFh.
8. **km_init**	Initialize kernel memory

9. ----	Prints more of the opening message "- Copyright (C) "
10. ----	Internal functions to manipulate the 1-Wire bus, make timing measurements, and calculate an appropriate setting for the 1-Wire clock divisor register.
11. **ow_init**	Initialize 1-Wire master. As an input parameter, the **ow_init** function requires that ACC contain the desired value to program into the 1-Wire clock divisor register. If ow_init is called as part of the rom_init function, this input parameter is passed automatically from the previous internal functions.
12. **network_init**	Initialize network layer
13. ----	Prints more of the opening message " S/N: "
14. **eth_init**	Initialize driver
15. ----	Prints the 8-byte serial number "NNNNNNNNNNNNNNN"
16. **init_sockets**	Initialize socket layer
17. ----	Prints more of the opening message " MAC ID: "
18. **tick_init**	Initialize Timer0 periodic interrupt
19. ----	Prints the 6-byte MAC address "MMMMMMMMMMM"
20. **task_genesis**	Initialize scheduler and create an idle thread
21. ----	SETB EA enables interrupts

*The function call **rom_init** takes care of all required initialization. This includes copying the interrupt vector table **(rom_copyivt**) and function redirect table **(rom_redirect_init)**, as well as calling all of the above initialization routines. The rom_init function also prints a copyright message. No extended documentation of these functions is currently available.

ASYNCHRONOUS TCP/IP MAINTENANCE FUNCTIONS

The default timer-interrupt handler (**WOS_tick**) periodically calls the default task scheduler (**WOS_IOPoll**). This **WOS_IOPoll** routine automatically calls certain asynchronous TCP/IP stack-maintenance functions and then calls the **User_IOPoll** redirect function. The **User_IOPoll** hook provides a simple means for the user to extend the task scheduler if he/she desires. Since **WOS_IOPoll** is already designed to handle the necessary TCP/IP maintenance function and can easily be supplemented with user-specific task-scheduler code, it is the preferred method for ensuring proper and timely execution of the specified TCP/IP maintenance tasks. Therefore, only high-level descriptions are given for the functions called by **WOS_IOPoll**.

MAINTENANCE FUNCTION	DESCRIPTION
1) **IP_ProcessReceiveQueues**	Drive all the protocol receive queues. Checks the ARP, TCP, UDP, and ICMP receive queues for messages that need to be processed. Processing the receive packet may involve passing the message to a higher level protocol handler or building a response queuing the response for transmission.
2) **IP_ProcessOutput**	Drive all the protocol transmit queues. Checks all active connections to deter mine whether output is needed in the form of data, acknowledge, or both.
3) **IP6_ProcessReceiveQueues**	Drive all the IPv6 protocol receive queues
4) **IP6_ProcessOutput**	Drive all the IPv6 protocol transmit queues
5) **TCP_RetryTop**	Run TCP retry process. Searches connections for unacknowledged segments. Any found with expired retry timers are retransmitted.
6) **ETH_ProcessOutput**	Age all entries in the ARP cache
7) **IGMP_GroupMaintenance**	Let IGMP output run. Checks all groups to see if IGMP report messages need to be generated.

OTHER FUNCTIONS/POINTERS

eth_processinterrupt—default Ethernet interrupt handler. The Ethernet interrupt handler unloads packets from the Ethernet hardware buffer and enqueues them into the appropriate TCP/IP protocol queues.

arp_generaterequest—generates ARP request

MAC_ID—pointer to MAC ID, stored Big Endian

entercritsection—enter critical section. This function increments the variable wos_crit_count located in direct data memory at address 6Bh.

leavecritsection—leave critical section. This function decrements the variable wos_crit_count located in direct data memory at address 6Bh.

WOS_tick—default timer interrupt handler. A flow diagram for this interrupt handler is provided in Figure 23-1.

BLOB—pointer to the start address of xdata memory ignored by NetBoot. The BLOB memory space, by default, occupies at least 4kB of contiguous memory, ending at address (BOOT_MEMBEGIN - 1). Since this memory is not touched by the DS80C400 Silicon Software, it is well suited for user xdata.

RAMTOP—reserved for use by Maxim/Dallas Semiconductor

BOOT_MEMBEGIN—pointer to starting address of network stack buffers and heap memory. The user may define this starting address as an input parameter to the rom_init function call or may allow a default assignment to be made by rom_init. The default starting address for heap memory is the second 256-byte page boundary (i.e., 00xx00h) following address = BLOB + 1000h.

BOOT_MEMEND—pointer to the ending address of network stack buffers and heap memory. The user may define this ending address as an input parameter to the rom_init function call or may allow a default assignment to be made by rom_init. The default ending address for heap memory is 00FFFFh.

autobaud—The same autobaud procedure is executed as can be executed during the boot process. Please reference the earlier text for a description of the autobaud function.

ROM REDIRECT FUNCTION TABLE

Since the socket interface is used by both NetBoot (from DS80C400 silicon software) and the user code (possibly running under a run-time environment or operating system), the code must be flexible enough to support all types of memory managers, as well as task and thread schedulers. Therefore, the DS80C400 silicon software socket interface code does not call these functions directly, but it makes use of a function redirect table . During a NetBoot, the DS80C400 silicon software provides its own minimal implementations of these functions. However, to use the socket layer from an application, users can substitute their own implementations for the following functions:

Table 23-1. ROM REDIRECT FUNCTIONS

	FUNCTION	TABLE OFFSET	DESCRIPTION
	bootstate	00h	Reserved for use by Maxim/Dallas Semiconductor
MEMORY MANAGER	kernelmalloc	03h	see **rom_kernelmalloc**
	kernelfree	06h	see **rom_kernelfree**
	malloc	09h	see **rom_malloc**
	free	0Ch	see **rom_free**
	mallocdirty	0Fh	see **rom_mallocdirty**
	deref	12h	see **rom_deref**
	underef	54h	Opposite of **deref** (12h)
TASK MANAGER	getfreeram	15h	see **rom_getfreeram**
	gettimemillis	18h	Returns uptime since the system was initialized in milliseconds.
	getthreadID	1Bh	Returns thread ID.
	threadresume	1Eh	Resumes thread
	threadIOsleep	21h	Sleeps, waiting for I/O
	threadIOsleepNC	24h	Sleeps, waiting for I/O (run from critical section)
	threadsave	27h	Saves thread
	threadrestore	2Ah	Restores thread
	sleep	2Dh	Sleeps for a number of milliseconds
	gettaskID	30h	see **task_getcurrent**
	infosendchar	33h	Prints debug character to debug port
	IPchecksum	36h	Computes IP checksum
	reserved	39h	Reserved for use by Maxim/Dallas Semiconductor
TASK MANAGER (HOOKS)	DHCPnotify	3Ch	see **rom_dhcp_notify**
	taskcreate	3Fh	see **task_create**
	taskduplicate	42h	see **task_duplicate**
	taskdestroy	45h	see **task_destroy**
	taskswitchin	48h	see **rom_task_switch_in**
	taskswitchout	4Bh	see **rom_task_switch_out**
	getMACID	4Eh	Reads MAC ID from DS2502 1-Wire device and IP/gateway/TFTP server from other 1-Wire device
	reserved	51h	Reserved for use by Maxim/Dallas Semiconductor
	USERIOpoll	57h	Called by scheduler
	errornotification	5Ah	Called when out of memory or other error detected

Functions should be replaced in groups, e.g., if the user provides his/her own memory manager, all the memory manager functions should be replaced. Note: All DS80C400 silicon software functions (including the exported functions) make heavy use of this table. Therefore, it must always exist, either in its default state or modified by the user. The function redirect table that is contained in the DS80C400 silicon software is copied to memory using the ROM_redirect_init function. NetBoot calls this function. If the user does not use NetBoot, ROM_Redirect_Init must be called. ROM_Redirect_Init restores the function redirect table without altering any other state.

ROM REDIRECT FUNCTIONS

The usage of those ROM redirect functions not previously described as a directly exported ROM function (i.e., member of the ROM export table) are covered here.

gettimemillis

The gettimemillis function gets the current time in milliseconds. The DS80C400 silicon software does not support a real-time clock; therefore, the DS80C400 silicon software version of this function returns the number of milliseconds since the system was initialized. A user replacement of this function should return the absolute time.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		R4:R0	Time in milliseconds

getthreadID

The **getthreadID** function gets the current thread ID. The DS80C400 silicon software does not support threads; therefore, the DS80C400 silicon software version of this function always returns ACC = 01h.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Always returns 01h

threadresume

The **threadresume** function resumes a suspended thread. The DS80C400 silicon software does not support threads; therefore, the DS80C400 silicon software version of this function resumes the task.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	Thread ID	ACC	Success (= 00h) or failure code
R0	Task ID		

threadIOsleep

threadIOsleepNC

The **threadIOsleep** and **threadIOsleepNC** functions put a thread to sleep, waiting for I/O. The **NC** version of this function does not enter a critical section and is only called from critical sections. The DS80C400 silicon software does not support threads; therefore, the DS80C400 silicon software version of this function works on the task.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	Set <> 00h for infinite timeout	ACC	Success (= 00h) or failure code
R3:R0	Timeout value (if ACC = 00h)		

threadsave

threadrestore

The **threadsave** and **threadrestore** functions save/restore (the state of) a thread. The DS80C400 silicon software does not support threads and, therefore, the DS80C400 silicon software versions of these functions do nothing.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		-	

sleep

The **sleep** function sleeps, suspending a task for at least the requested amount of time.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	Task ID	ACC	Success (= 00h) or failure code
R3:R0	Sleep time in milliseconds		

infosendchar

The **infosendchar** function sends a character to the serial port 0. The DS80C400 silicon software version of this function accesses the serial loader pin (P1.7) and does nothing if this pin is in the logic low state. The DS80C400 silicon software does not use interrupt-driven I/O to the serial port.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	Character to send	-	

IPchecksum

The **IPchecksum** function calculates the IP checksum for an IP packet. The DS80C400 silicon software version of this function does not use the checksum accelerator feature of the DS80C400.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
DPTR0 R5:R4	Pointer to data buffer Size of data buffer	R1:R0	Checksum

getMACID

The **getMACID** function reads the MAC ID and stores it in the MAC_ID variable (a MAC_ID pointer is given in the ROM export table). The DS80C400 silicon software version of this function accesses the 1-Wire port and searches for a DS2502U-E48 1-Wire chip containing the MAC ID.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
-		ACC	Success (= 00h) or failure code

underef

Resolves a physical address into a memory handle. This function undereferences an absolute address and returns a memory handle.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
DPTR0	pointer to memory block	ACC R3:R2	(destroyed) *blockhandle*

userIOpoll

Allows extension of the WOS_IOPoll mechanism. This function is called on every WOS_IOPoll, i.e., driven by the timer when no interrupts are active. A user could add housekeeping functions, for example. The default implementation of this function does nothing.

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
—		—	

errornotification

Provides a means for notification when an error occurs. This function is called when the system error occurs such as not having enough memory. The list of error causes/reasons currently defined within the ROM is given below. The default implementation of this function does nothing, however, a user implementation might print an error message or reset the system.

Error Cause	Value	Description
ERROR_KMEM	0	Kernel memory exhausted
ERROR_MRM	1	Heap memory exhausted
ERROR_TCP_MEM	2	TCP detected low memory
ERROR_TCP_RESEND	3	TCP packet has to be resent
ERROR_KFREE_FAIL	4	Attempt to free a kernel memory block failed
ERROR_FREE_NULL	5	Attempt to free a NULL pointer
ERROR_FREE_DEREF	6	Could not dereference a memory handle on free

INPUT	DESCRIPTION	OUTPUT	DESCRIPTION
ACC	Value for a given error cause	-	

TIMESLICE AND TASK SCHEDULER TIMING

The task scheduler is primarily driven by the timer 0. The exported ROM function **tick_init** handles the initial timer 0 configuration. The **tick_init** routine configures timer 0 as a 16-bit timer running from an (oscillator frequency / 12) input clock, with an initial start value of TH0:TL0 = FA00h. Timer 0 is then enabled as a high-priority interrupt source and started (TR0 = 1). Once running, the timer 0 interrupt is triggered at a periodic interval so that it can be determined whether the scheduler needs to be run and whether it should be run given the current CPU activity. This periodic interval is equal to (65,536d – FA00h) × 1 / (crystal frequency) × 12 and is defined as the basic timeslice. The exported ROM function **WOS_tick** is the default timer 0 interrupt service routine that is executed each timeslice. The **WOS_tick** routine updates its millisecond counter and signals that the task scheduler should be run, once the fourth timeslice has elapsed. At this point, the task scheduler is run unless other interrupts are in progress, in which case, the task scheduler is deferred. If the scheduler is deferred, its ability to run is reassessed on each of the following (timer 0 interrupt) timeslices or whenever a task is suspended or put to sleep. Note that the **tick_init** function is automatically called by the **rom_init** function. Figure 23-1 illustrates the general flow that occurs on each (timer 0 interrupt) timeslice.

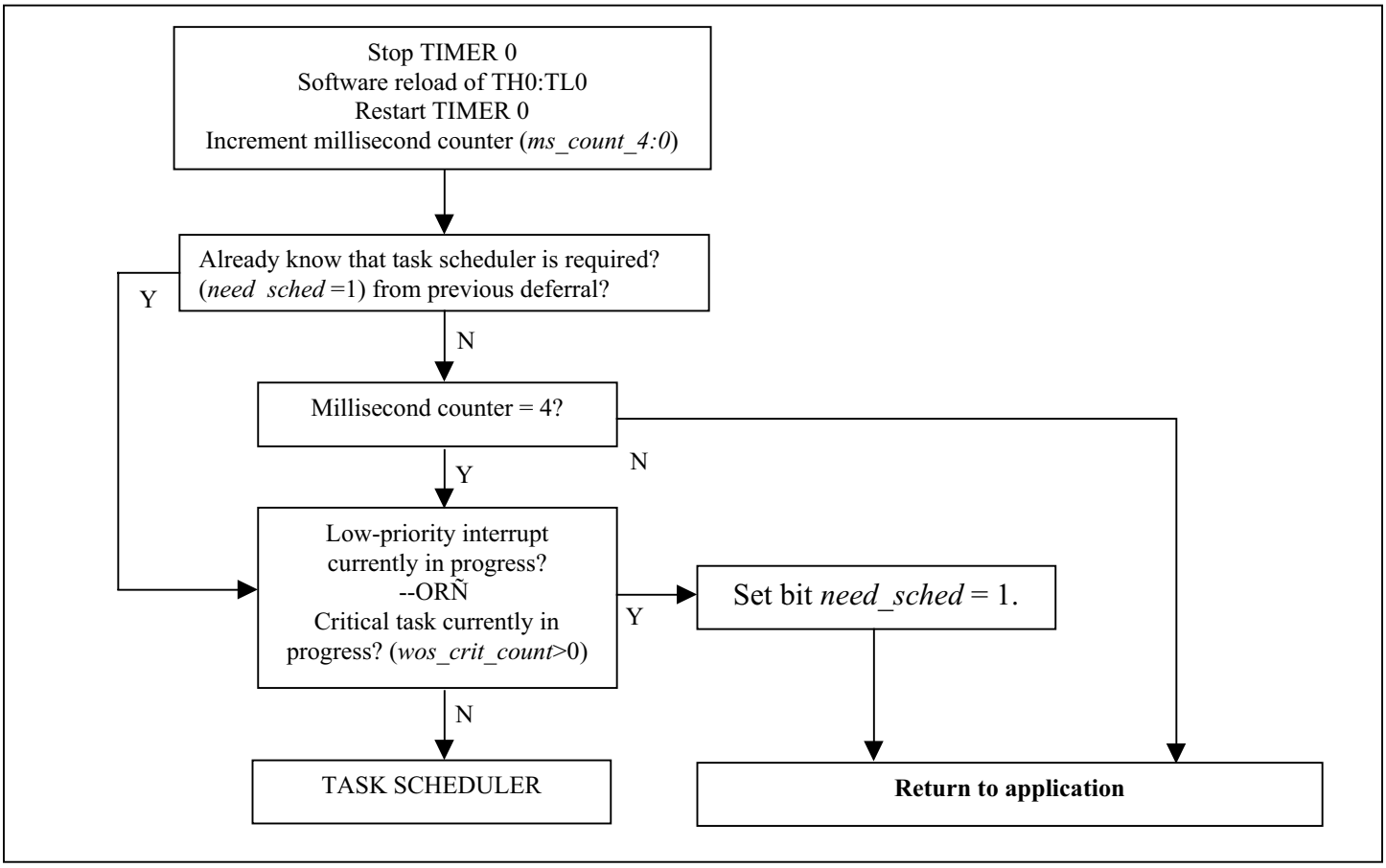


Figure 23-1. Timer 0 Interrupt Routine (WOS_tick) Flow