Analog Devices Inc./Maxim Integrated - DS80C390-FNR+ Datasheet

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	8051
Core Size	8-Bit
Speed	40MHz
Connectivity	CANbus, EBI/EMI, SIO, UART/USART
Peripherals	Power-Fail Reset, WDT
Number of I/O	32
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	3.85V ~ 5.5V
Data Converters	-
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	64-LQFP
Supplier Device Package	64-LQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/analog-devices/ds80c390-fnr

AC ELECTRICAL CHARACTERISTICS—(MULTIPLEXED ADDRESS/DATA BUS) (Note 10, Note 11)

PARAMETER	SYMBOL	CONDITIONS	40MHz		VARIABLE CLOCK		UNITS
			MIN	MAX	MIN	MAX	
Oscillator Frequency	$1 / t_{CLCL}$	External oscillator	0	40	0	40	MHz
		External crystal	1	40	1	40	
ALE Pulse Width	t_{LHLL}				$0.375\ t_{MCS} - 5$		ns
Port 0 Instruction Address or $\overline{CE0–4}$ Valid to ALE Low	t_{AVLL}				$0.125\ t_{MCS} - 5$		ns
Address Hold After ALE Low	t_{LLAX1}				$0.125\ t_{MCS} - 5$		ns
ALE Low to Valid Instruction In	t_{LLIV}					$0.625\ t_{MCS} - 20$	ns
ALE Low to $\overline{PSEN}$ Low	t_{LLPL}				$0.125\ t_{MCS} - 5$		ns
$\overline{PSEN}$ Pulse Width	t_{PLPH}				$0.5\ t_{MCS} - 8$		ns
$\overline{PSEN}$ Low to Valid Instruction In	t_{PLIV}					$0.5\ t_{MCS} - 20$	ns
Input Instruction Hold After $\overline{PSEN}$	t_{PXIX}		0		0		ns
Input Instruction Float After $\overline{PSEN}$	t_{PXIZ}					$0.25\ t_{MCS} - 5$	ns
Port 0 Address to Valid Instruction In	t_{AVIV1}					$0.75\ t_{MCS} - 22$	ns
Port 2, 4 Address to Valid Instruction In	t_{AVIV2}					$0.875\ t_{MCS} - 30$	ns
$\overline{PSEN}$ Low to Address Float	t_{PLAZ}			0		0	ns

Note 11: All parameters apply to both commercial and industrial temperature operation unless otherwise noted. The value t_{MCS} is a function of the machine cycle clock in terms of the processor's input clock frequency. These relationships are described in the *Stretch Value Timing* table. All signals characterized with load capacitance of 80pF except Port 0, ALE, $\overline{PSEN}$, $\overline{RD}$, and $\overline{WR}$ with 100pF. Interfacing to memory devices with float times (turn off times) over 25ns can cause bus contention. This does not damage the parts, but causes an increase in operating current. Specifications assume a 50% duty cycle for the oscillator. Port 2 and ALE timing changes in relation to duty cycle variation. Some AC timing characteristic drawings contain references to the CLK signal. This waveform is provided to assist in determining the relative occurrence of events and cannot be used to determine the timing of signals relative to the external clock. AC timing is characterized and guaranteed by design but is not production tested.

Figure 3. Multiplexed 9-Cycle Address/Data $\overline{\text{PCE0-3}}$ MOVX Read/Write Operation

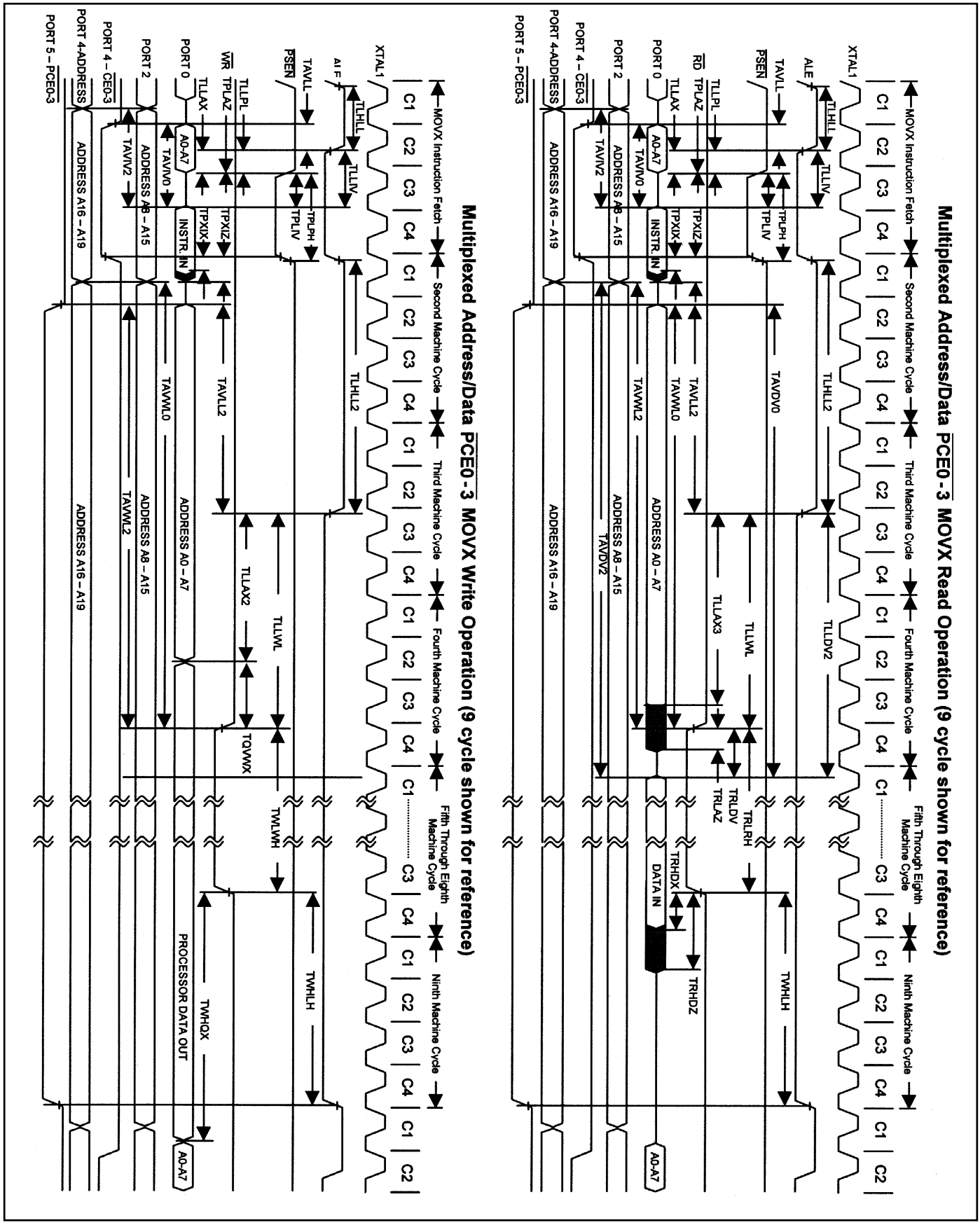


Figure 6. Multiplexed 2-Cycle Data Memory $\overline{CE0\text{-}3}$ Write

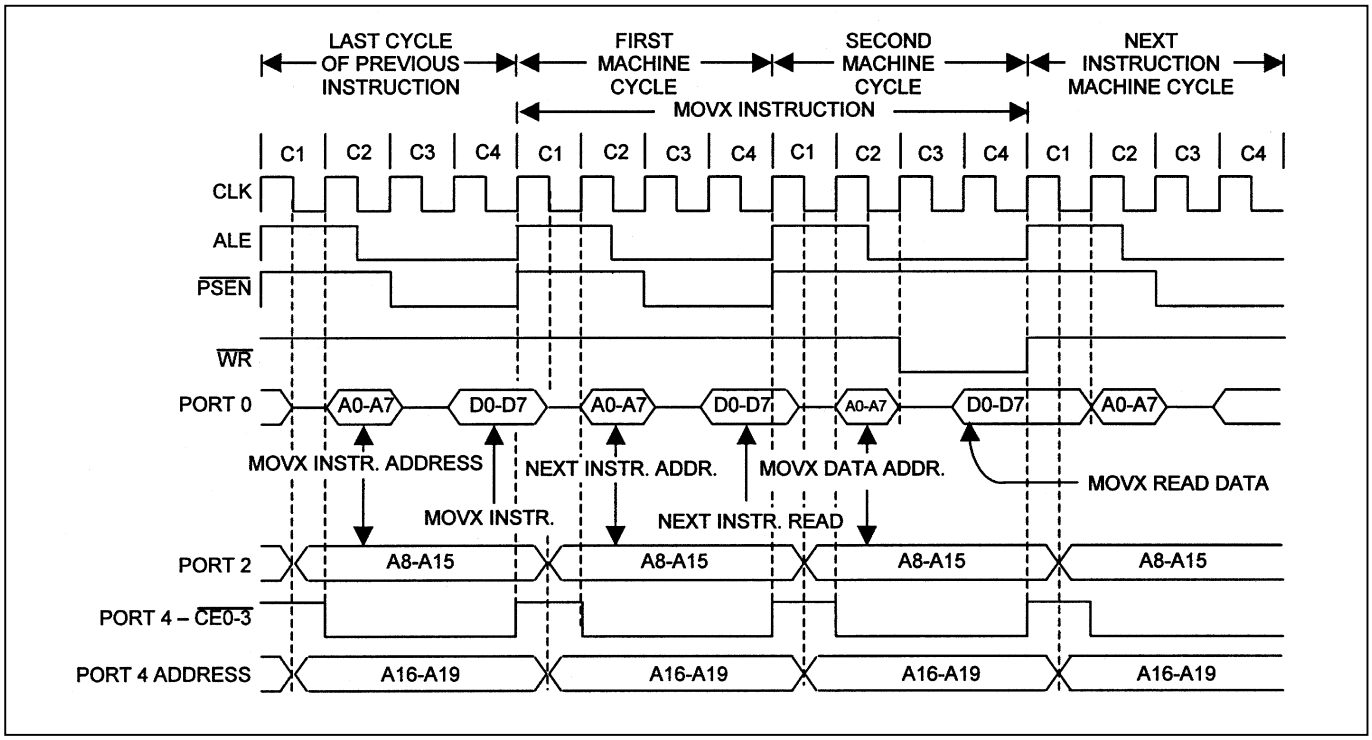


Figure 7. Multiplexed 3-Cycle Data Memory $\overline{PCE0\text{-}3}$ Read or Write

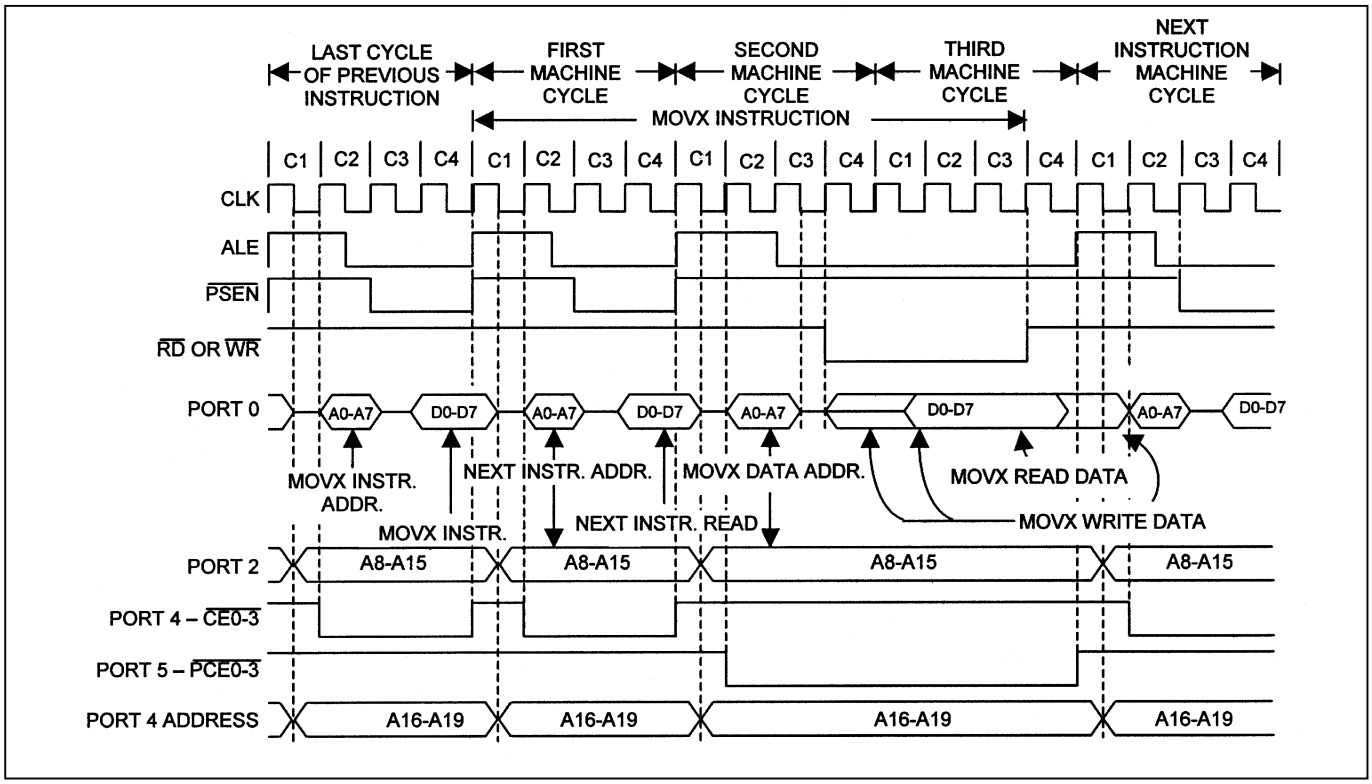


Figure 10. Multiplexed 9-Cycle Data Memory $\overline{\text{PEC0-3}}$ Read or Write

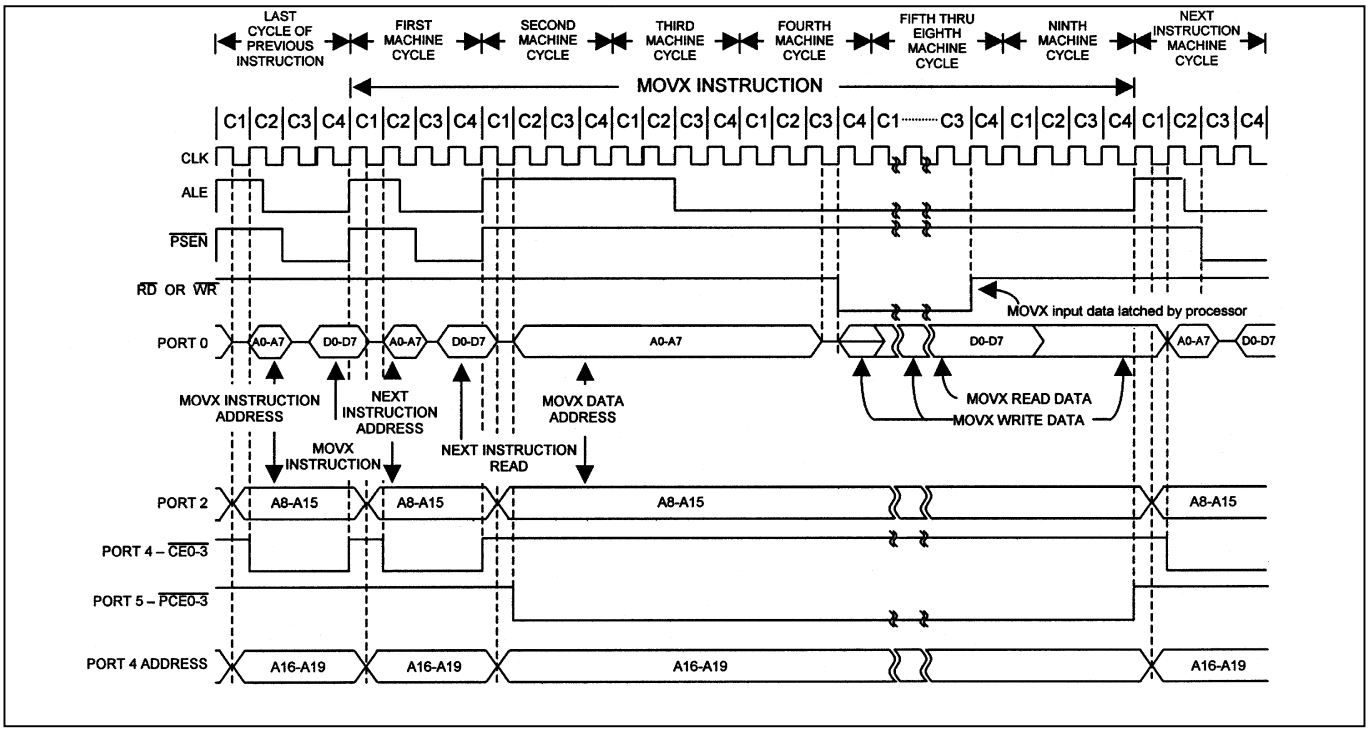
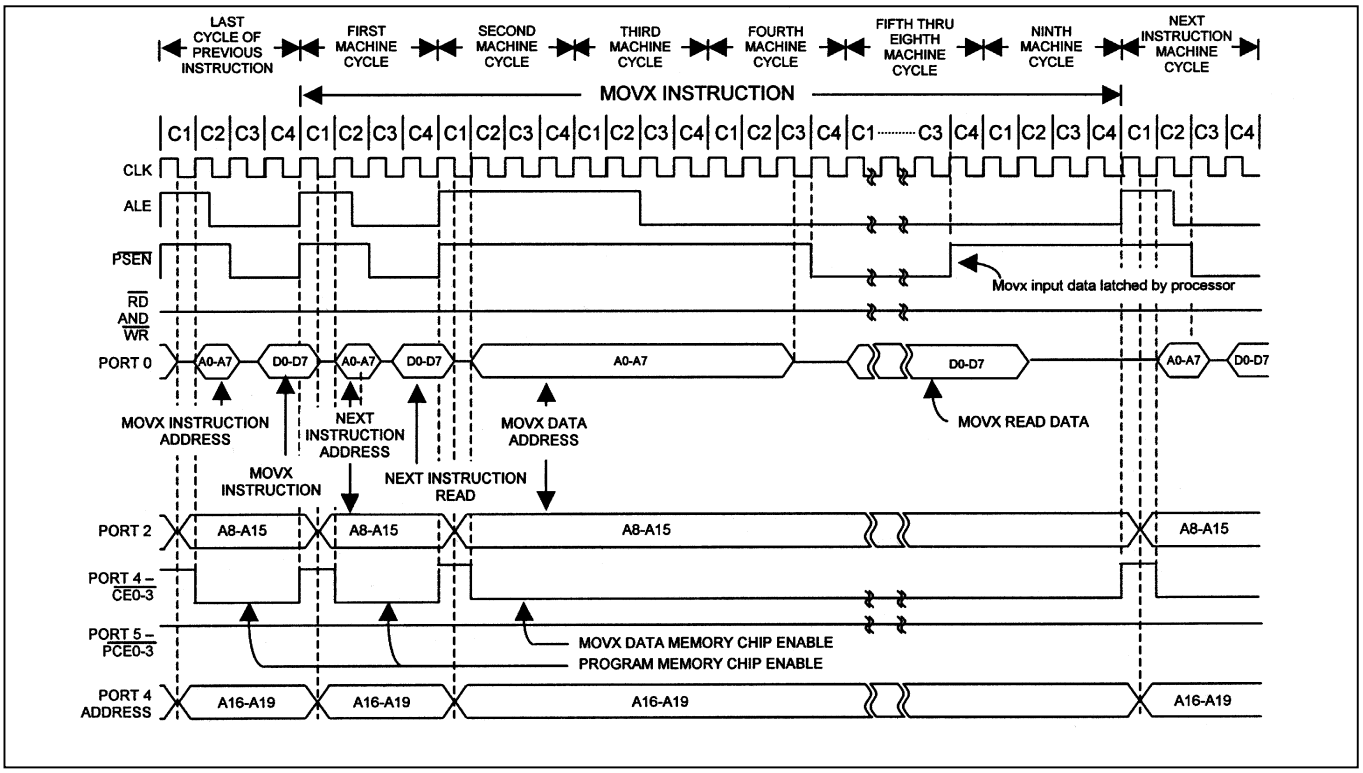


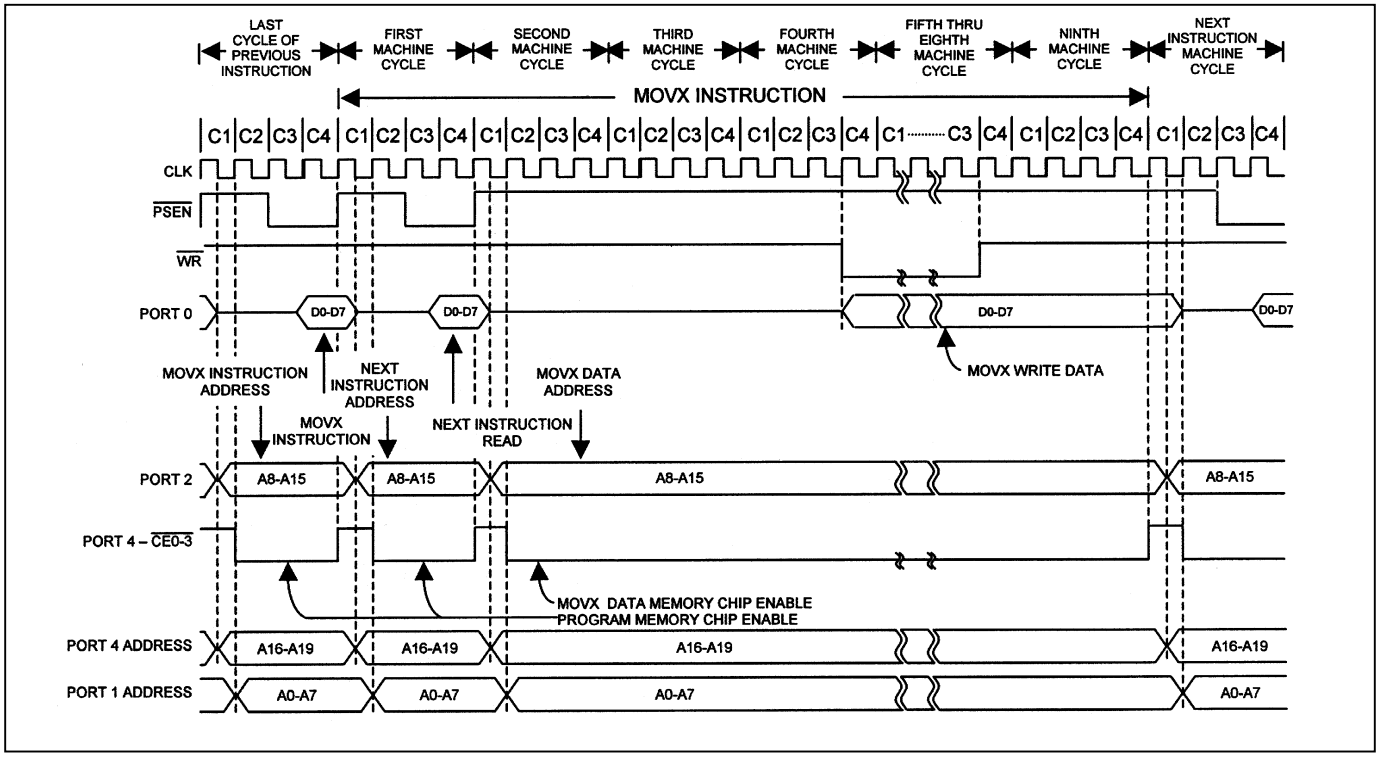
Figure 11. Multiplexed 9-Cycle Data Memory $\overline{\text{CE0-3}}$ Read



MOVX CHARACTERISTICS (NONMULTIPLEXED ADDRESS/DATA BUS)

PARAMETER	SYMBOL	MIN	MAX	UNITS	STRETCH VALUES C_{ST} (MD2:0)
$\overline{RD}$ Pulse Width	t_{RLRH}	0.5 t_{MCS} - 6		ns	C_{ST} = 0
		C_{ST} x t_{MCS} - 6			1 ≤ C_{ST} ≤ 7
$\overline{WR}$ Pulse Width	t_{WLWH}	0.5 t_{MCS} - 6		ns	C_{ST} = 0
		C_{ST} x t_{MCS} - 6			1 ≤ C_{ST} ≤ 7
$\overline{RD}$ Low to Valid Data In	t_{RLDV}		0.5 t_{MCS} - 20	ns	C_{ST} = 0
			C_{ST} x t_{MCS} - 25		1 ≤ C_{ST} ≤ 7
Data Hold After Read	t_{RHDX}	0		ns	
Data Float After Read	t_{RHDZ}		0.125 t_{MCS} - 5	ns	C_{ST} = 0
			0.375t_{MCS} - 5		1 ≤ C_{ST} ≤ 3
			1.375 t_{MCS} - 5		4 ≤ C_{ST} ≤ 7
Port 1 Address, Port 4 CE, Port 5 PCE to Valid Data In	t_{AVDV1}		0.75 t_{MCS} - 26	ns	C_{ST} = 0
			($4C_{ST}$ + 0.5) x t_{MCS} - 30		1 ≤ C_{ST} ≤ 3
			($4C_{ST}$ + 2.5) x t_{MCS} - 30		4 ≤ C_{ST} ≤ 7
Port 2, 4 Address to Valid Data In	t_{AVDV2}		0.75 t_{MCS} - 30	ns	C_{ST} = 0
			($4C_{ST}$ + 0.625) x t_{MCS} - 30		1 ≤ C_{ST} ≤ 3
			($4C_{ST}$ + 2.625) x t_{MCS} - 30		4 ≤ C_{ST} ≤ 7
Port 0 Address, Port 4 CE, Port 5 PCE to $\overline{RD}$ or $\overline{WR}$ Low	t_{AVWL1}	0.25 t_{MCS} - 11		ns	C_{ST} = 0
		0.5 t_{MCS} - 11			1 ≤ C_{ST} ≤ 3
		2.5 t_{MCS} - 11			4 ≤ C_{ST} ≤ 7
Port 2, 4 Address to $\overline{RD}$ or $\overline{WR}$ Low	t_{AVWL2}	0.375 t_{MCS} - 11		ns	C_{ST} = 0
		0.625t_{MCS} - 11			1 ≤ C_{ST} ≤ 3
		2.625 t_{MCS} - 11			4 ≤ C_{ST} ≤ 7
Data Valid to $\overline{WR}$ Transition	t_{QVWX}	-8		ns	
Data Hold After $\overline{WR}$ High	t_{WHQX}	0.25 t_{MCS} - 8		ns	C_{ST} = 0
		0.5t_{MCS} - 10			1 ≤ C_{ST} ≤ 3
		1.5 t_{MCS} - 10			4 ≤ C_{ST} ≤ 7
$\overline{RD}$ or $\overline{WR}$ High to ALE, Port 4 CE or Port 5 PCE High	t_{WHLH}	-5	10	ns	C_{ST} = 0
		0.25 t_{MCS} - 7	0.25 t_{MCS} + 10		1 ≤ C_{ST} ≤ 3
		1.25 t_{MCS} - 7	1.25 t_{MCS} + 10		4 ≤ C_{ST} ≤ 7

Figure 24. Nonmultiplexed 9-Cycle Data Memory $\overline{CE0\text{-}3}$ Write



t_{MCS} TIME PERIODS

SYSTEM CLOCK SELECTION			t_{MCS}
4X/$\overline{2X}$	CD1	CD0	
1	0	0	1 t_{CLCL}
0	0	0	2 t_{CLCL}
X	1	0	4 t_{CLCL}
X	1	1	1024 t_{CLCL}

EXTERNAL CLOCK CHARACTERISTICS

PARAMETER	SYMBOL	MIN	MAX	UNITS
Clock High Time	t_{CHCX}	8		ns
Clock Low Time	t_{CLCX}	8		ns
Clock Rise Time	t_{CLCH}		4	ns
Clock Fall Time	t_{CHCL}		4	ns

Figure 25. External Clock Drive

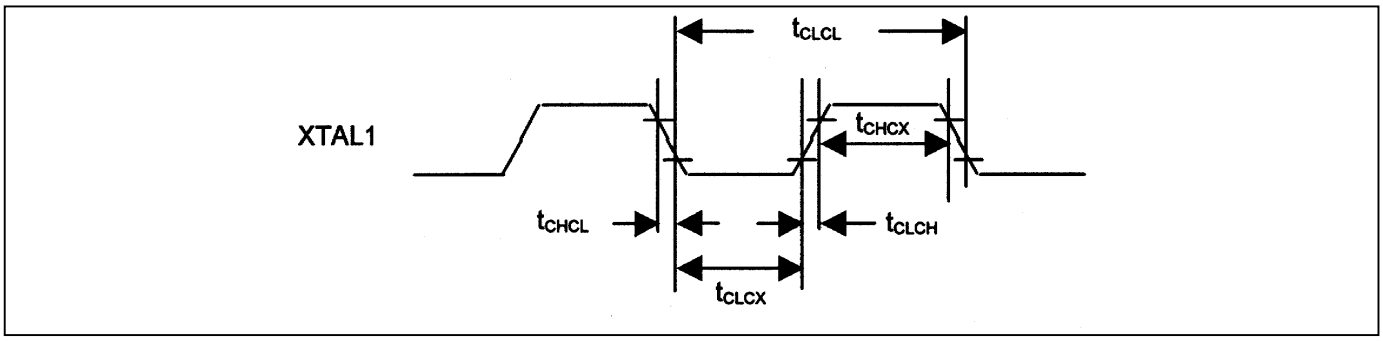
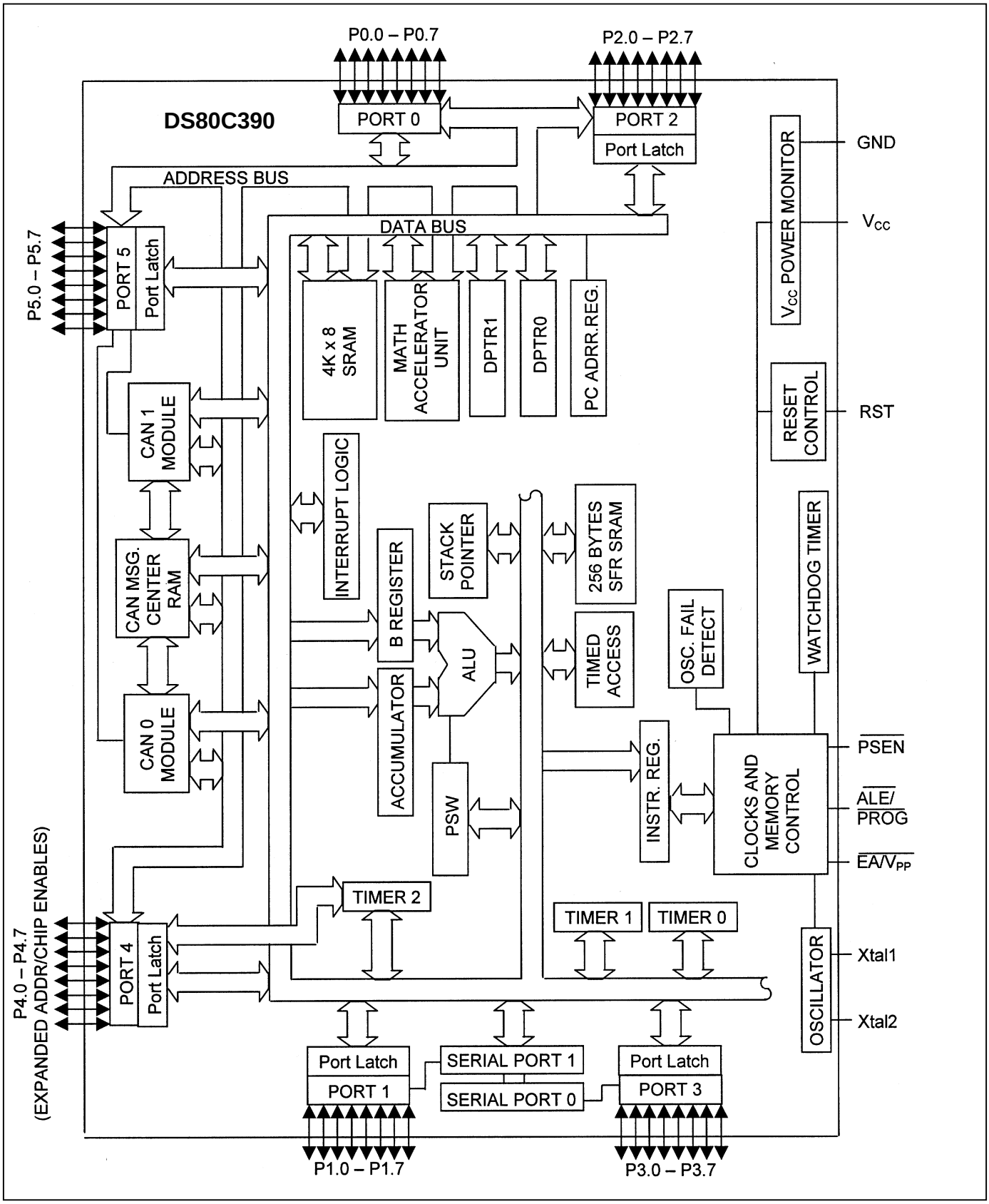


Figure 28. Block Diagram



ON-CHIP ARITHMETIC ACCELERATOR

An on-chip math accelerator allows the microcontroller to perform 32-bit and 16-bit multiplication, division, shifting, and normalization using dedicated hardware. 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 MC) are accessed, eliminating the need for a special step to choose the operation. The normalize function facilitates the conversion of 4-byte unsigned binary integers into floating point format. Table 2 shows the operations supported by the math accelerator and their time of execution.

Table 2. Arithmetic Accelerator Execution Times

OPERATION	RESULT	EXECUTION TIME (t_{CLCL})
32-Bit/16-Bit Divide	32-Bit Quotient, 16-Bit Remainder	36
16-Bit/16-Bit Divide	16-Bit Quotient, 16-Bit Remainder	24
16-Bit/16-Bit Multiply	32-Bit Product	24
32-Bit Shift Left/Right	32-Bit Result	36
32-Bit Normalize	32-Bit Mantissa, 5-Bit Exponent	36

Table 3 demonstrates the procedure to perform mathematical operations using the hardware math 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 access 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. Consult the description of the MCNT0 SFR for details of how the shift and normalize functions operate.

Software must ensure that the input value for the normalize operation is not zero or the function will not complete. Compilers such as the one from Keil Software have updated their libraries and compensate for this condition.

Table 3. Arithmetic Accelerator Sequencing

DIVIDE (32/16 OR 16/16)	MULTIPLY (16 X 16)
Load MA with dividend LSB. *Load MA with dividend LSB + 1.** *Load MA with dividend LSB + 2.** Load MA with dividend MSB. Load MB with divisor LSB. Load MB with divisor MSB. Poll the MST bit until cleared. (9 machine cycles). Read MA to retrieve the quotient MSB. *Read MA to retrieve the quotient LSB + 2.*** *Read MA to retrieve the quotient LSB + 1.*** Read MA to retrieve the quotient LSB. Read MB to retrieve the remainder MSB. Read MB to retrieve the remainder LSB.	Load MB with multiplier LSB. Load MB with multiplier MSB. Load MA with multiplicand LSB. Load MA with multiplicand MSB. Poll the MST bit until cleared. (6 machine cycles). Read MA for product MSB. Read MA for product LSB + 2. Read MA for product LSB + 1. Read MA for product LSB.
SHIFT RIGHT/LEFT	**NORMALIZE**
Load MA with data LSB. Load MA with data LSB + 1. Load MA with data LSB + 2. Load MA with data MSB. Configure MCNT0 register as required Poll the MST bit until cleared. (9 machine cycles) Read MA for result MSB. Read MA for result LSB + 2. Read MA for result LSB + 1. Read MA for result LSB.	Load MA with data LSB. Load MA with data LSB + 1. Load MA with data LSB + 2. Load MA with data MSB.*** Load MCNT0 with 00h. Poll the MST bit until cleared. (9 machine cycles) Read MA for mantissa MSB. Read MA for mantissa LSB + 2. Read MA for mantissa LSB + 1. Read MA for mantissa LSB. Read MCNT0.4–MCNT0.0 for exponent.

**Not performed for 16-bit numerator.*

***Not performed for 16/16 divide.*

****Value to be normalized must be nonzero.*

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;D2h) 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.

MEMORY ADDRESSING

The DS80C390 incorporates three internal memory areas:

- 256 bytes of scratchpad (or direct) RAM
- 4kB of SRAM configurable as various combinations of MOVX data memory, stack memory, and MOVC program memory
- 512 bytes of RAM reserved for the CAN message centers.

Up to 4MB of external memory is addressed via a multiplexed or demultiplexed 20-bit address bus/8-bit data bus and four chip-enable (active during program memory access) or four peripheral-enable (active during data memory access) signals. Three different addressing modes are supported, as selected by the AM1, AM0 bits in the ACON SFR.

16-Bit Address Mode

Memory is accessed by 16-bit address mode similarly to the traditional 8051. It is op-code compatible with the 8051 microprocessor and identical to the byte and cycle count of the Dallas Semiconductor High-Speed Microcontroller family. A device operating in this mode can access up to 64kB of program and data memory. The device defaults to this mode following any reset.

22-Bit Paged-Address Mode

The 22-bit paged-address mode retains binary-code compatibility with the 8051 instruction set, but adds one machine cycle to the ACALL, LCALL, RET, and RETI instructions with respect to Dallas Semiconductor's High-Speed Microcontroller family timing. This is transparent to standard 8051 compilers. Interrupt latency is also increased by one machine cycle. In this mode, interrupt vectors are fetched from 0000xxh.

22-Bit Contiguous Address Mode

The 22-bit contiguous addressing mode uses a full 22-bit program counter, and all modified branching instructions automatically save and restore the entire program counter. The 22-bit branching instructions such as ACALL, AJMP, LCALL, LJMP, MOV DPTR, RET, and RETI instructions require an assembler, compiler, and linker that specifically supports these features. The INC DPTR is lengthened by one cycle but remains byte-count-compatible with the standard 8051 instruction set.

Internally, the device uses a 22-bit program counter. The lowest order 22 bits are used for memory addressing, with a special 23rd bit used to map the 4kB SRAM above the 4MB memory space in bootstrap loader applications. Address bits 16–23 for the 22-bit addressing modes are generated through additional SFRs dependent on the type of instruction as shown in Table 4.

Table 4. Extended Address Generation

INSTRUCTION	ADDRESS BITS 23–16	ADDRESS BITS 15–8	ADDRESS BITS 7–0
MOVX instructions using DPTR	DPX;93h	DPH;83h	DPL;82h
MOVX instructions using DPTR1	DPX1;95h	DPH1;85h	DPL1;84h
MOVX instructions using @Ri	MXAX;EAh	P2;A0h	Ri
Addressing program memory in 22-bit paged mode	AP;9Ch	—	—
10-bit stack pointer mode	—	ESP;9Bh	SP;81h

INTERNAL MOVX SRAM

The DS80C390 contains 4kB of SRAM that can be configured as user accessible MOVX memory, program memory, or optional stack memory. The specific configuration and locations are governed by the internal data memory configuration bits (IDM1, IDM0) in the memory control register (MCON;C6h). Note that when the SA bit (ACON.2) is set, the first 1kB of the MOVX data memory is reserved for use by the 10-bit expanded stack. Internal memory accesses will not generate $\overline{WR}$, $\overline{RD}$, or $\overline{PSEN}$ strobes.

The DS80C390 can configure its 4kB of internal SRAM as combined program and data memory. This allows the application software to execute self-modifiable code. The technique loads the 4kB SRAM with bootstrap loader software, and then modifies the IDM1 and IDM0 bits to map the 4kB starting at memory location 40000h. This allows the system to run the bootstrap loader without disturbing the 4MB external memory bus, making the device in-system reprogrammable for flash or NV RAM.

Table 5. Internal MOVX SRAM Configuration

IDM1	IDM0	CMA	MEMORY		
			MOVX DATA	CAN MESSAGE	SHARED PROGRAM/DATA
0	0	0	00F000h–00FFFFh	00EE00h–00EFFFh	—
0	0	1	00F000h–00FFFFh	401000h–4011FFh	—
0	1	0	000000h–000FFFh	00EE00h–00EFFFh	—
0	1	1	000000h–000FFFh	401000h–4011FFh	—
1	0	0	400000h–400FFFh	00EE00h–00EFFFh	—
1	0	1	400000h–400FFFh	401000h–4011FFh	—
1	1	0	—	00EE00h–00EFFFh	400000h–400FFFh*
1	1	1	—	401000h–4011FFh	400000h–400FFFh*

**10-bit expanded stack is not available in shared program/data memory mode.*

EXTERNAL MEMORY ADDRESSING

The enabling and mapping of the chip-enable signals is done through the Port 4 control register (P4CNT;92h) and memory control register (MCON; 96h). Table 7 shows which chip-enable and address line signals are active on Port 4. Following reset, the device will be configured with P4.7–P4.4 as address lines and P4.3–P4.0 configured as $\overline{CE3\text{-}0}$, with the first program fetch being performed from 00000h with $\overline{CE0}$ active. The following tables illustrate which memory ranges are controlled by each chip enable as a function of which address lines are enabled.

Table 6. External Memory Addressing Pin Assignments

ADDRESS/DATA BUS	$\overline{CE3}$–$\overline{CE0}$	$\overline{PCE3}$–$\overline{PCE0}$	ADDR 19–16	ADDR 15–8	ADDR 7–0	DATA BUS
Multiplexed	P4.3–P4.0	P5.7–P5.4	P4.7–P4.4	P2	P0	P0
Demultiplexed	P4.3–P4.0	P5.7–P5.4	P4.7–P4.4	P2	P1	P0

Table 7. Extended Address and Chip-Enable Generation

P4CNT.5–3	PORT 4 PIN FUNCTION				P4CNT.2–0	PORT 4 PIN FUNCTION			
	P4.7	P4.6	P4.5	P4.4		P4.3	P4.2	P4.1	P4.0
000	I/O	I/O	I/O	I/O	000	I/O	I/O	I/O	I/O
100	I/O	I/O	I/O	A16	100	I/O	I/O	I/O	$\overline{CE0}$
101	I/O	I/O	A17	A16	101	I/O	I/O	$\overline{CE1}$	$\overline{CE0}$
110	I/O	A18	A17	A16	110	I/O	$\overline{CE2}$	$\overline{CE1}$	$\overline{CE0}$
111(default)	A19	A18	A17	A16	111(default)	$\overline{CE3}$	$\overline{CE2}$	$\overline{CE1}$	$\overline{CE0}$

Table 9. Data Memory Cycle Stretch Values

MD2	MD1	MD0	STRETCH CYCLE COUNT	MOVX MACHINE CYCLES	$\overline{RD}$, $\overline{WR}$ PULSE WIDTH (IN OSCILLATOR CLOCKS)			
					t_{MCS} (4X/$\overline{2X}$ = 1 CD1:0 = 00)	t_{MCS} (4X/$\overline{2X}$ = 0 CD1:0 = 00)	t_{MCS} (4X/$\overline{2X}$ = X CD1:0 = 10)	t_{MCS} (4X/$\overline{2X}$ = X CD1:0 = 11)
0	0	0	0*	2	0.5 t_{CLCL}	1 t_{CLCL}	2 t_{CLCL}	2048 t_{CLCL}
0	0	1	1**	3	t_{CLCL}	2 t_{CLCL}	4 t_{CLCL}	4096 t_{CLCL}
0	1	0	2	4	2 t_{CLCL}	4 t_{CLCL}	8 t_{CLCL}	8192 t_{CLCL}
0	1	1	3	5	3 t_{CLCL}	6 t_{CLCL}	12 t_{CLCL}	12,288 t_{CLCL}
1	0	0	4	9	4 t_{CLCL}	8 t_{CLCL}	16 t_{CLCL}	16,384 t_{CLCL}
1	0	1	5	10	5 t_{CLCL}	10 t_{CLCL}	20 t_{CLCL}	20,480 t_{CLCL}
1	1	0	6	11	6 t_{CLCL}	12 t_{CLCL}	24 t_{CLCL}	24,576 t_{CLCL}
1	1	1	7	12	7 t_{CLCL}	14 t_{CLCL}	28 t_{CLCL}	28,672 t_{CLCL}

**All internal MOVX operations execute at the 0 Stretch setting.*
***Default stretch setting for external MOVX operations following reset.*

EXTENDED STACK POINTER

The DS80C390 supports both the traditional 8-bit and an extended 10-bit stack pointer that improves the performance of large programs written in high-level languages such as C. Enable the 10-bit stack pointer feature by setting the stack address mode bit, SA (ACON.2). The bit is cleared following a reset, forcing the device to use an 8-bit stack located in the scratchpad RAM area. When the SA bit is set, the device will address up to 1kB of stack memory in the first 1kB of the internal MOVX memory. The 10-bit stack pointer address is generated by concatenating the lower two bits of the extended stack pointer (ESP;9Bh) and the traditional 8051 stack pointer (SP;81h). The 10-bit stack pointer cannot be enabled when the 4kB of SRAM is mapped as both program and data memory.

ENHANCED DUAL DATA POINTERS

The DS80C390 contains two data pointers, DPTR0 and DPTR1, designed to improve performance in applications that require high data throughput. Incorporating a second data pointer allows the software to greatly speed up block data (MOVX) moves by using one data pointer as a source register and the other as the destination register.

DPTR0 is located at the same address as the original 8051 data pointer, allowing the DS80C390 to execute standard 8051 code with no modifications. The second data pointer, DPTR1, is split between the DPH1 and DPL1 SFRs, similar to the DPTR0 configuration. The active data pointer is selected with the data pointer select bit SEL (DPS.0). Any instructions that reference the DPTR (i.e., MOVX A, @DPTR), will select DPTR0 if SEL = 0, and DPTR1 if SEL = 1. Because the bits adjacent to SEL are not implemented, the state of SEL (and thus the active data pointer) can be quickly toggled by the `INC DPS` instruction without disturbing other bits in the DPS register.

Unlike the standard 8051, the DS80C390 has the ability to decrement as well as increment the data pointers without additional instructions. When the `INC DPTR` instruction is executed, the active DPTR increments or decrements according to the ID1, ID0 (DPS.7-6), and SEL (DPS.0) bits as shown. The inactive DPTR is not affected.

Table 10. Data Pointer Auto Increment/ Decrement Configuration

ID1	ID0	SEL	INC DPTR RESULT
X	0	0	Increment DPTR0
X	1	0	Decrement DPTR0
0	X	1	Increment DPTR1
1	X	1	Decrement DPTR1

Another useful feature of the device is its ability to automatically switch the active data pointer after a DPTR-based instruction is executed. This feature can greatly reduce the software overhead associated with data memory block moves, which toggle between the source and destination registers. 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-related instructions is executed.

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

As a brief example, if TSL is set to 1, then both data pointers can be updated with two INC DPTR instructions. Assume that SEL = 0, making DPTR the active data pointer. The first INC DPTR increments DPTR and toggles SEL to 1. The second instruction increments DPTR1 and toggles SEL back to 0.

```
INC DPTR
INC DPTR
```

CLOCK CONTROL AND POWER MANAGEMENT

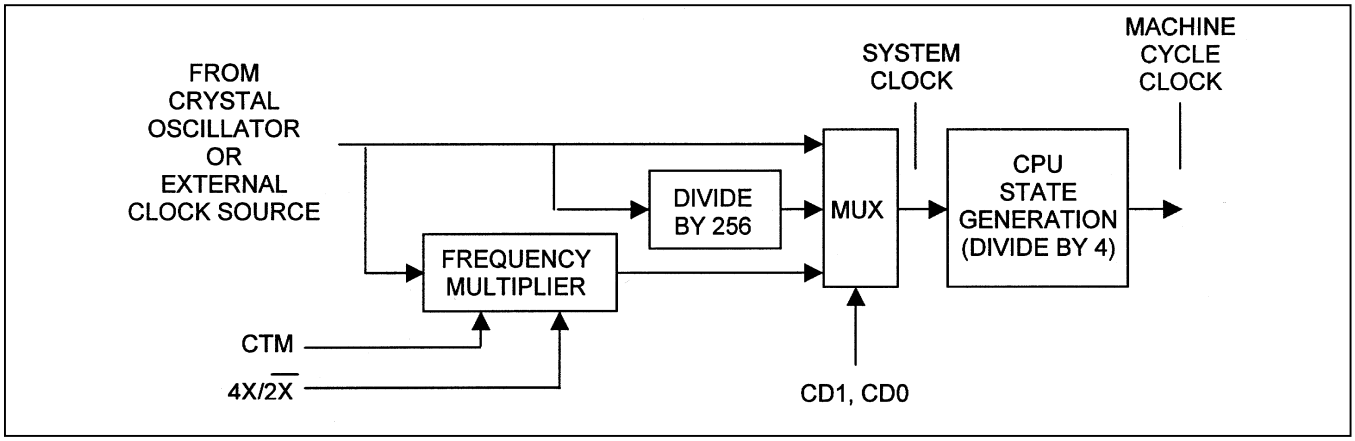
The DS80C390 includes a number of unique features that allow flexibility in selecting system clock sources and operating frequencies. To support the use of inexpensive crystals while allowing full speed operation, a clock multiplier is included in the processor's clock circuit. Also, in addition to the standard 80C32 idle and power-down (Stop) modes, the DS80C390 provides a new power management mode. This mode allows the processor to continue instruction execution, yet at a very low speed to significantly reduce power consumption (below even idle mode). The DS80C390 also features several enhancements to stop mode that make this extremely low-power mode more useful. Each of these features is discussed in detail below.

System Clock Control

As mentioned previously, the microcontroller contains special clock-control circuitry that simultaneously provides maximum timing flexibility and maximum availability and economy in crystal selection. The logical operation of the system clock-divide control function is shown in Figure 29. 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 times

Figure 29. 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 five 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, CD0 to 0 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 twice 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 clocks. Regardless of the configuration of the frequency multiplier, the system clock of

the microcontroller can never be operated faster than 40MHz. This means that the maximum crystal oscillator or external clock source is 10MHz when using the 4X setting, and 20MHz 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 thus less expensive.

Table 11. System Clock Configuration

CD1	CD0	4X/$\overline{2X}$	FUNCTION	CLOCKS PER MACHINE CYCLE	MAX EXTERNAL FREQUENCY (MHz)
0	0	0	Frequency Multiplier (2X)	2	20
0	0	1	Frequency Multiplier (4X)	1	10
0	1	N/A	Reserved	—	—
1	0	N/A	Divide-by-4 (Default)	4	40
1	1	N/A	Power Management Mode	1024	40

The system clock and machine cycle rate changes one machine cycle after the instruction changing 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 problems associated with the PMM.

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 will 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 will cause the CD1, CD0 bits to remain unchanged. Switching from frequency multiplier to non-multiplier 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 CTM (Crystal Multiplier Enable) bit.
3) Set the 4X/2X bit to the appropriate state.
4) Set the CTM (crystal multiplier enable) bit.
5) Poll the CKRDY bit (EXIF.4), waiting until it is set to 1. This will take approximately 65,536 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.

OSCILLATOR-FAIL DETECT

The microprocessor contains a safety mechanism called an on-chip oscillator-fail-detect circuit. When enabled, this circuit causes the processor to be held in reset if the oscillator frequency falls below 40kHz. In operation, this circuit complements the watchdog timer. Normally, the watchdog timer is initialized so that it times out and causes a processor reset in the event that the processor loses control. In the event of a crystal or external oscillator failure, however, the watchdog timer does not function and there is the potential for the processor to fail in an uncontrolled state. The use of the oscillator-fail-detect circuit forces the processor to a known state (i.e., reset) even if the oscillator stops.

The oscillator-fail-detect circuitry is enabled when software sets the enable bit OFDE (PCON.4) to 1. Please note that software must use a timed-access procedure (described later) to write this bit. The OFDF (PCON.5) bit also sets to 1 when the circuitry detects an oscillator failure, and the processor is forced into a reset state. This bit can only be cleared to 0 by a power-fail reset or by software. The oscillator-fail-detect circuitry is not activated when the oscillator is stopped due to the processor entering stop mode.

STOP MODE

Setting the STOP bit of the power control register (PCON.1) invokes stop mode. Stop mode is the lowest power state (besides power off) since it turns off all internal clocking. All processor operation ceases at the end of the instruction that sets the STOP bit. The CPU can exit stop mode via an external interrupt, if enabled, or a reset condition. Internally generated interrupts (timer, serial port, watchdog) cannot cause an exit from stop mode because internal clocks are not active in stop mode.

BANDGAP SELECT

The DS80C390 provides two enhancements to stop mode. As described below, the device provides a band-gap reference to determine power-fail interrupt and reset thresholds. The bandgap select bit, BGS (RCON.0), controls the bandgap reference. Setting BGS to 1 keeps the bandgap reference enabled during stop mode. The default or reset condition of the bit is logic 0, which disables the bandgap during stop mode. This bit has no control of the reference during full power, PMM, or idle modes.

With the bandgap reference enabled, the power-fail reset and interrupt are valid means for leaving stop mode. This allows software to detect and compensate for a power-supply sag or brownout, even when in stop mode. In stop mode with the bandgap enabled, I_{CC} is higher compared to with the bandgap disabled. If a user does not require a power-fail reset or interrupt while in stop mode, the bandgap can remain disabled. Only the most power-sensitive applications should disable the bandgap reference in stop mode, as this results in an uncontrolled power-down condition.

RING OSCILLATOR

The second enhancement to Stop mode reduces power consumption and allows the device to restart instantly when exiting stop mode. The ring oscillator is an internal clock that can optionally provide the clock source to the microcontroller when exiting stop mode in response to an interrupt.

During stop mode the crystal oscillator is halted to maximize power savings. Typically, 4ms to 10ms is required for an external crystal to begin oscillating again once the device receives the exit stimulus. The ring oscillator, by contrast, is a free-running digital oscillator that has no startup delay. Setting the ring oscillator select bit, RGSL (EXIF.1), enables the ring oscillator feature. If enabled, the microcontroller uses the ring oscillator as the clock source to exit stop mode, resuming operation in less than 100ns. After 65,536 oscillations of the external clock source (not the ring oscillator), the device clears the ring-oscillator-mode bit, RGMD (EXIF.2), to indicate that the device has switched from the ring oscillator to the external clock source.

The ring oscillator runs at approximately 10MHz but varies over temperature and voltage. As a result, no serial communication or precision timing should be attempted while running from the ring oscillator since the operating frequency is not precise. The default state exits stop mode without using the ring oscillator.

TIMED-ACCESS PROTECTION

Selected SFR bits are critical to operation, making it desirable to protect them against an accidental write operation. The timed-access procedure prevents an errant processor from accidentally altering bits that would seriously affect processor operation. The timed-access procedure requires that the write of a protected bit be immediately preceded by the following two instructions:

```
MOV   0C7h, #0AAh
MOV   0C7h, #55h
```

Writing an AAh followed by a 55h to the timed-access register (location C7h) opens a three-cycle window that allows software to modify one of the protected bits. If the instruction that seeks to modify the protected bit is not immediately preceded by these instructions, the write is ignored. The protected bits are:

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
RCON.0	BGS	Bandgap Select
ACON.2	SA	Stack Address Mode
ACON.1–0	AM1–AM0	Address Mode Select bits
MCON.7–6	IDM1–IDM0	Internal Memory Configuration and Location bits
MCON.5	CMA	CAN Data Memory Assignment
MCON.3–0	PDCE3–PDCE.0	Program/Data Chip Enables
C0C.3	CRST	CAN 0 Reset
C1C.3	CRST	CAN 1 Reset
P4CNT.6	SBCAN	Single Bus CAN
P4CNT.5–0		Port 4 Pin Configuration Control Bits
P5CNT.2–0	P5.7–P5.5	Configuration Control Bits
COR.7	IRDACK	IRDA Clock Output Enable
COR.6–5	C1BPR7–C1BPR6	CAN 1 Baud Rate Prescale Bits
COR.4–3	C0BPR7–C0BPR6	CAN 0 Baud Rate Prescale Bits
COR.2–1	COD1–COD0	CAN Clock Output Divide Bit 1 and Bit 0
COR.0	CLKOE	CAN Clock Output Enable

EMI REDUCTION

One of the major contributors to radiated noise in an 8051-based system is the toggling of ALE. The microcontroller allows software to disable ALE when not used by setting the ALEOFF (PMR.2) bit to 1. When ALEOFF = 1, ALE automatically toggles during an off-chip MOVX. However, ALE remains static when performing on-chip memory access. The default state of ALEOFF is 0 so ALE normally toggles at a frequency of XTAL/4.

PERIPHERAL OVERVIEW

The DS80C390 provides several of the most commonly needed peripheral functions in microcomputer-based systems. New functions include a second serial port, power-fail reset, power-fail interrupt flag, and a programmable watchdog timer. In addition, the microcontroller contains two CAN modules for industrial communication applications. Each of these peripherals is described in the following paragraphs. More details are available in the *High-Speed Microcontroller User's Guide* and the *DS80C390 Supplement*.

SERIAL PORTS

The microcontroller provides a serial port (UART) that is identical to the 80C52. In addition it includes a second hardware serial port that is a full duplicate of the standard one. This second port optionally uses pins P1.2 (RXD1) and P1.3 (TXD1). It has duplicate control functions included in new SFR locations. The second serial port can alternately be mapped to P5.2 and P5.3 to allow use of both serial ports in nonmultiplexed mode.

Both ports can operate simultaneously but can be at different baud rates or even in different modes. The second serial port has similar control registers (SCON1, SBUF1) to the original. The new serial port can only use Timer 1 for baud-rate generation.

Table 13. Interrupt Summary

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)**	EX0 (IE.0)	PX0 (IP.0)
TF0	Timer 0	0Bh	2	TF0 (TCON.5)*	ET0 (IE.1)	PT0 (IP.1)
INT1	External Interrupt 1	13h	3	IE1 (TCON.3)**	EX1 (IE.2)	PX1 (IP.2)
TF1	Timer 1	1Bh	4	TF1 (TCON.7)*	ET1 (IE.3)	PT1 (IP.3)
SCON0	TI0 or RI0 from Serial Port 0	23h	5	RI_0 (SCON0.0); TI_0 (SCON0.1)	ES0 (IE.4)	PS0 (IP.4)
TF2	Timer 2	2Bh	6	TF2 (T2CON.7)	ET2 (IE.5)	PT2 (IP.7)
SCON1	TI1 or RI1 from Serial Port 1	3Bh	7	RI_1 (SCON1.0); TI_1 (SCON1.1)	ES1 (IE.6)	PS1 (IP.6)
INT2	External Interrupt 2	43h	8	IE2 (EXIF.4)	EX2 (EIE.0)	PX2 (EIP.0)
INT3	External Interrupt 3	4Bh	9	IE3 (EXIF.5)	EX3 (EIE.1)	PX3 (EIP.1)
INT4	External Interrupt 4	53h	10	IE4 (EXIF.6)	EX4 (EIE.2)	PX4 (EIP.2)
INT5	External Interrupt 5	5Bh	11	IE5 (EXIF.7)	EX5 (EIE.3)	PX5 (EIP.3)
C0I	CAN0 Interrupt	6Bh	12	various	C0IE (EIE.6)	C0IP (EIP.6)
C1I	CAN1 Interrupt	73h	13	various	C1IE (EIE.5)	C1IP (EIP.5)
WDTI	Watchdog Timer	63h	14	WDIF (WDCON.3)	EWDI (EIE.4)	PWDI (EIP.4)
CANBUS	CAN0/1 Bus Activity	7Bh	15	various	CANBIE (EIE.7)	CANBIP (EIP.7)

Unless marked, all flags must be cleared by the application software.
**Cleared automatically by hardware when the service routine is entered.*
***If edge-triggered, flag is cleared automatically by hardware when the service routine is entered. If level-triggered, flag follows the state of the interrupt pin.*

CONTROLLER AREA NETWORK (CAN) MODULE

The DS80C390 incorporates two CAN controllers that are fully compliant with the CAN 2.0B specification. CAN is a highly robust, high-performance communication protocol for serial communications. Popular in a wide range of applications including automotive, medical, heating, ventilation, and industrial control, the CAN architecture allows for the construction of sophisticated networks with a minimum of external hardware.

The CAN controllers support the use of 11-bit standard or 29-bit extended acceptance identifiers for up to 15 messages, with the standard 8-byte data field, in each message. Fourteen of the 15 message centers are programmable in either transmit or receive modes, with the 15th designated as a FIFO-buffered, receive-only message center to help prevent data overruns. All message centers support two separate 8-bit media masks and media arbitration fields for incoming message verification. This feature supports the use of higher-level 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 test incoming data with or without the use of the global masks.

Global controls and status registers in each CAN unit allow the microcontroller to evaluate error messages, generate interrupts, locate and validate new data, establish the CAN bus timing, establish identification mask bits, and verify the source of individual messages. Each message center is individually equipped with the necessary status and control bits to establish direction, identification mode (standard or extended), data field size, data status, automatic remote frame request and acknowledgment, and perform masked or non-masked identification acceptance testing.

COMMUNICATING WITH THE CAN MODULE

The microcontroller interface to the CAN modules is divided into two groups of registers. All the global CAN status and control bits as well as the individual message center control/status registers are located in the SFR map. The remaining registers associated with the message centers (data identification, identification/arbitration masks, format, and data) are located in MOVX data space. The CMA bit (MCON.5) allows the message centers to be mapped to either 00EE00h–00EEFFh (CMA = 0) or 401000h–4011FFh (CMA = 1), reducing the possibility of a memory conflict with application software. Note that setting the CMA bit employs a special 23rd address bit that is only used for addressing CAN MOVX memory. The DS80C390's internal architecture requires that the device be in one of the two 22-bit addressing modes when the CMA bit is set to correctly use the 23rd bit and access the CAN MOVX memory. A special lockout feature prevents the accidental software corruption of the control, status, and mask registers while a CAN operation is in progress. Each CAN processor uses 15 message centers. Each message center is composed of four specific areas, including the following:

1) Four arbitration registers (C0MxAR0–3 and C1MxAR0–3) that store either the 11-bit or 29-bit arbitration value. These registers are located in the MOVX memory map.
2) A format register (C0MxF and C1MxF) that informs the CAN processor as to the direction (transmit or receive), the number of data bytes in the message, the identification format (standard or extended), and the optional use of the identification mask or media mask during message evaluation. This register is located in the MOVX memory map.
3) Eight data bytes for storage of 0 to 8 bytes of data (C0MxD0–7 and C1MxD0–7), which are located in the MOVX memory map.
4) Message control registers (C0MxC and C1MxC), which are located in the SFR memory for fast access.

Each of the message centers is identical with the exception of message center 15. Message center 15 has been designed as a receive-only center, and is also buffered through the use of a two-message FIFO to help prevent message loss in a message-overrun situation. The receipt of a third message before either of the first two are read will overwrite the second message, leaving the first message undisturbed.

Modification of the CAN registers located in MOVX memory is protected through the SWINT bits, with one bit protecting each respective CAN module. Consult the description of this bit in the *High-Speed Microcontroller User's Guide: DS80C390 Supplement* for more information. Each CAN module contains a block of control/status/mask registers, 14 functionally identical message centers, plus a 15th message center that is receive-only and incorporates a buffered FIFO. The following tables describe the organization of the message centers located in MOVX space.

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–C0M15D7	CAN 0 MESSAGE 15 DATA BYTE 0–7								xxxxF7h–FEh
	Reserved								xxxxFFh

[1]The first two bytes of the CAN 0 MOVX memory address are dependent on the setting of the CMA bit (MCON.5) CMA = 0, xxxx = 00EE; CMA = 1, xxxx = 4010.

MOVX MESSAGE CENTERS FOR CAN 1

CAN 1 CONTROL/STATUS/MASK REGISTERS

REGISTER	7	6	5	4	3	2	1	0	MOVX DATA ADDRESS[1]
C1MID0	MID07	MID06	MID05	MID04	MID03	MID02	MID01	MID00	xxxx00h
C1MA0	M0AA7	M0AA6	M0AA5	M0AA4	M0AA3	M0AA2	M0AA1	M0AA0	xxxx01h
C1MID1	MID17	MID16	MID15	MID14	MID13	MID12	MID11	MID10	xxxx02h
C1MA1	M1AA7	M1AA6	M1AA5	M1AA4	M1AA3	M1AA2	M1AA1	M1AA0	xxxx03h
C1BT0	SJW1	SJW0	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	xxxx04h
C1BT1	SMP	TSEG26	TSEG25	TSEG24	TSEG13	TSEG12	TSEG11	TSEG10	xxxx05h
C1SGM0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	xxxx06h
C1SGM1	ID20	ID19	ID18	0	0	0	0	0	xxxx07h
C1EGM0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	xxxx08h
C1EGM1	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	xxxx09h
C1EGM2	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	xxxx0Ah
C1EGM3	ID4	ID3	ID2	ID1	ID0	0	0	0	xxxx0Bh
C1M15M0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	xxxx0Ch
C1M15M1	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	xxxx0Dh
C1M15M2	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	xxxx0Eh
C1M15M3	ID4	ID3	ID2	ID1	ID0	0	0	0	xxxx0Fh

CAN 1 MESSAGE CENTER 1

	Reserved								xxxx10h–11h
C1M1AR0	CAN 1 MESSAGE 1 ARBITRATION REGISTER 0								xxxx12h
C1M1AR1	CAN 1 MESSAGE 1 ARBITRATION REGISTER 1								xxxx13h
C1M1AR2	CAN 1 MESSAGE 1 ARBITRATION REGISTER 2								xxxx14h
C1M1AR3	CAN 1 MESSAGE 1 ARBITRATION REGISTER 3							WTOE	xxxx15h
C1M1F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	T/$\overline{R}$	EX/$\overline{ST}$	MEME	MDME	xxxx16h
C1M1D0–7	CAN 1 MESSAGE 1 DATA BYTES 0–7								xxxx17h–1Eh
	Reserved								xxxx1Fh

CAN 1 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 1 MESSAGE CENTER 15

—	Reserved								xxxxF0h–F1h
C1M15AR0	CAN 1 MESSAGE 15 ARBITRATION REGISTER 0								xxxxF2h
C1M15AR1	CAN 1 MESSAGE 15 ARBITRATION REGISTER 1								xxxxF3h
C1M15AR2	CAN 1 MESSAGE 15 ARBITRATION REGISTER 2								xxxxF4h
C1M15AR3	CAN 1 MESSAGE 15 ARBITRATION REGISTER 3							WTOE	xxxxF5h
C1M15F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	0	EX/$\overline{ST}$	MEME	MDME	xxxxF6h
C1M15D0–C1M15D7	CAN 1 MESSAGE 15 DATA BYTE 0–7								xxxxF7h–FEh
	Reserved								xxxxFFh

[1]The first two bytes of the CAN 1 MOVX memory address are dependent on the setting of the CMA bit (MCON.5) CMA = 0, xxxx = 00EF; CMA = 1, xxxx = 4011.

CAN INTERRUPTS

The DS80C390 supports three interrupts associated with the CAN controllers. One interrupt is dedicated to each CAN controller, providing receive/transmit acknowledgments from each of its 15 message centers. The remaining interrupt, the CAN bus activity interrupt, is used to detect CAN bus activity on the C0RX or C1RX pins.

The message center interrupts are enabled/disabled by individual ETI (transmit) and ERI (receive) enable bits in the corresponding message control register (located in SFR memory) for each message center. All the message center interrupts of each CAN module are ORed together into their respective CAN interrupt. The successful transmission or receipt of a message sets the INTRQ bit in the corresponding message control register (located in SFR memory). This bit can only be cleared through software. In addition, the global interrupt-enable bit (IE.7) and the specific CAN interrupt-enable bit, EIE.6 (CAN0) or EIE.5 (CAN1), must be correctly set to acknowledge a message center interrupt.

Interrupt assertion of error and status conditions associated with the CAN modules is controlled by the ERIE and STIE bits located in the CAN control registers, C0C and C1C.

ARBITRATION AND MASKING

After a CAN module has ascertained that an incoming message is bit-error-free, the identification field of that message is then compared against one or more arbitration values to determine if they will be loaded into a message center. Each enabled message center (see the MSRDY bit in the *CAN Message Control Register*) is tested in order from 1 to 15. The first message center to successfully pass the test receives the incoming message and ends the testing. Using masking registers allows the use of more complex identification schemes, as tests can be made based on bit patterns rather than an exact match between all bits in the identification field and arbitration values. Each CAN processor also incorporates a set of five masks to allow messages with different IDs to be grouped and successfully loaded into a message center. Note that some of these masks are optional as per the bits shown in the *Arbitration/Masking Feature Summary* table (Table 14).

There are several possible arbitration tests, varying according to which message center is involved. If all the enabled tests succeed, the message is loaded into the respective message center. The most basic test, performed on all messages, compares either 11 (CAN 2.0A) or 29 (CAN 2.0B) bits of the identification field to the appropriate arbitration register, based on the EX/$\overline{ST}$ bit in the CAN 0/1 format register. The MEME bit (C0MxF.1 or C1MxF.1) controls whether the arbitration and ID registers are compared directly or through a mask register. A special set of arbitration registers dedicated to message center 15 allows added flexibility in filtering this location.

If desired, further arbitration can be performed by comparing the first two bytes of the data field in each message against two 8-bit media arbitration register bytes. The MDME bit in the CAN message center format registers (C0MxF.0 or C1MxF.0) either disables (MDME = 0) arbitration, or enables (MDME = 1) arbitration using the media ID mask registers 0–1.

If the 11-bit or 29-bit arbitration and the optional media-byte arbitration are successful, the message is loaded into the respective message center. The format register also allows the microcontroller to program each message center to function in a receive or transmit mode through the T/$\overline{R}$ bit, and to use from 0 to 8 data bytes within the data field of a message. Note that message center 15 can only be used in a receive mode. To avoid a priority inversion, the DS80C390 CAN processors are configured 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.