

Welcome to [E-XFL.COM](https://www.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	Not For New Designs
Core Processor	8051
Core Size	8-Bit
Speed	25MHz
Connectivity	SMBus (2-Wire/I <sup>2</sup> C), SPI, UART/USART
Peripherals	Cap Sense, POR, PWM, WDT
Number of I/O	13
Program Memory Size	8KB (8K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	256 x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	-
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	16-SOIC (0.154", 3.90mm Width)
Supplier Device Package	16-SOIC
Purchase URL	<a href="https://www.e-xfl.com/product-detail/silicon-labs/c8051f828-gs">https://www.e-xfl.com/product-detail/silicon-labs/c8051f828-gs</a>

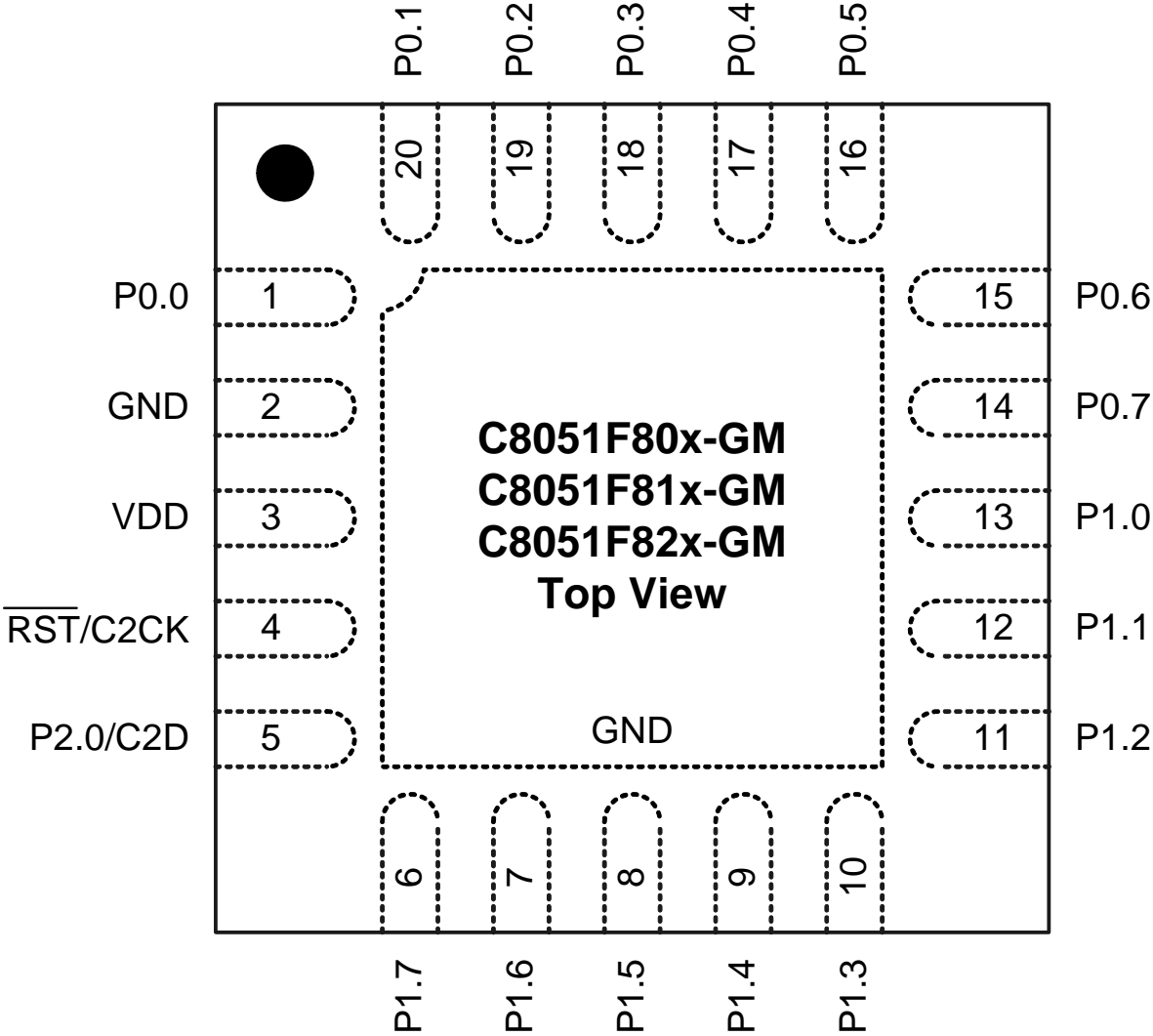


Figure 3.1. QFN-20 Pinout Diagram (Top View)

## 8.4.1. Window Detector Example

Figure 8.4 shows two example window comparisons for right-justified data, with  $ADC0LTH:ADC0LTL = 0x0080$  (128d) and  $ADC0GTH:ADC0GTL = 0x0040$  (64d). The input voltage can range from 0 to  $VREF \times (1023/1024)$  with respect to GND, and is represented by a 10-bit unsigned integer value. In the left example, an  $AD0WINT$  interrupt will be generated if the  $ADC0$  conversion word ( $ADC0H:ADC0L$ ) is within the range defined by  $ADC0GTH:ADC0GTL$  and  $ADC0LTH:ADC0LTL$  (if  $0x0040 < ADC0H:ADC0L < 0x0080$ ). In the right example, an  $AD0WINT$  interrupt will be generated if the  $ADC0$  conversion word is outside of the range defined by the  $ADC0GT$  and  $ADC0LT$  registers (if  $ADC0H:ADC0L < 0x0040$  or  $ADC0H:ADC0L > 0x0080$ ). Figure 8.5 shows an example using left-justified data with the same comparison values.

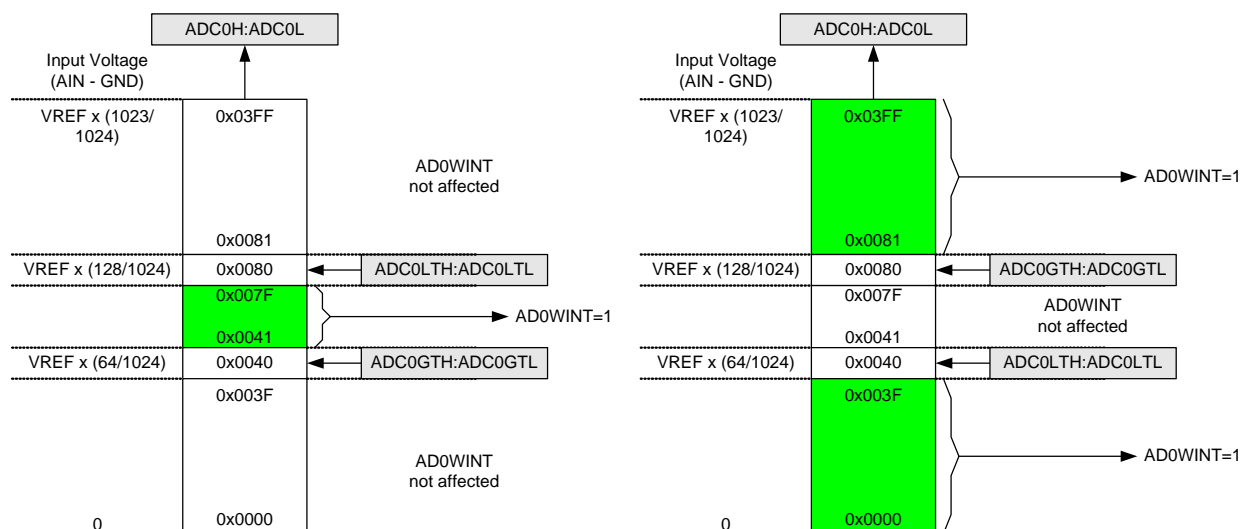


Figure 8.4. ADC Window Compare Example: Right-Justified Data

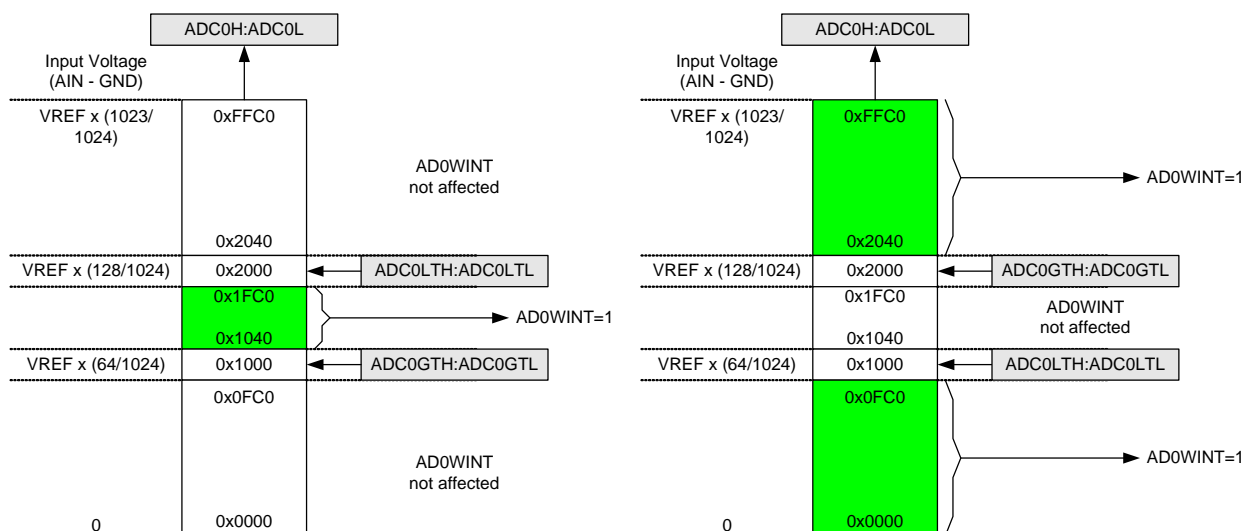


Figure 8.5. ADC Window Compare Example: Left-Justified Data

## SFR Definition 10.1. REF0CN: Voltage Reference Control

Bit	7	6	5	4	3	2	1	0
Name			REFGND	REFSL		TEMPE	BIASE	
Type	R	R	R/W	R/W	R/W	R/W	R/W	R
Reset	0	0	0	1	0	0	0	0

SFR Address = 0xD1

Bit	Name	Function
7:6	Unused	Read = 00b; Write = Don't Care.
5	REFGND	<b>Analog Ground Reference.</b> Selects the ADC0 ground reference. 0: The ADC0 ground reference is the GND pin. 1: The ADC0 ground reference is the P0.1/AGND pin.
4:3	REFSL	<b>Voltage Reference Select.</b> Selects the ADC0 voltage reference. 00: The ADC0 voltage reference is the P0.0/VREF pin. 01: The ADC0 voltage reference is the VDD pin. 10: The ADC0 voltage reference is the internal 1.8 V digital supply voltage. 11: The ADC0 voltage reference is the internal 1.65 V high speed voltage reference.
2	TEMPE	<b>Temperature Sensor Enable.</b> Enables/Disables the internal temperature sensor. 0: Temperature Sensor Disabled. 1: Temperature Sensor Enabled.
1	BIASE	<b>Internal Analog Bias Generator Enable Bit.</b> 0: Internal Bias Generator off. 1: Internal Bias Generator on.
0	Unused	Read = 0b; Write = Don't Care.

# C8051F80x-83x

## SFR Definition 12.3. CPT0MX: Comparator0 MUX Selection

Bit	7	6	5	4	3	2	1	0
Name	CMX0N[3:0]				CMX0P[3:0]			
Type	R/W				R/W			
Reset	1	1	1	1	1	1	1	1

SFR Address = 0x9F

Bit	Name	Function	
7:4	CMX0N[3:0]	<b>Comparator0 Negative Input MUX Selection.</b>	
		<b>20-Pin and 24-Pin Devices</b>	<b>16-Pin Devices</b>
		0000	P0.1
		0001	P0.3
		0010	P0.5
		0011	P0.7
		0100	P1.1
		0101	P1.3
		0110	P1.5
		0111	P1.7
		1000	VREG Output.
		1001–1111	No input selected.
3:0	CMX0P[3:0]	<b>Comparator0 Positive Input MUX Selection.</b>	
		<b>20-Pin and 24-Pin Devices</b>	<b>16-Pin Devices</b>
		0000	P0.0
		0001	P0.2
		0010	P0.4
		0011	P0.6
		0100	P1.0
		0101	P1.2
		0110	P1.4
		0111	P1.6
		1000	VREG Output.
		1001–1111	No input selected.

# C8051F80x-83x

With the CIP-51's maximum system clock at 25 MHz, it has a peak throughput of 25 MIPS. The CIP-51 has a total of 109 instructions. The table below shows the total number of instructions that require each execution time.

Clocks to Execute	1	2	2/3	3	3/4	4	4/5	5	8
Number of Instructions	26	50	5	14	6	3	2	2	1

## 14.1. Instruction Set

The instruction set of the CIP-51 System Controller is fully compatible with the standard MCS-51™ instruction set. Standard 8051 development tools can be used to develop software for the CIP-51. All CIP-51 instructions are the binary and functional equivalent of their MCS-51™ counterparts, including opcodes, addressing modes and effect on PSW flags. However, instruction timing is different than that of the standard 8051.

### 14.1.1. Instruction and CPU Timing

In many 8051 implementations, a distinction is made between machine cycles and clock cycles, with machine cycles varying from 2 to 12 clock cycles in length. However, the CIP-51 implementation is based solely on clock cycle timing. All instruction timings are specified in terms of clock cycles.

Due to the pipelined architecture of the CIP-51, most instructions execute in the same number of clock cycles as there are program bytes in the instruction. Conditional branch instructions take one less clock cycle to complete when the branch is not taken as opposed to when the branch is taken. Table 14.1 is the CIP-51 Instruction Set Summary, which includes the mnemonic, number of bytes, and number of clock cycles for each instruction.

## 16. In-System Device Identification

The C8051F80x-83x has SFRs that identify the device family and derivative. These SFRs can be read by firmware at runtime to determine the capabilities of the MCU that is executing code. This allows the same firmware image to run on MCUs with different memory sizes and peripherals, and dynamically changing functionality to suit the capabilities of that MCU.

In order for firmware to identify the MCU, it must read three SFRs. HWID describes the MCU's family, DERIVID describes the specific derivative within that device family, and REVID describes the hardware revision of the MCU.

### SFR Definition 16.1. HWID: Hardware Identification Byte

Bit	7	6	5	4	3	2	1	0
Name	HWID[7:0]							
Type	R	R	R	R	R	R	R	R
Reset	0	0	1	0	0	0	1	1

SFR Address = 0xB5

Bit	Name	Description
7:0	HWID[7:0]	<b>Hardware Identification Byte.</b> Describes the MCU family. 0x23: Devices covered in this document (C8051F80x-83x)

**Table 18.1. Interrupt Summary**

Interrupt Source	Interrupt Vector	Priority Order	Pending Flag	Bit addressable?	Cleared by HW?	Enable Flag	Priority Control
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 (INT0)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Timer 0 Overflow	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
External Interrupt 1 (INT1)	0x0013	2	IE1 (TCON.3)	Y	Y	EX1 (IE.2)	PX1 (IP.2)
Timer 1 Overflow	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
UART0	0x0023	4	RI0 (SCON0.0) TI0 (SCON0.1)	Y	N	ES0 (IE.4)	PS0 (IP.4)
Timer 2 Overflow	0x002B	5	TF2H (TMR2CN.7) TF2L (TMR2CN.6)	Y	N	ET2 (IE.5)	PT2 (IP.5)
SPI0	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y		ESPI0 (IE.6)	PSPI0 (IP.6)
SMB0	0x003B	7	SI (SMB0CN.0)	Y	N	ESMB0 (EIE1.0)	PSMB0 (EIP1.0)
Port Match	0x0043	8	None	N/A	N/A	EMAT (EIE1.1)	PMAT (EIP1.1)
ADC0 Window Compare	0x004B	9	AD0WINT (ADC0CN.3)	Y	N	EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
ADC0 Conversion Complete	0x0053	10	AD0INT (ADC0CN.5)	Y	N	EADC0 (EIE1.3)	PADC0 (EIP1.3)
Programmable Counter Array	0x005B	11	CF (PCA0CN.7) CCFn (PCA0CN.n)	Y	N	EPCA0 (EIE1.4)	PPCA0 (EIP1.4)
Comparator0	0x0063	12	CP0FIF (CPT0CN.4) CP0RIF (CPT0CN.5)	N	N	ECP0 (EIE1.5)	PCP0 (EIP1.5)
RESERVED							
RESERVED							
CS0 Conversion Complete	0x007B	15	CS0INT (CS0CN.5)	N	N	ECSCPT (EIE2.0)	PSCCPT (EIP2.0)
CS0 Greater Than	0x0083	16	CS0CMPF (CS0CN.0)	N	N	ECSGRT (EIE2.1)	PSCGRT (EIP2.1)

## 18.2. Interrupt Register Descriptions

The SFRs used to enable the interrupt sources and set their priority level are described in this section. Refer to the data sheet section associated with a particular on-chip peripheral for information regarding valid interrupt conditions for the peripheral and the behavior of its interrupt-pending flag(s).

# C8051F80x-83x

The following guidelines are recommended for any system that contains routines which write or erase Flash from code.

## 19.4.1. VDD Maintenance and the VDD Monitor

1. If the system power supply is subject to voltage or current "spikes," add sufficient transient protection devices to the power supply to ensure that the supply voltages listed in the Absolute Maximum Ratings table are not exceeded.
2. Make certain that the minimum VDD rise time specification of 1 ms is met. If the system cannot meet this rise time specification, then add an external VDD brownout circuit to the RST pin of the device that holds the device in reset until VDD reaches the minimum device operating voltage and re-asserts RST if VDD drops below the minimum device operating voltage.
3. Keep the on-chip VDD Monitor enabled and enable the VDD Monitor as a reset source as early in code as possible. This should be the first set of instructions executed after the Reset Vector. For C-based systems, this will involve modifying the startup code added by the C compiler. See your compiler documentation for more details. Make certain that there are no delays in software between enabling the VDD Monitor and enabling the VDD Monitor as a reset source. Code examples showing this can be found in "AN201: Writing to Flash from Firmware," available from the Silicon Laboratories website.

**Note:** On C8051F80x-83x devices, both the VDD Monitor and the VDD Monitor reset source must be enabled to write or erase Flash without generating a Flash Error Device Reset.

On C8051F80x-83x devices, both the VDD Monitor and the VDD Monitor reset source are enabled by hardware after a power-on reset.

4. As an added precaution, explicitly enable the VDD Monitor and enable the VDD Monitor as a reset source inside the functions that write and erase Flash memory. The VDD Monitor enable instructions should be placed just after the instruction to set PSWE to a 1, but before the Flash write or erase operation instruction.
5. Make certain that all writes to the RSTSRC (Reset Sources) register use direct assignment operators and explicitly DO NOT use the bit-wise operators (such as AND or OR). For example, "RSTSRC = 0x02" is correct, but "RSTSRC |= 0x02" is incorrect.
6. Make certain that all writes to the RSTSRC register explicitly set the PORSF bit to a 1. Areas to check are initialization code which enables other reset sources, such as the Missing Clock Detector or Comparator, for example, and instructions which force a Software Reset. A global search on "RSTSRC" can quickly verify this.

## 19.4.2. PSWE Maintenance

1. Reduce the number of places in code where the PSWE bit (b0 in PSCTL) is set to a 1. There should be exactly one routine in code that sets PSWE to a 1 to write Flash bytes and one routine in code that sets both PSWE and PSEE both to a 1 to erase Flash pages.
2. Minimize the number of variable accesses while PSWE is set to a 1. Handle pointer address updates and loop maintenance outside the "PSWE = 1;... PSWE = 0;" area. Code examples showing this can be found in "AN201: Writing to Flash from Firmware," available from the Silicon Laboratories website.
3. Disable interrupts prior to setting PSWE to a 1 and leave them disabled until after PSWE has been reset to 0. Any interrupts posted during the Flash write or erase operation will be serviced in priority order after the Flash operation has been completed and interrupts have been re-enabled by software.
4. Make certain that the Flash write and erase pointer variables are not located in XRAM. See your compiler documentation for instructions regarding how to explicitly locate variables in different memory areas.
5. Add address bounds checking to the routines that write or erase Flash memory to ensure that a routine called with an illegal address does not result in modification of the Flash.

---

## 21.2. Power-Fail Reset / $V_{DD}$ Monitor

When a power-down transition or power irregularity causes  $V_{DD}$  to drop below  $V_{RST}$ , the power supply monitor will drive the  $\overline{RST}$  pin low and hold the CIP-51 in a reset state (see Figure 21.2). When  $V_{DD}$  returns to a level above  $V_{RST}$ , the CIP-51 will be released from the reset state. Even though internal data memory contents are not altered by the power-fail reset, it is impossible to determine if  $V_{DD}$  dropped below the level required for data retention. If the PORSF flag reads 1, the data may no longer be valid. The  $V_{DD}$  monitor is enabled and selected as a reset source after power-on resets. Its defined state (enabled/disabled) is not altered by any other reset source. For example, if the  $V_{DD}$  monitor is disabled by code and a software reset is performed, the  $V_{DD}$  monitor will still be disabled after the reset.

**Important Note:** If the  $V_{DD}$  monitor is being turned on from a disabled state, it should be enabled before it is selected as a reset source. Selecting the  $V_{DD}$  monitor as a reset source before it is enabled and stabilized may cause a system reset. In some applications, this reset may be undesirable. If this is not desirable in the application, a delay should be introduced between enabling the monitor and selecting it as a reset source. The procedure for enabling the  $V_{DD}$  monitor and configuring it as a reset source from a disabled state is shown below:

1. Enable the  $V_{DD}$  monitor (VDMEN bit in VDM0CN = 1).
2. If necessary, wait for the  $V_{DD}$  monitor to stabilize.
3. Select the  $V_{DD}$  monitor as a reset source (PORSF bit in RSTSRC = 1).

See Figure 21.2 for  $V_{DD}$  monitor timing; note that the power-on-reset delay is not incurred after a  $V_{DD}$  monitor reset. See Section “7. Electrical Characteristics” on page 39 for complete electrical characteristics of the  $V_{DD}$  monitor.

**Table 23.1. Port I/O Assignment for Analog Functions**

Analog Function	Potentially Assignable Port Pins	SFR(s) used for Assignment
ADC Input	P0.0–P1.7	ADC0MX, PnSKIP, PnMDIN
Comparator0 Input	P0.0–P1.7	CPT0MX, PnSKIP, PnMDIN
CS0 Input	P0.0–P1.7	CS0MX, CS0SS, CS0SE, PnMDIN
Voltage Reference (VREF0)	P0.0	REF0CN, P0SKIP, PnMDIN
Ground Reference (AGND)	P0.1	REF0CN, P0SKIP
External Oscillator in Crystal Mode (XTAL1)	P0.2	OSCXCN, P0SKIP, P0MDIN
External Oscillator in RC, C, or Crystal Mode (XTAL2)	P0.3	OSCXCN, P0SKIP, P0MDIN

## 23.2.2. Assigning Port I/O Pins to Digital Functions

Any Port pins not assigned to analog functions may be assigned to digital functions or used as GPIO. Most digital functions rely on the Crossbar for pin assignment; however, some digital functions bypass the Crossbar in a manner similar to the analog functions listed above. **Port pins used by these digital functions and any Port pins selected for use as GPIO should have their corresponding bit in PnSKIP set to 1.** Table 23.2 shows all available digital functions and the potential mapping of Port I/O to each digital function.

## 24.2. 32-bit CRC Algorithm

The C8051F80x-83x CRC unit calculates the 32-bit CRC using a poly of 0x04C11DB7. The CRC-32 algorithm is "reflected", meaning that all of the input bytes and the final 32-bit output are bit-reversed in the processing engine. The following is a description of a simplified CRC algorithm that produces results identical to the hardware:

1. XOR the least-significant byte of the current CRC result with the input byte. If this is the first iteration of the CRC unit, the current CRC result will be the set initial value (0x00000000 or 0xFFFFFFFF).
2. Right-shift the CRC result.
3. If the LSB of the CRC result is set, XOR the CRC result with the reflected polynomial (0xEDB88320).
4. Repeat at Step 2 for the number of input bits (8).

For example, the 32-bit C8051F80x-83x CRC algorithm can be described by the following code:

```
unsigned long UpdateCRC (unsigned long CRC_acc, unsigned char CRC_input){
    unsigned char i; // loop counter
    #define POLY 0xEDB88320 // bit-reversed version of the poly 0x04C11DB7
    // Create the CRC "dividend" for polynomial arithmetic (binary arithmetic
    // with no carries)
    CRC_acc = CRC_acc ^ CRC_input;
    // "Divide" the poly into the dividend using CRC XOR subtraction
    // CRC_acc holds the "remainder" of each divide
    // Only complete this division for 8 bits since input is 1 byte
    for (i = 0; i < 8; i++)
    {
        // Check if the MSB is set (if MSB is 1, then the POLY can "divide"
        // into the "dividend")
        if ((CRC_acc & 0x00000001) == 0x00000001)
        {
            // if so, shift the CRC value, and XOR "subtract" the poly
            CRC_acc = CRC_acc >> 1;
            CRC_acc ^= POLY;
        }
        else
        {
            // if not, just shift the CRC value
            CRC_acc = CRC_acc >> 1;
        }
    }
    return CRC_acc; // Return the final remainder (CRC value)
}
```

Table 24.2 lists example input values and the associated outputs using the 32-bit C8051F80x-83x CRC algorithm (an initial value of 0xFFFFFFFF is used):

**Table 24.2. Example 32-bit CRC Outputs**

Input	Output
0x63	0xF9462090
0xAA, 0xBB, 0xCC	0x41B207B3
0x00, 0x00, 0xAA, 0xBB, 0xCC	0x78D129BC

**SFR Definition 26.1. SMB0CF: SMBus Clock/Configuration**

Bit	7	6	5	4	3	2	1	0
Name	ENSMB	INH	BUSY	EXTHOLD	SMBTOE	SMBFTE	SMBCS[1:0]	
Type	R/W	R/W	R	R/W	R/W	R/W	R/W	
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xC1

Bit	Name	Function
7	ENSMB	<b>SMBus Enable.</b> This bit enables the SMBus interface when set to 1. When enabled, the interface constantly monitors the SDA and SCL pins.
6	INH	<b>SMBus Slave Inhibit.</b> When this bit is set to logic 1, the SMBus does not generate an interrupt when slave events occur. This effectively removes the SMBus slave from the bus. Master Mode interrupts are not affected.
5	BUSY	<b>SMBus Busy Indicator.</b> This bit is set to logic 1 by hardware when a transfer is in progress. It is cleared to logic 0 when a STOP or free-timeout is sensed.
4	EXTHOLD	<b>SMBus Setup and Hold Time Extension Enable.</b> This bit controls the SDA setup and hold times according to Table 26.2. 0: SDA Extended Setup and Hold Times disabled. 1: SDA Extended Setup and Hold Times enabled.
3	SMBTOE	<b>SMBus SCL Timeout Detection Enable.</b> This bit enables SCL low timeout detection. If set to logic 1, the SMBus forces Timer 3 to reload while SCL is high and allows Timer 3 to count when SCL goes low. If Timer 3 is configured to Split Mode, only the High Byte of the timer is held in reload while SCL is high. Timer 3 should be programmed to generate interrupts at 25 ms, and the Timer 3 interrupt service routine should reset SMBus communication.
2	SMBFTE	<b>SMBus Free Timeout Detection Enable.</b> When this bit is set to logic 1, the bus will be considered free if SCL and SDA remain high for more than 10 SMBus clock source periods.
1:0	SMBCS[1:0]	<b>SMBus Clock Source Selection.</b> These two bits select the SMBus clock source, which is used to generate the SMBus bit rate. The selected device should be configured according to Equation 26.1. 00: Timer 0 Overflow 01: Timer 1 Overflow 10: Timer 2 High Byte Overflow 11: Timer 2 Low Byte Overflow

**Table 26.3. Sources for Hardware Changes to SMB0CN**

Bit	Set by Hardware When:	Cleared by Hardware When:
MASTER	<ul style="list-style-type: none"> <li>■ A START is generated.</li> </ul>	<ul style="list-style-type: none"> <li>■ A STOP is generated.</li> <li>■ Arbitration is lost.</li> </ul>
TXMODE	<ul style="list-style-type: none"> <li>■ START is generated.</li> <li>■ SMB0DAT is written before the start of an SMBus frame.</li> </ul>	<ul style="list-style-type: none"> <li>■ A START is detected.</li> <li>■ Arbitration is lost.</li> <li>■ SMB0DAT is not written before the start of an SMBus frame.</li> </ul>
STA	<ul style="list-style-type: none"> <li>■ A START followed by an address byte is received.</li> </ul>	<ul style="list-style-type: none"> <li>■ Must be cleared by software.</li> </ul>
STO	<ul style="list-style-type: none"> <li>■ A STOP is detected while addressed as a slave.</li> <li>■ Arbitration is lost due to a detected STOP.</li> </ul>	<ul style="list-style-type: none"> <li>■ A pending STOP is generated.</li> </ul>
ACKRQ	<ul style="list-style-type: none"> <li>■ A byte has been received and an ACK response value is needed (only when hardware ACK is not enabled).</li> </ul>	<ul style="list-style-type: none"> <li>■ After each ACK cycle.</li> </ul>
ARBLOST	<ul style="list-style-type: none"> <li>■ A repeated START is detected as a MASTER when STA is low (unwanted repeated START).</li> <li>■ SCL is sensed low while attempting to generate a STOP or repeated START condition.</li> <li>■ SDA is sensed low while transmitting a 1 (excluding ACK bits).</li> </ul>	<ul style="list-style-type: none"> <li>■ Each time SI is cleared.</li> </ul>
ACK	<ul style="list-style-type: none"> <li>■ The incoming ACK value is low (ACKNOWLEDGE).</li> </ul>	<ul style="list-style-type: none"> <li>■ The incoming ACK value is high (NOT ACKNOWLEDGE).</li> </ul>
SI	<ul style="list-style-type: none"> <li>■ A START has been generated.</li> <li>■ Lost arbitration.</li> <li>■ A byte has been transmitted and an ACK/NACK received.</li> <li>■ A byte has been received.</li> <li>■ A START or repeated START followed by a slave address + R/W has been received.</li> <li>■ A STOP has been received.</li> </ul>	<ul style="list-style-type: none"> <li>■ Must be cleared by software.</li> </ul>

## 26.4.3. Hardware Slave Address Recognition

The SMBus hardware has the capability to automatically recognize incoming slave addresses and send an ACK without software intervention. Automatic slave address recognition is enabled by setting the EHACK bit in register SMB0ADM to 1. This will enable both automatic slave address recognition and automatic hardware ACK generation for received bytes (as a master or slave). More detail on automatic hardware ACK generation can be found in Section 26.4.2.2.

The registers used to define which address(es) are recognized by the hardware are the SMBus Slave Address register (SFR Definition 26.3) and the SMBus Slave Address Mask register (SFR Definition 26.4). A single address or range of addresses (including the General Call Address 0x00) can be specified using these two registers. The most-significant seven bits of the two registers are used to define which addresses will be ACKed. A 1 in bit positions of the slave address mask SLVM[6:0] enable a comparison between the received slave address and the hardware's slave address SLV[6:0] for those bits. A 0 in a bit of the slave address mask means that bit will be treated as a "don't care" for comparison purposes. In this

### 26.5.2. Read Sequence (Master)

During a read sequence, an SMBus master reads data from a slave device. The master in this transfer will be a transmitter during the address byte, and a receiver during all data bytes. The SMBus interface generates the START condition and transmits the first byte containing the address of the target slave and the data direction bit. In this case the data direction bit (R/W) will be logic 1 (READ). Serial data is then received from the slave on SDA while the SMBus outputs the serial clock. The slave transmits one or more bytes of serial data.

If hardware ACK generation is disabled, the ACKRQ is set to 1 and an interrupt is generated after each received byte. Software must write the ACK bit at that time to ACK or NACK the received byte.

With hardware ACK generation enabled, the SMBus hardware will automatically generate the ACK/NACK, and then post the interrupt. **It is important to note that the appropriate ACK or NACK value should be set up by the software prior to receiving the byte when hardware ACK generation is enabled.**

Writing a 1 to the ACK bit generates an ACK; writing a 0 generates a NACK. Software should write a 0 to the ACK bit for the last data transfer, to transmit a NACK. The interface exits Master Receiver Mode after the STO bit is set and a STOP is generated. The interface will switch to Master Transmitter Mode if SMB0-DAT is written while an active Master Receiver. Figure 26.6 shows a typical master read sequence. Two received data bytes are shown, though any number of bytes may be received. Notice that the 'data byte transferred' interrupts occur at different places in the sequence, depending on whether hardware ACK generation is enabled. The interrupt occurs **before** the ACK with hardware ACK generation disabled, and **after** the ACK when hardware ACK generation is enabled.

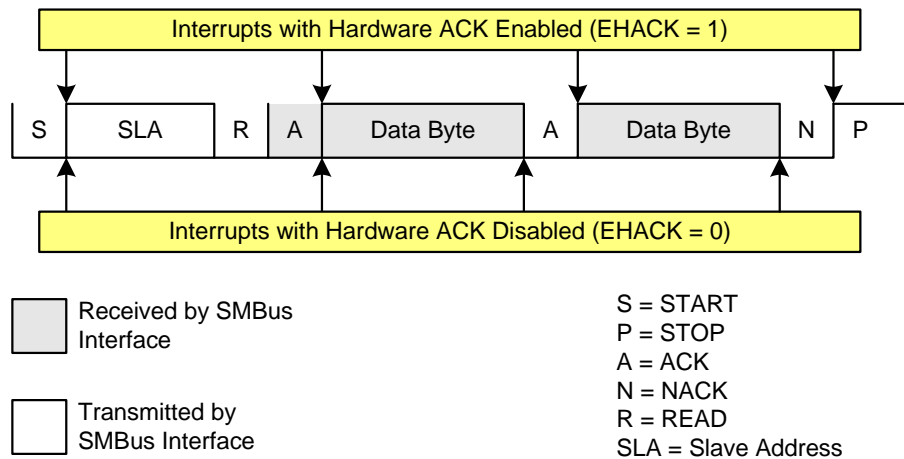


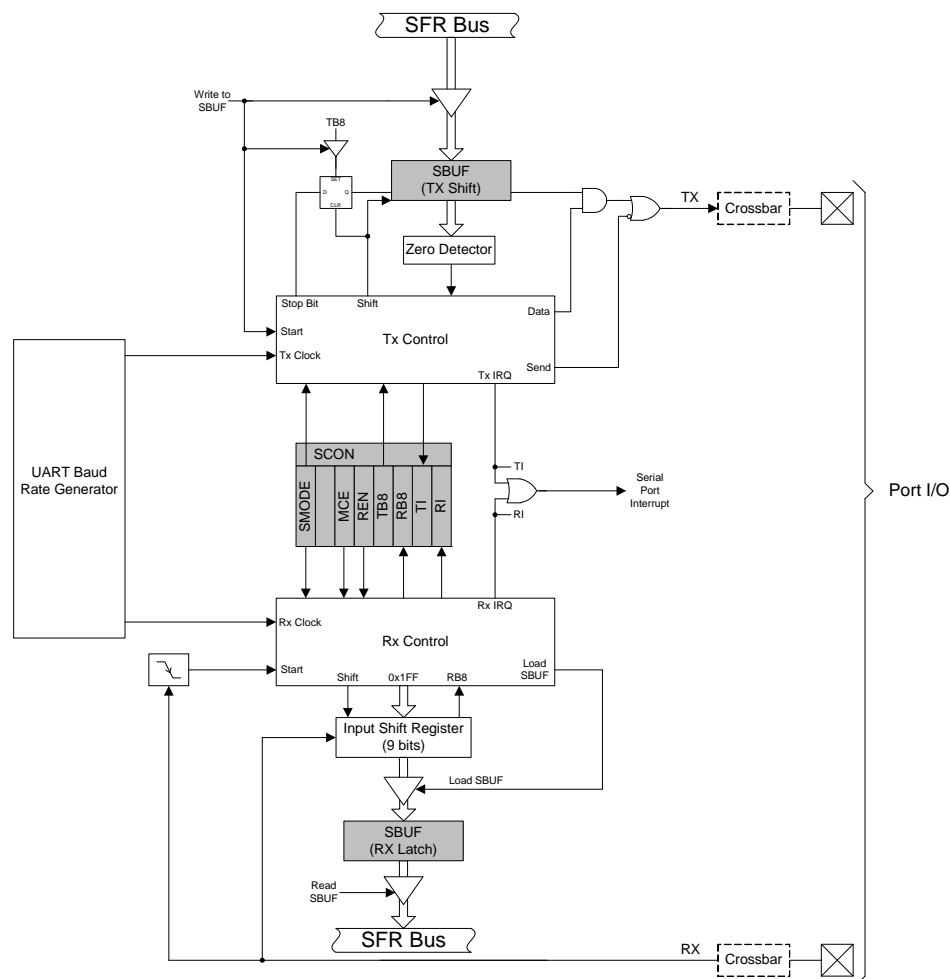
Figure 26.6. Typical Master Read Sequence

## 27. UART0

UART0 is an asynchronous, full duplex serial port offering modes 1 and 3 of the standard 8051 UART. Enhanced baud rate support allows a wide range of clock sources to generate standard baud rates (details in Section “27.1. Enhanced Baud Rate Generation” on page 202). Received data buffering allows UART0 to start reception of a second incoming data byte before software has finished reading the previous data byte.

UART0 has two associated SFRs: Serial Control Register 0 (SCON0) and Serial Data Buffer 0 (SBUF0). The single SBUF0 location provides access to both transmit and receive registers. **Writes to SBUF0 always access the Transmit register. Reads of SBUF0 always access the buffered Receive register; it is not possible to read data from the Transmit register.**

With UART0 interrupts enabled, an interrupt is generated each time a transmit is completed (TI0 is set in SCON0), or a data byte has been received (RI0 is set in SCON0). The UART0 interrupt flags are not cleared by hardware when the CPU vectors to the interrupt service routine. They must be cleared manually by software, allowing software to determine the cause of the UART0 interrupt (transmit complete or receive complete).



### Figure 27.1. UART0 Block Diagram

### 27.3. Multiprocessor Communications

9-Bit UART mode supports multiprocessor communication between a master processor and one or more slave processors by special use of the ninth data bit. When a master processor wants to transmit to one or more slaves, it first sends an address byte to select the target(s). An address byte differs from a data byte in that its ninth bit is logic 1; in a data byte, the ninth bit is always set to logic 0.

Setting the MCE0 bit (SCON0.5) of a slave processor configures its UART such that when a stop bit is received, the UART will generate an interrupt only if the ninth bit is logic 1 (RB80 = 1) signifying an address byte has been received. In the UART interrupt handler, software will compare the received address with the slave's own assigned 8-bit address. If the addresses match, the slave will clear its MCE0 bit to enable interrupts on the reception of the following data byte(s). Slaves that weren't addressed leave their MCE0 bits set and do not generate interrupts on the reception of the following data bytes, thereby ignoring the data. Once the entire message is received, the addressed slave resets its MCE0 bit to ignore all transmissions until it receives the next address byte.

Multiple addresses can be assigned to a single slave and/or a single address can be assigned to multiple slaves, thereby enabling "broadcast" transmissions to more than one slave simultaneously. The master processor can be configured to receive all transmissions or a protocol can be implemented such that the master/slave role is temporarily reversed to enable half-duplex transmission between the original master and slave(s).

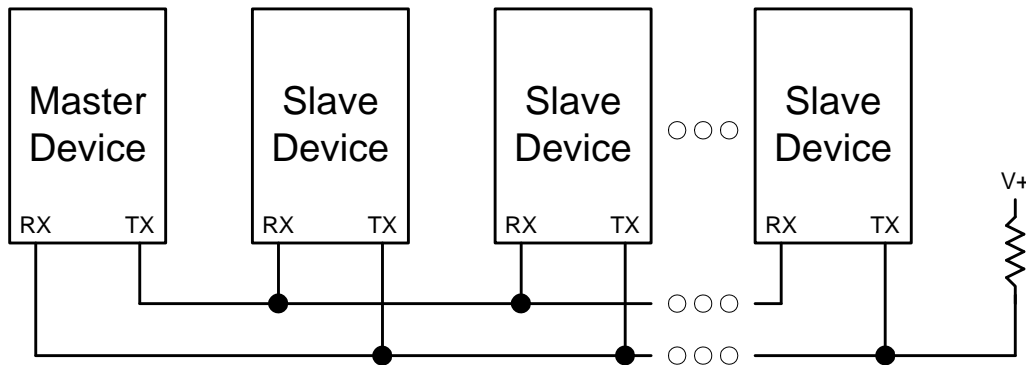
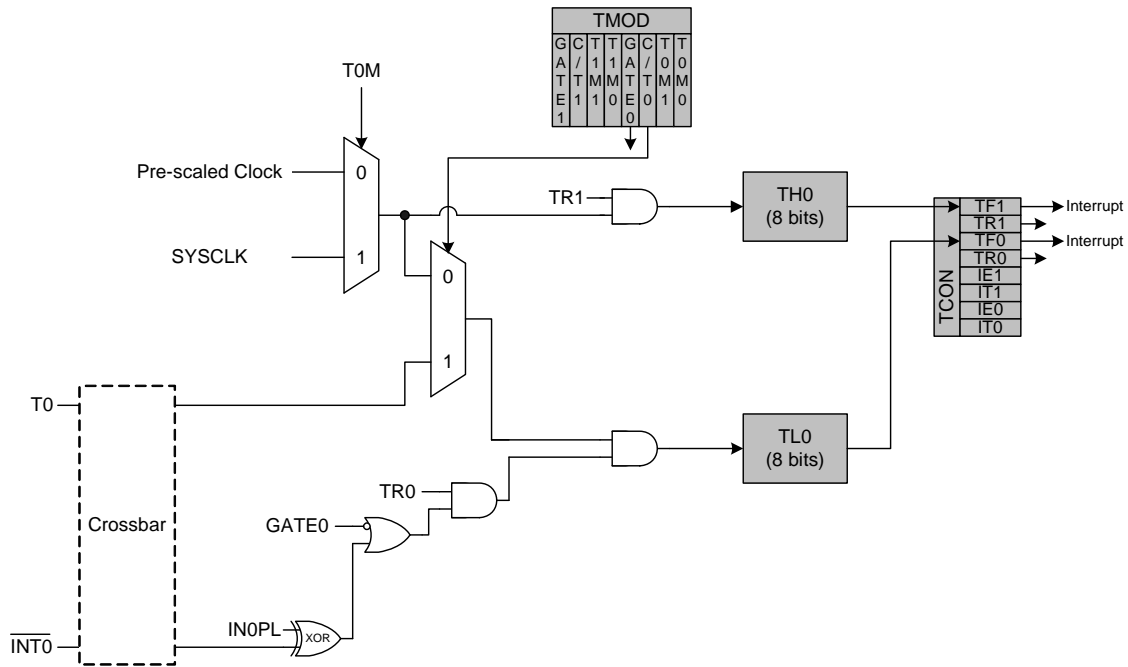


Figure 27.6. UART Multi-Processor Mode Interconnect Diagram



**Figure 28.3. T0 Mode 3 Block Diagram**

### 29.3.6. 16-Bit Pulse Width Modulator Mode

A PCA module may be operated in 16-Bit PWM mode. 16-bit PWM mode is independent of the other (8-bit through 15-bit) PWM modes. In this mode, the 16-bit capture/compare module defines the number of PCA clocks for the low time of the PWM signal. When the PCA counter matches the module contents, the output on CEXn is asserted high; when the 16-bit counter overflows, CEXn is asserted low. 16-Bit PWM Mode is enabled by setting the ECOMn, PWMn, and PWM16n bits in the PCA0CPMn register.

The duty cycle of the PWM output signal can be varied by writing to an “Auto-Reload” Register, which is dual-mapped into the PCA0CPHn and PCA0CPLn register locations. The auto-reload registers are accessed (read or written) when the bit ARSEL in PCA0PWM is set to 1. The capture/compare registers are accessed when ARSEL is set to 0. This synchronous update feature allows software to asynchronously write a new PWM high time, which will then take effect on the following PWM period.

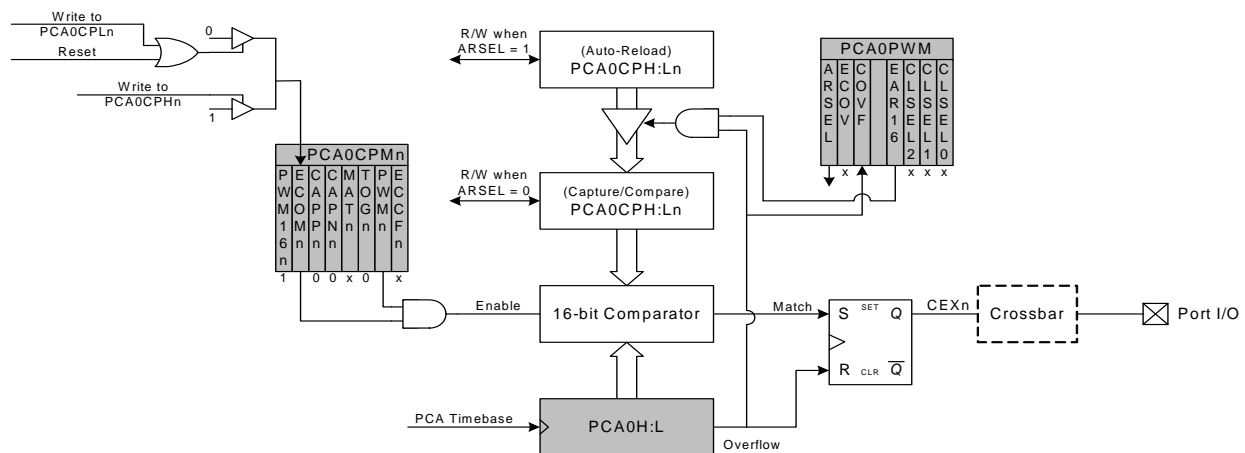
For backwards-compatibility with the 16-bit PWM mode available on other devices, the PWM duty cycle can also be changed without using the “Auto-Reload” register. To output a varying duty cycle without using the “Auto-Reload” register, new value writes should be synchronized with PCA CCFn match interrupts. Match interrupts should be enabled ( $ECCFn = 1$  AND  $MATn = 1$ ) to help synchronize the capture/compare register writes. If the MATn bit is set to 1, the CCFn flag for the module will be set each time a 16-bit comparator match (rising edge) occurs. The CF flag in PCA0CN can be used to detect the overflow (falling edge). The duty cycle for 16-Bit PWM Mode is given by Equation 29.4.

**Important Note About Capture/Compare Registers:** When writing a 16-bit value to the PCA0 Capture/Compare registers, the low byte should always be written first. Writing to PCA0CPLn clears the ECOMn bit to 0; writing to PCA0CPHn sets ECOMn to 1.

$$\text{Duty Cycle} = \frac{(65536 - PCA0CPn)}{65536}$$

### Equation 29.4. 16-Bit PWM Duty Cycle

Using Equation 29.4, the largest duty cycle is 100% (PCA0CPn = 0), and the smallest duty cycle is 0.0015% (PCA0CPn = 0xFFFF). A 0% duty cycle may be generated by clearing the ECOMn bit to 0.



### Figure 29.10. PCA 16-Bit PWM Mode

# C8051F80x-83x

## SFR Definition 29.1. PCA0CN: PCA0 Control

Bit	7	6	5	4	3	2	1	0
Name	CF	CR				CCF2	CCF1	CCF0
Type	R/W	R/W	R	R	R	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xD8; Bit-Addressable

Bit	Name	Function
7	CF	<b>PCA Counter/Timer Overflow Flag.</b> Set by hardware when the PCA Counter/Timer overflows from 0xFFFF to 0x0000. When the Counter/Timer Overflow (CF) interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.
6	CR	<b>PCA Counter/Timer Run Control.</b> This bit enables/disables the PCA Counter/Timer. 0: PCA Counter/Timer disabled. 1: PCA Counter/Timer enabled.
5:3	Unused	Read = 000b, Write = Don't care.
2	CCF2	<b>PCA Module 2 Capture/Compare Flag.</b> This bit is set by hardware when a match or capture occurs. When the CCF2 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.
1	CCF1	<b>PCA Module 1 Capture/Compare Flag.</b> This bit is set by hardware when a match or capture occurs. When the CCF1 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.
0	CCF0	<b>PCA Module 0 Capture/Compare Flag.</b> This bit is set by hardware when a match or capture occurs. When the CCF0 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

**SFR Definition 29.2. PCA0MD: PCA0 Mode**

Bit	7	6	5	4	3	2	1	0
Name	CIDL	WDTE	WDLCK		CPS2	CPS1	CPS0	ECF
Type	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Reset	0	1	0	0	0	0	0	0

SFR Address = 0xD9

Bit	Name	Function
7	CIDL	<b>PCA Counter/Timer Idle Control.</b> Specifies PCA behavior when CPU is in idle mode. 0: PCA continues to function normally while the system controller is in Idle mode. 1: PCA operation is suspended while the system controller is in idle mode.
6	WDTE	<b>Watchdog Timer Enable.</b> If this bit is set, PCA Module 2 is used as the watchdog timer. 0: Watchdog Timer disabled. 1: PCA Module 2 enabled as Watchdog Timer.
5	WDLCK	<b>Watchdog Timer Lock.</b> This bit locks/unlocks the Watchdog Timer Enable. When WDLCK is set, the Watchdog Timer may not be disabled until the next system reset. 0: Watchdog Timer Enable unlocked. 1: Watchdog Timer Enable locked.
4	Unused	Read = 0b, Write = Don't care.
3:1	CPS[2:0]	<b>PCA Counter/Timer Pulse Select.</b> These bits select the timebase source for the PCA counter 000: System clock divided by 12 001: System clock divided by 4 010: Timer 0 overflow 011: High-to-low transitions on ECI (max rate = system clock divided by 4) 100: System clock 101: External clock divided by 8 (synchronized with the system clock) 11x: Reserved
0	ECF	<b>PCA Counter/Timer Overflow Interrupt Enable.</b> This bit sets the masking of the PCA Counter/Timer Overflow (CF) interrupt. 0: Disable the CF interrupt. 1: Enable a PCA Counter/Timer Overflow interrupt request when CF (PCA0CN.7) is set.

**Note:** When the WDTE bit is set to 1, the other bits in the PCA0MD register cannot be modified. To change the contents of the PCA0MD register, the Watchdog Timer must first be disabled.