CML Microcircuits

COMMUNICATION SEMICONDUCTORS

AN / Telecom / CMX850 /2 October 2003

Application Note

CMX850 - 8051 Architectural Overview, Programmer's Guide and Hardware Description

TABLE OF CONTENTS

CMX850 Introduction and Overview 2
Memory Organisation 2
Dual Data Pointers 5
XRAM Access and the MEMCON register 6
Burst Mode Memory Access 7
MUXAD 7
Local Boot ROM 10
Special Function Registers 12
I/O Ports 14
Power Map 16
Timers & Counters 17
Real Time Clock (RTC) and Alarm Registers 17
Watch-Dog Timers (WDT) 19
Serial Port Interface 20
Setting the Serial Port Mode 20
Setting the Serial Port Baud Rate 22
Writing to the Serial Port 24
Reading the Serial Port 25
RS232 Connection to a PC 25
CAS/FSK Detector Block 26
Interrupts 27
Interrupt Sources and Vector Addresses 27
Enabling & Disabling Interrupts 29
Interrupt Priorities 29
Interrupt Handling 29
Interrupt Activation Levels 30
Instructions & Addressing 31
CMX850 Instruction Set 31
Addressing Modes 46
Direct Addressing 46
Indirect Addressing 47
Register Addressing 47
Immediate Addressing 47
Indexed Addressing 47
Conclusion 47
Appendix 48
References 50

CMX850 Introduction and Overview

The CMX850 combines an enhanced 8051 Microcontroller with a feature-rich V.22bis (2400bps) modem to create the platform for a powerful and flexible telecom processor. In addition to its many 8051-Microcontroller enhancements, the CMX850 supports all of the standard 8051 features and instructions as well. This application note provides supplemental information regarding CMX850 operation and programming and should be used in conjunction with the CMX850 data sheet.

The following table provides detail on the differences between the standard 8051 Microcontroller and the enhanced 8051 Microcontroller contained in the CMX850:

Feature	Standard 8051	CMX850 8051
On-chip data memory	128 bytes	256 bytes
On-chip data XRAM	0	8k bytes
On-chip program memory	>4k bytes	0
Address space for external RAM	64k bytes	64k bytes
Address space for external ROM	64k bytes (Can be increased with additional components)	64k bytes, (Can be increased beyond 4Mbytes using minimal additional components)
Non-multiplexed memory interface	No	Yes
General Purpose I/O Lines	32	35
I/O Ports	4	6
Port direction control registers to minimize power consumption?	No	Yes
Interrupt Sources	5	13
“Super Priority” interrupt pin?	No	Yes
Data Pointers	1	2
Continuous program memory reads?	Yes	No (for reduced power consumption)
Time stretching of external memory fetches?	No	Yes
Number of oscillator cycles per machine cycle	12	12
Number of timer/counters	2	2 (plus Real Time Clock & Watchdog timer)
Keyboard decoder?	No	Yes
Number of ADC	0	2
Number of PWM	0	2
Real-time clock & alarm?	No	Yes
Advanced oscillator & powersave controls?	No	Yes (multiple schemes)
Watchdog timer	No	Yes

Table 1: Comparison of CMX850 8051 and Standard 8051 Micro Controllers

Memory Organisation

The CMX850 memory structure is identical to the standard 8051 memory structure. There is internal memory space for short-term data, variables, vectors and routines, while external memory is used for program code and additional Data. Unlike many other memory structures, the 8051 is not flat and continuous; the internal memory space between $7F and $FF contains two 128 byte addressable memory locations in parallel (IDATA RAM and the SFR) that can be addressed separately.

Referring to Figure 1, there are a total of 256 bytes of internal rewriteable scratchpad memory (RAM). The lower 128 bytes can be addressed directly and indirectly and is common with the 8051 format. This lower section of RAM includes four banks of memory (eight registers per bank, addresses $00 through $1F) that can be used as register space enabling rapid exchange of interrupt data. This could typically be used in multitasking software where the quick set-up of routine data is essential.

Address space $20 to $2F is bit addressable making it particularly useful for flag settings etc. Details of the specific 8051 bit operations can be found later in this document.

The upper 128 bytes of internal RAM can only be used in indirect addressing mode. This limited addressing method means that another 128 bytes of address space can be created which can be directly addressed only, and it is in these 128 bytes that the SFRs (Special Function Registers) reside. More details on the SFRs are provided later in this document.

Figure 2 shows how the remaining memory is structured; the additional 8kB of XRAM is discussed later is this section. The three external CSN pins, labelled CSN1, CSN2 and CSN3, allow for the direct addressing of three 64kbyte memory blocks without the need for external de-multiplexing of the data and address lines.

CSN1 is by default the base address for all user programs and where program reset vectors and interrupts normally reside. Details of how to map 8kB of the external memory to the XRAM are discussed later in this chapter.

Using a combination of standard port pins it is also possible to enlarge the memory further by addressing additional sections of memory. This is called “page switching” or banking.

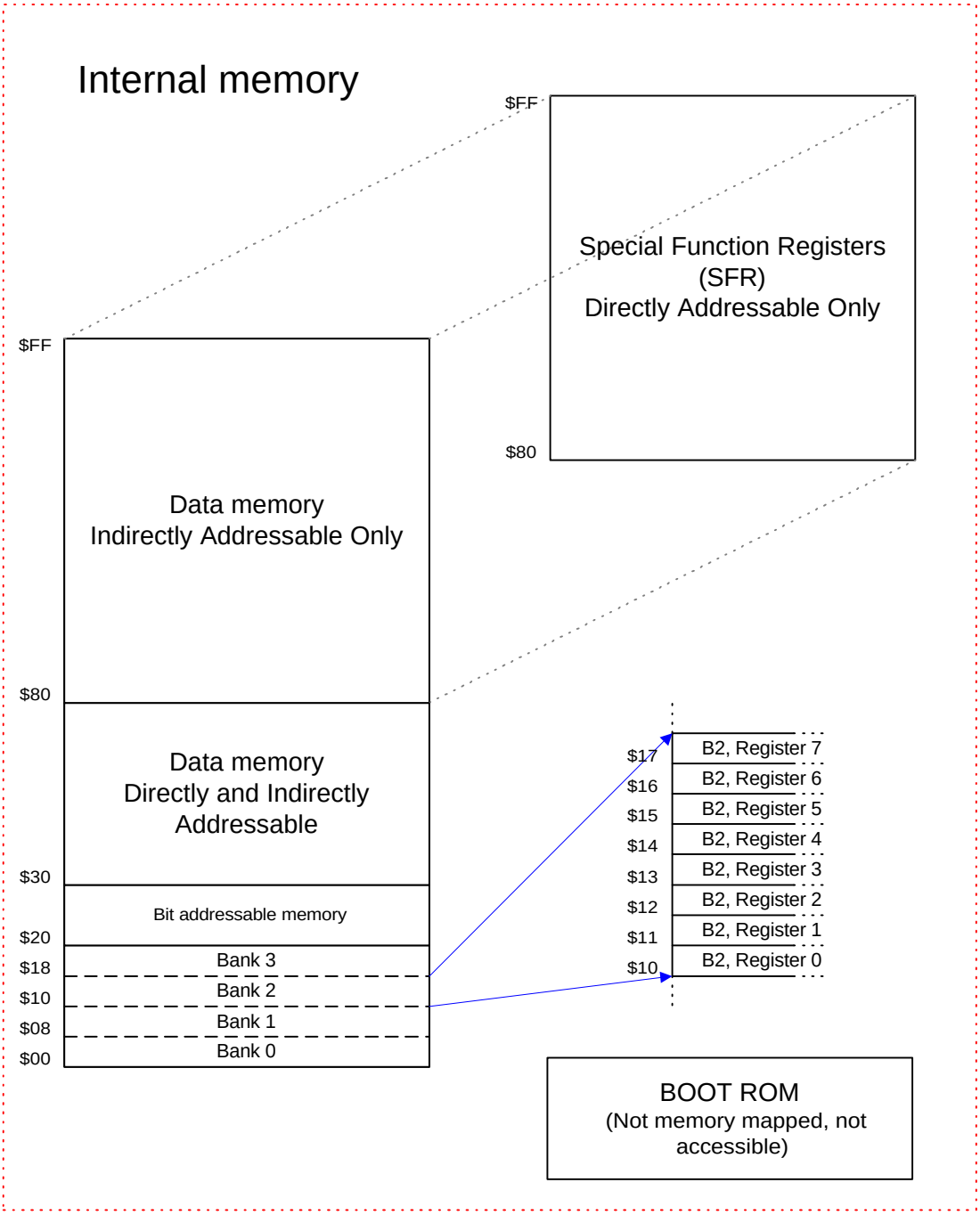


Figure 1: CMX850 Internal Memory Map

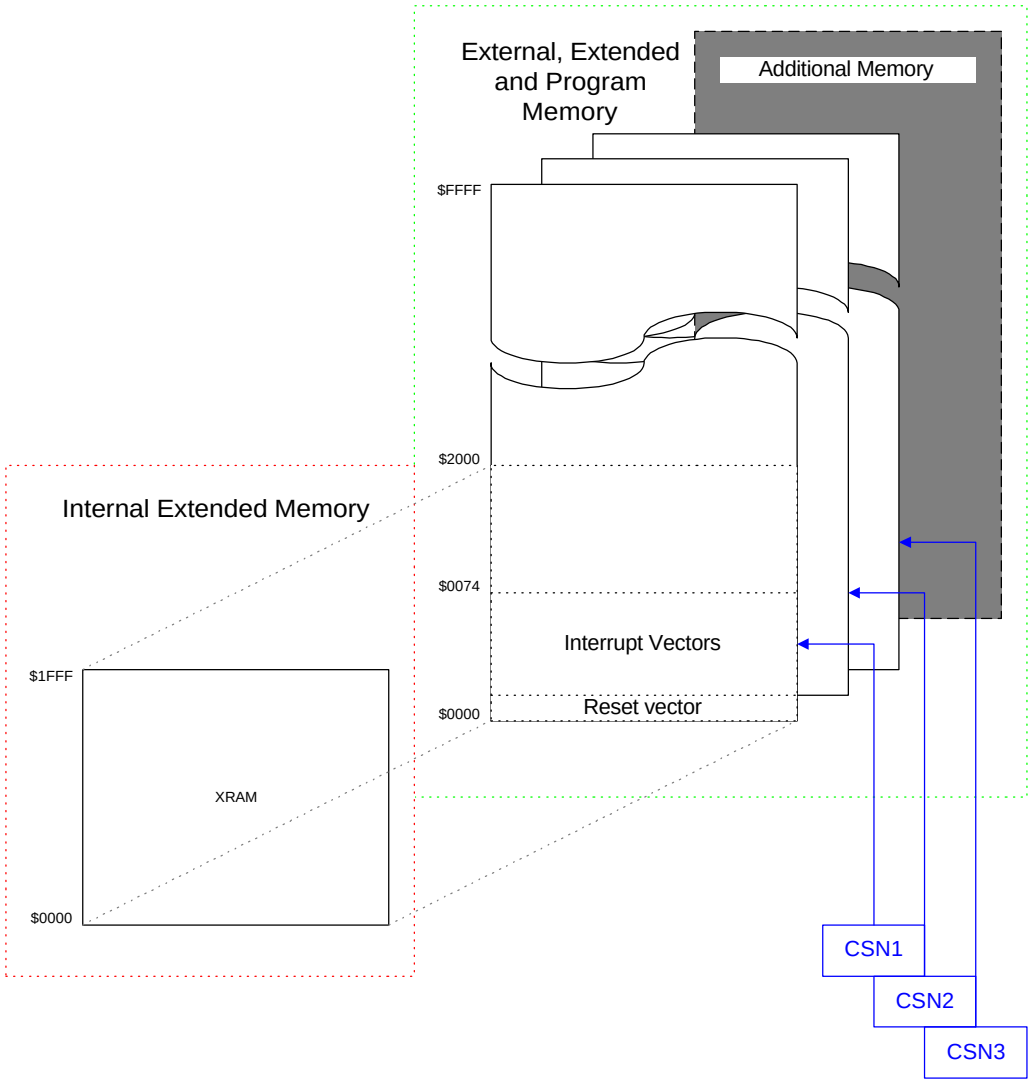


Figure 2: Additional Memory Map

Dual Data Pointers

The original 8051 core had one data pointer but the CMX850 has two, and this makes transfers of data blocks throughout the extended memory space quicker and easier. The CMX850 data pointers are each mapped into a high and low register within the SFR space; DPL and DPH ($82, $83) for the first data pointer and DPL1 and DPH1 ($84, $85) for the second data pointer. The currently selected data pointer is determined by bit 0 of the DPS SFR ($86).

The software routine below demonstrates one method of moving a block of data in external memory.

```
; SH and SL are high and low byte source address.
; DH and DL are high and low byte of destination address.
; DPS is the data pointer select. Reset condition is DPS=0, DPTR0 is selected.

DPS EQU $86                ;Acknowledge data pointer select
MOV R5, #64                ;#64 bytes to move in total
MOV DPTR, #DHDL            ;Load destination address into DPTR0
INC DPS                    ;Change active data pointer
```

```
MOV DPTR, #SHSL          ;Load source address onto DPTR1

; The following code loops X number of times, X being the value to be found in register 5

MOVE:
MOVX A, @DPTR            ;Read source data byte from DPTR1
INC DPTR                 ;Increment to the next destination address
INC DPS                  ;Change data pointer to the destination at DPTR0
MOVX @DPTR, A            ;Write data to destination
INC DPTR                 ;Increment to the next source address
INC DPS                  ;Change data pointer to source
DJNZ R5, MOVE            ;All data moved?
INC DPS                  ;Leave DPTR0 selected
```

XRAM Access and the MEMCON register

Extended internal memory (over and above the scratchpad memory) is available in the form of 8kB of XRAM. Normal program memory is situated externally from the CMX850 where the memory will typically be in the form of ROM, FLASH or EEPROM. However, the bottom 8kB of external program memory can be mapped into the internal XRAM if required.

The MEMCON register (SFR $FA) allows the MOVX instruction to point either to external memory or the internal XRAM. Please note that extra care must be taken when the destination of the MOVX command is altered while the program is being executed from XDATA.

<table>
<tr><td>Bit Position</td><td colspan="4">Bit Setting</td></tr>
<tr><td></td><td>1</td><td>0</td><td></td><td></td></tr>
<tr><td>7</td><td>Enable data pin bus-holding devices</td><td>Disable data pin bus-holding devices</td><td></td><td></td></tr>
<tr><td>6</td><td>On-chip XRAM mapped into bottom 8Kbytes of program memory</td><td>Program memory is entirely off-chip</td><td></td><td></td></tr>
<tr><td>5</td><td>MOVX write operations are stretched by one machine cycle (12 xtal cycles)</td><td>MOVX write operations are not stretched</td><td></td><td></td></tr>
<tr><td>4</td><td rowspan="2">00 - MOVX write destination is on-chip XRAM</td><td rowspan="2">01 - 0 1 MOVX write destination is off-chip, using pin CSN1</td><td rowspan="2">10 - MOVX write destination is off-chip, using pin CSN2</td><td rowspan="2">11 - MOVX write destination is off-chip, using pin CSN3</td></tr>
<tr><td>3</td></tr>
<tr><td>2</td><td>MOVX read operations are stretched by one machine cycle (12 xtal cycles)</td><td>MOVX read operations are not stretched</td><td></td><td></td></tr>
<tr><td>1</td><td rowspan="2">00 - MOVX read source is on-chip XRAM</td><td rowspan="2">01 - MOVX read source is off-chip, using pin CSN1</td><td rowspan="2">10 - MOVX read source is off-chip, using pin CSN2</td><td rowspan="2">11- MOVX read source is off-chip, using pin CSN3</td></tr>
<tr><td>0</td></tr>
</table>

Table 2: MEMCON Bit Descriptions

The ability to be able to run small pieces of code internally makes it possible for the CMX850 to dynamically update its external program memory if it is located in FLASH memory. A typical scenario would be to serially download a new program via the CMX850 under the

control of a “Thin Stub” routine into the XRAM which would then direct a further download to the external FLASH memory. This set-up makes board designs a lot simpler and adds convenience to systems where code can be upgraded in the field.

NOTE: The term “thin stub” refers to a section of software that is attached to the main code to perform a simple download function. In the case of the CMX850, a thin stub resides in the external program memory and allows new program code to be downloaded into FLASH memory. If the thin stub were not present then the FLASH memory would have to be removed and programmed in an external programmer.

Note: The thin stub is described later in this section.

It is also possible to increase the number of cycles required to perform a MOVX instruction. This is a very useful feature where if, for example, external memory access is slower than the current instruction cycle time. Bits 2 and 5 of the MEMCON Register are available to perform this stretch of instruction time, making it possible to update memory access in “real time”.

MEMCON Bit 7 makes it possible to reduce power consumption when the data or address bus is idling over long periods of time, or if the CMX850 is in a powered down state. It works by holding onto the bus very weakly, when used with an inactive bus, the bus follows the peripheral pins idle state. E.g. if the peripheral pin defaults to Vdd the CMX850 pin will follow.

Burst Mode Memory Access

One of the CMX850 power saving features is a reduced clocking speed combined with a “burst mode” memory access that allows external memory accesses to be kept to the absolute minimum. When used in concert with the CMX850’s advanced power management features, the burst mode memory access can help reduce power consumption considerably.

It achieves this because certain memory devices only power up while they are being accessed from the micro controller. By burst writing or reading the data to and from the memory device, “full power” is only required for the time the bus is active, the remaining time the memory device can be powered down.

The effect of this type of arrangement is to reduce the average power consumed in each cycle. This method makes much better use of the available power than a slow read or write access, where the memory is active for much longer.

MUXAD

The MUXAD pin makes it possible to multiplex both data and address lines onto the same bus, thus making it possible to use I/O pins for multiple tasks. The illustration below shows what pins are freed and what additional components (address latch) are required when using the MUXAD pin.

Note: When MUXAD is disabled the port addresses in the SFR (P0.0..0.2 & P5.5..5.7), become a bit addressable memory location. They can be treated as extra register space. The $\overline{\text{INT9}}$ super priority interrupt input and upper three columns of the keyboard decoder are also not available.

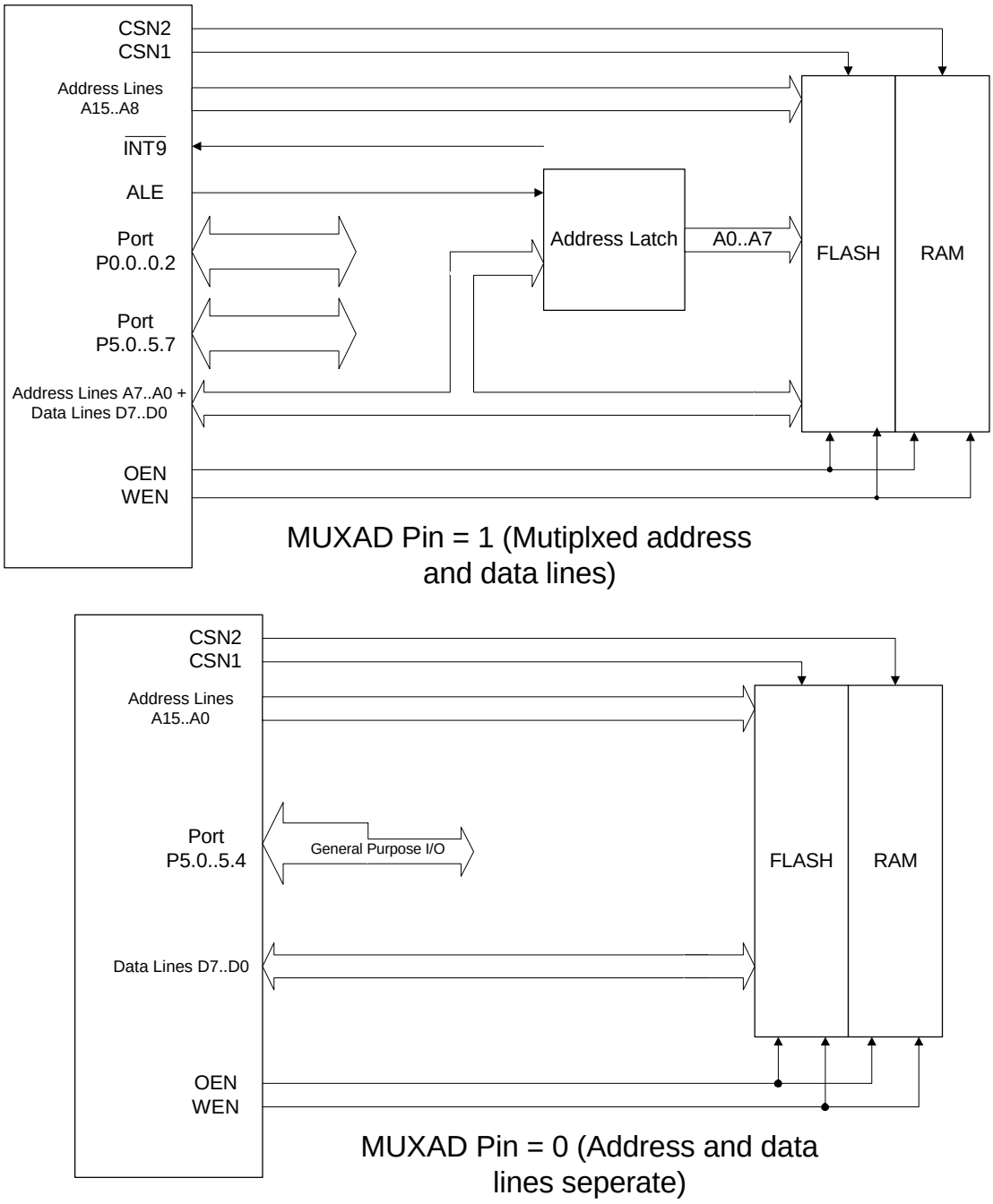
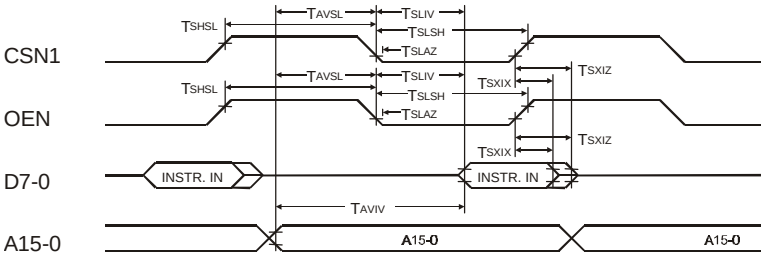


Figure 3: Effect of MUXAD on External Memory Accesses

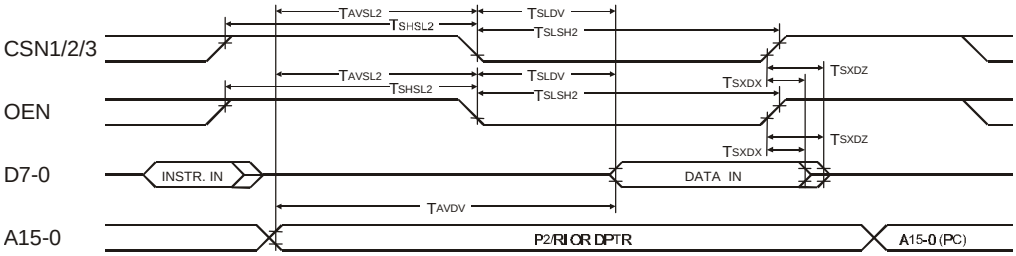
NOTE: Port 0 pins do not have memory address/data driven onto them during an external memory access, as would happen on a standard 8051 Microcontroller. The CMX850 external memory interface is completely separate from the Port 0 circuit. With the MUXAD pin set to 0, only port pins 5.0 to 5.4 remain available for use. P2 is retained internally and is placed into A8..15 of the bus during MOVX instructions using @Ri addressing.

With the MUXAD pin asserted, access to data is time multiplexed with the lower 8 address lines. See illustration below for timing details. The program still address the data as normal, but now Port 5 is entirely available for I/O and bits D0..2 of port 0 have been freed up for other purposes.

EXTERNAL PROGRAM MEMORY READ CYCLE



EXTERNAL DATA READ CYCLE



EXTERNAL DATA WRITE CYCLE

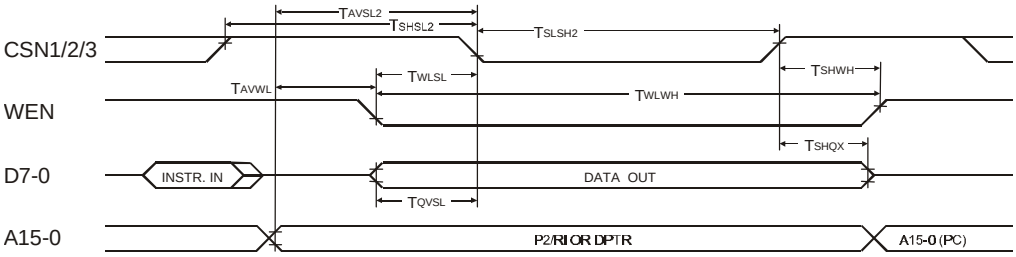
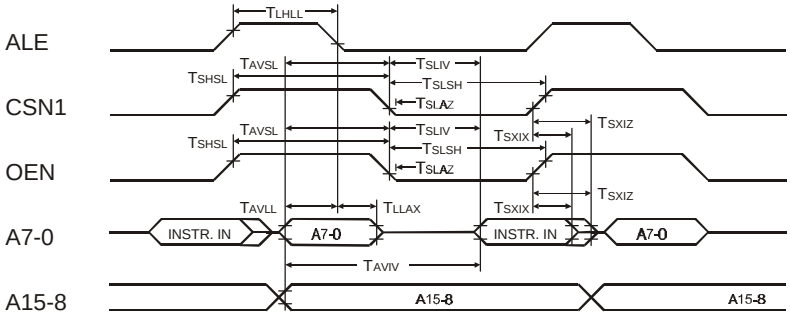
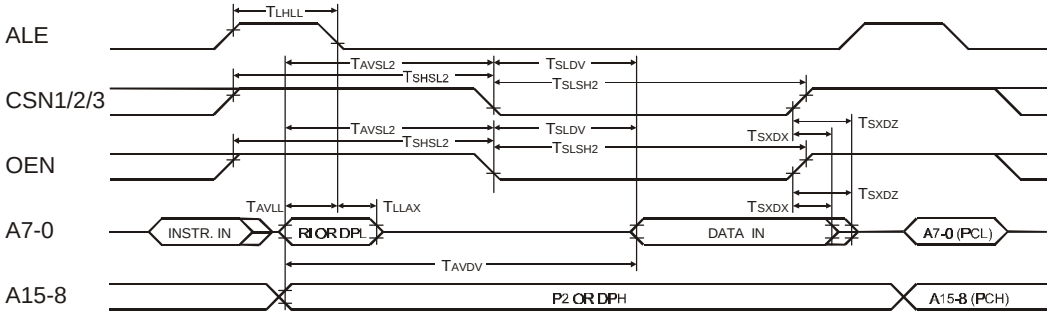


Figure 4: Non-Multiplexed Memory Interface Timing Diagrams

EXTERNAL PROGRAM MEMORY READ CYCLE



EXTERNAL DATA READ CYCLE



EXTERNAL DATA WRITE CYCLE

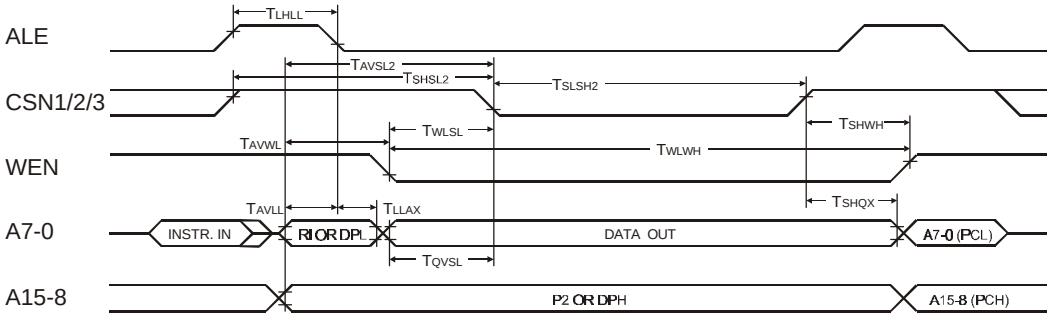


Figure 5: Multiplexed Memory Interface Timing Diagrams

Local Boot ROM

It is common practice to include functions that allow for memory downloads while the micro controller remains soldered to the application board. The wide spread use of low cost FLASH memory has made this not only possible but desirable. In Circuit Programming is supported in the CMX850 by the use of hardwired "Thin Stub" located on chip.

The Thin Stub is used here to describe a small section of code that resides in ROM within the CMX850. Its function is to set-up the CMX850 serial port following a reset ready for the download of a Fat Stub software routine. The Fat Stub, which is another software routine, is downloaded from the host programmer. Following this download the Thick Stub is situated in the 8K of internal XRAM and organises the FLASH download.

Thin Stub

Details of the Boot ROM power up sequence can be found in the CMX850 data sheet (from revision D/850/5), for reference however it is as follows:

Power Up CMX850
RESETN Pin = AVSS
VBIAS Pin = short to AVDD
2.5 Sec Delay
RESETN Pin = AVDD
1uS Delay
VBIAS Pin = remove short
The CMX850 is now in Boot Rom mode and the serial port is ready to accept the Thick Stub (19K2, 8 bits no parity, 1 stop bit)

Thick Stub

The Thick Stub is subsequently downloaded via the serial port automatically into the 8kB XRAM, from where it takes control of the serial port, external memory control and addressing of the FLASH.

The Thick Stub must ensure that binary data is only loaded when ready (by controlling the RS232 handshake lines via software allocated port pins or by using a Xon/Xoff software routine if only the RXD and TXD pins are to be used. It may also flag an error if there is a FLASH boundary overrun, incorrect FLASH ID code, non-functional FLASH etc.

The Thick Stub will also need to control memory access of the FLASH memory as well as be able to control the writes to the command register according to the FLASH requirements.

Note: Example Thick Stub software is provided with the EV8500 Evaluation Kit.

Special Function Registers

The special function registers (SFR) are used with the 8051 architecture as a means of adding extra functionality to a Microcontroller without making any radical software changes. The SFRs are accessible via direct addressing of the upper 128 bytes of internal RAM ($80 to $FF).

Within the CMX850 the SFR's are organized as follows:

(lsb)	'..... 000'	'..... 001'	'..... 010'	'..... 011'	'..... 100'	'..... 101'	'..... 110'	'..... 111'
$F8			MEMCON	RTCCON	TIME0	TIME1	TIME2	TIME3
$F0	(B)		WDTCON	WDTLD	ALM0	ALM1	ALM2	ALM3
$E8		CASDET	CBUSCON	CBUSBUF	KBCON	KBSTAT	KBBUF	
$E0	(ACC)		ADCCON1	ADCCON2	ADCBUFL	ADCBUFH	ADCTHRL	ADCTHRH
$D8	P5	P5DIR	P5OD	P5RES		PWMCON	PWM1	PWM2
$D0	(PSW)	FSKBUF						
$C8								
$C0	P4	P4DIR	P4OD	P4RES				
$B8	(IP)	IP_1						
$B0	(P3)	P3DIR	P3OD	P3RES				
$A8	(IE)	IE_1	ICON1A	ICON1B				
$A0	(P2)							
$98	(SCON)	(SBUF)			OSCCON	SPDCON	SPXMASK	PDXMASK
$90	(P1)	P1DIR	P1OD	P1RES	P0DIR			
$88	(TCON)	(TMOD)	(TL0)	(TL1)	(TH0)	(TH1)		
$80	(P0)	(SP)	(DPL)	(DPH)	DPL1	DPH1	DPS	(PCON)
	Bit Addressable	Standard 80C51 SFRs are shown in parentheses						

Table 3: CMX850 SFR Map

The SFRs within parentheses are the basic 8051 SFRs and will be found on all 8051 based cores; the remaining registers are CMX850 specific. Additionally each register is further broken down in the following bit addressed table.

	Memory Address (Bit addressable)		Abbreviation	Register	Data Sheet Section
	Hex	Binary			
	FA	11111010	MEMCON	Memory Control Register	1.5.4.1
	FB	11111011	RTCCON	Real Time Clock Control Register	1.5.12.1
	FC	11111100	TIME0	Real Time Clock Time Register 0	1.5.12.2
	FD	11111101	TIME1	Real Time Clock Time Register 1	1.5.12.2
	FE	11111110	TIME2	Real Time Clock Time Register 2	1.5.12.2
	FF	11111111	TIME3	Real Time Clock Time Register 3	1.5.12.2
*	F0	11110000	(B)	B Register	1.5.14.1
	F2	11110010	WDTCON	Watchdog Control Register	1.5.11.1
	F3	11110011	WDTLD	Watchdog Load Register	1.5.11.2
	F4	11110100	ALM0	Real Time Clock Alarm Register 0	1.5.12.3
	F5	11110101	ALM1	Real Time Clock Alarm Register 1	1.5.12.3
	F6	11110110	ALM2	Real Time Clock Alarm Register 2	1.5.12.3
	F7	11110111	ALM3	Real Time Clock Alarm Register 3	1.5.12.3
	E9	11101001	CASDET	CAS Detect Control Register	1.5.13.1
	EA	11101010	CBUSCON	C-Bus Control Register	1.5.9.1
	EB	11101011	CBUSBUF	C-Bus Buffer Register	1.5.9.2

	EC	11101100	KBCON	Keyboard Control Register	1.5.10.1
	ED	11101101	KBSTAT	Keyboard Status Register	1.5.10.2
	EE	11101110	KBBUF	Keyboard Buffer Register	1.5.10.3
*	E0	11100000	(ACC)	Accumulator	1.5.14.1
	E2	11100010	ADCCON1	ADC 1 Control Register	1.5.8.1
	E3	11100011	ADCCON2	ADC 2 Control Register	1.5.8.1
	E4	11100100	ADCBUFL	ADC Low Buffer Register	1.5.8.2
	E5	11100101	ADCBUFH	ADC High Buffer Register	1.5.8.2
	E6	11100110	ADCTHRL	ADC Low Threshold Register	1.5.8.3
	E7	11100111	ADCTHRH	ADC High Threshold Register	1.5.8.3
*	D8	11011000	P5	Port 5	1.5.3.1
	D9	11011001	P5DIR	Port 5 Direction Register	1.5.3.2
	DA	11011010	P5OD	Port 5 Open-Drain Register	1.5.3.3
	DB	11011011	P5RES	Port 5 Resistor Pull-Up Register	1.5.3.4
	DD	11011101	PWMCON	PWM Control Register	1.5.7.1
	DE	11011110	PWM1	PWM 1 Data Register	1.5.7.2
	DF	11011111	PWM2	PWM 2 Data Register	1.5.7.2
*	D0	11010000	(PSW)	Program Status Word Register	1.5.14.2
	D1	11010001	FSKBUF	FSK Data Buffer	1.5.13.2
*	C0	11000000	P4	Port 4	1.5.3.1
	C1	11000001	P4DIR	Port 4 Direction Register	1.5.3.2
	C2	11000010	P4OD	Port 4 Open-Drain Register	1.5.3.3
	C3	11000011	P4RES	Port 4 Resistor Pull-Up Register	1.5.3.4
*	B8	10111000	(IP)	Interrupt Priority Control Register	1.5.5.2
	B9	10111001	IP_1	Interrupt Priority Control 1 Register	1.5.5.2
*	B0	10110000	(P3)	Port 3	1.5.3.1
	B1	10110001	P3DIR	Port 3 Direction Register	1.5.3.2
	B2	10110010	P3OD	Port 3 Open-Drain Register	1.5.3.3
	B3	10110011	P3RES	Port 4 Resistor Pull-Up Register	1.5.3.4
*	A8	10101000	(IE)	Interrupt Enable Control Register	1.5.5.1
	A9	10101001	IE_1	Interrupt Enable 1 Register	1.5.5.1
	AA	10101010	ICON1A	Interrupt Control register A	1.5.5.3
	AB	10101011	ICON1B	Interrupt Control register B	1.5.5.3
*	A0	10100000	(P2)	Port 2	1.5.3.1
*	98	10011000	(SCON)	Serial Control Register	1.5.15.4
	99	10011001	(SBUF)	Serial Data Buffer Register	1.5.15.5
	9C	10011100	OSCCON	Oscillator Control Register	1.5.6.1
	9D	10011101	SPDCON	Speed Control Register	1.5.6.2
	9E	10011110	SPXMASK	System Crystal Frequency Select	1.5.6.3
	9F	10011111	PDXMASK	Power Down Exit Register	1.5.6.4
*	90	10010000	(P1)	Port 1	1.5.3.1
	91	10010001	P1DIR	Port 1 Direction Register	1.5.3.2
	92	10010010	P1OD	Port 1 Open-Drain Register	1.5.3.3
	93	10010011	P1RES	Port 1 Resistor Pull-Up Register	1.5.3.4
	94	10010100	P0DIR	Port 0 Direction Register	1.5.3.1
*	88	10001000	(TCON)	Timer/Counter Control Register	1.5.15.1
	89	10001001	(TMOD)	Timer/Counter Mode Control Register	1.5.15.2

	8A	10001010	(TL0)	Timer/Counter O Low Byte Register	1.5.15.3
	8B	10001011	(TL1)	Timer/Counter 1 Low Byte Register	1.5.15.3
	8C	10001100	(TH0)	Timer/Counter O High Byte Register	1.5.15.3
	8D	10001101	(TH1)	Timer/Counter 1 High Byte Register	1.5.15.3
*	80	10000000	(P0)	Port 0	1.5.3.1
	81	10000001	(SP)	Stack Pointer	1.5.14.3
	82	10000010	(DPL)	Data Pointer Low Byte	1.5.2.1
	83	10000011	(DPH)	Data Pointer High Byte	1.5.2.1
	84	10000100	DPL1	CMX850 Data Pointer Low Byte	1.5.2.1
	85	10000101	DPH1	CMX850 Data Pointer High Byte	1.5.2.1
	86	10000110	DPS	Data Pointer Select Register	1.5.2.2
	87	10000111	(PCON)	Power Control Register	1.5.6.5

* Bit Addressable

Table 4: SFR Descriptions and Bit Addressability

Bit addressing (the ability to write to or alter individual bits) is very useful when setting up SFRs. An SFR whose address ends with an 8 or a 0 are bit addressable, as indicated above.

I/O Ports

The CMX850 has a total of 5 active I/O ports available for interfacing with external circuitry. Nearly all the bits available through these ports have multiple functions. Ports 0, 1 & 3 are standard ports and are used in all 8051 micro controllers. Port 2 is not fully implemented but is retained for backward compatibility with the MOVX A,@Rn instruction and XDATA access. The CMX850 provides two additional ports, ports 4 and 5, to increase flexibility. Nearly every port has multiple registers for precise control and power consumption optimisation.
All port registers can be accessed via the SFR space with locations as follows:

(lsb)	'..... 000'	'..... 001'	'..... 010'	'..... 011'	'..... 100'
$F8					
$F0					
$E8					
$E0					
$D8	P5	P5DIR	P5OD	P5RES	
$D0					
$C8					
$C0	P4	P4DIR	P4OD	P4RES	
$B8					
$B0	(P3)	P3DIR	P3OD	P3RES	
$A8					
$A0	(P2)				
$98					
$90	(P1)	P1DIR	P1OD	P1RES	P0DIR
$88					
$80	(P0)				

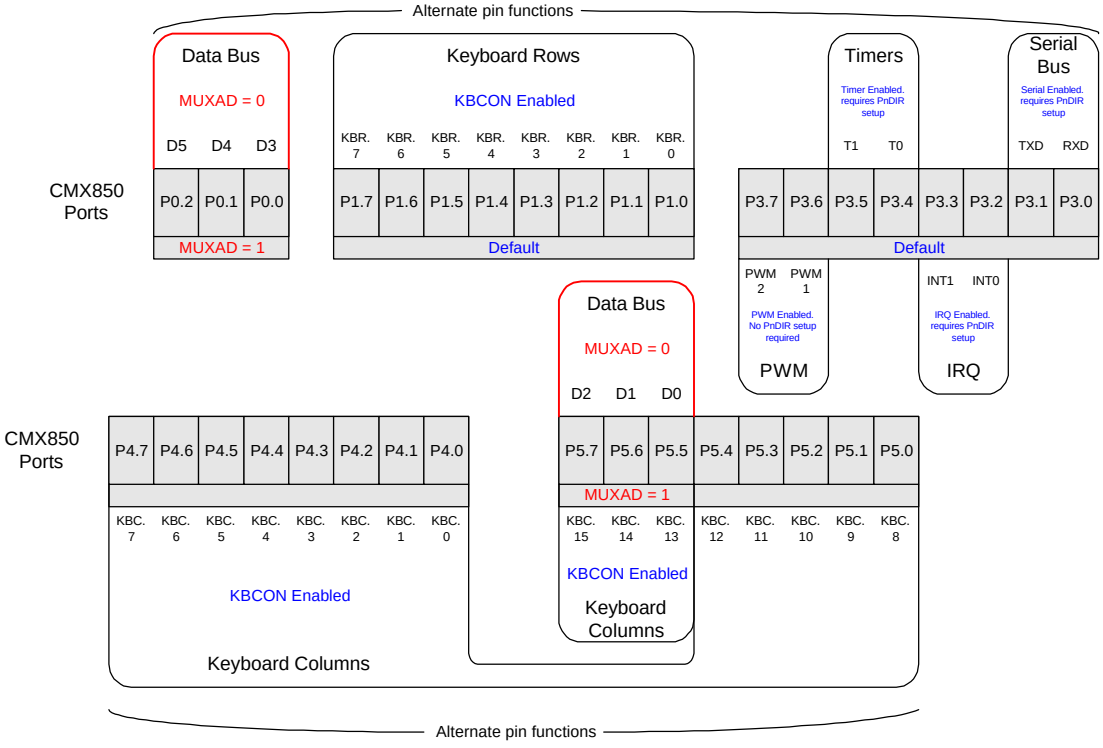
Table 5: Port Register Locations

As with all registers within the first column of the SFR, the port registers are all bit addressable. The register functions are as follows (for the symbol column, “n” refers to the port number):

SFR Register	Symbol	Function	Bit Assignment	Comment
Port Data Register	Pn	Data registers associated with the port output register. Read-Modify-Write operation capable		Bit Addressable
Port Direction Register	PnDIR	Each port bit can be selected as an input or output.	Input = 0 Output = 1	
Port Open-Drain register	PnOD	Each port bit can be configured with a pull-up/pull-down driver or open drain driver	Pull up/pull down driver = 0 Open drain driver = 1	Open drain refers to the characteristic of never sourcing current. That for a logic 1 the pin is high impedance
Port Resistor Pull-up Register	PnRES	Each port bit can be configured with or without a pull-up resistor	No resistor = 0 50kohm resistor to DVDD = 1	Current drain will increase if the 50kohm resistor is enabled
Note: A pull up resistor and pull-up/pull-down driver should not be selected simultaneously as this will increase the current consumption.				

Table 6: Port Register Descriptions

A number of the CMX850 port pins have multiple uses to allow operations such as accessing an external keyboard. A break down of the alternative pin allocations is as follows:



Note. MUXAD control overrides all other register setups.

Figure 6: CMX850 Pin Alternate Functions

Some instructions that read a port read the latch and others read the pin. The instructions that read the latch rather than the pin are the instructions that read a value change it and then rewrite it to the latch. These are called “read-modify-write” instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

ANL	Logical AND, e.g. ANL PI, A
ORL	Logical OR, e.g., ORL P2, A
XRL	Logical EX-OR, e.g. XRL P3, A
JBC	Jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL
CPL	Complement bit, e.g., CPL P3.0
INC	Increment, e.g., INC P2
DEC	Decrement, e.g., DEC P2
DJNZ	Decrernent and jump if not zero, e.g., DJNZ P3, LABEL
MOV PX. Y, C	Move carry bit to bit Y of Port X
CLR PX.Y	Clear bit Y of Port X
SETB PX.Y	Set bit Y of Port X

Table 7: read-modify-write

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, and then write the new byte back to the latch. The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a zero. Reading the latch rather than the pin will return the correct value of 1.

Power Map

The inherent flexibility of the CMX850 extends also to the power control settings. Many of the functional blocks within the CMX850 can be enabled or disabled and as such makes it possible to develop very flexible power saving schemes. Figure 7 shows what functional blocks exist that can be used to reduce overall power consumption. The illustration can also be used to calculated typical power consumption figures when used in conjunction with the CMX850 data sheet (see section 1.7.1.3 for details on specific current consumption figures.

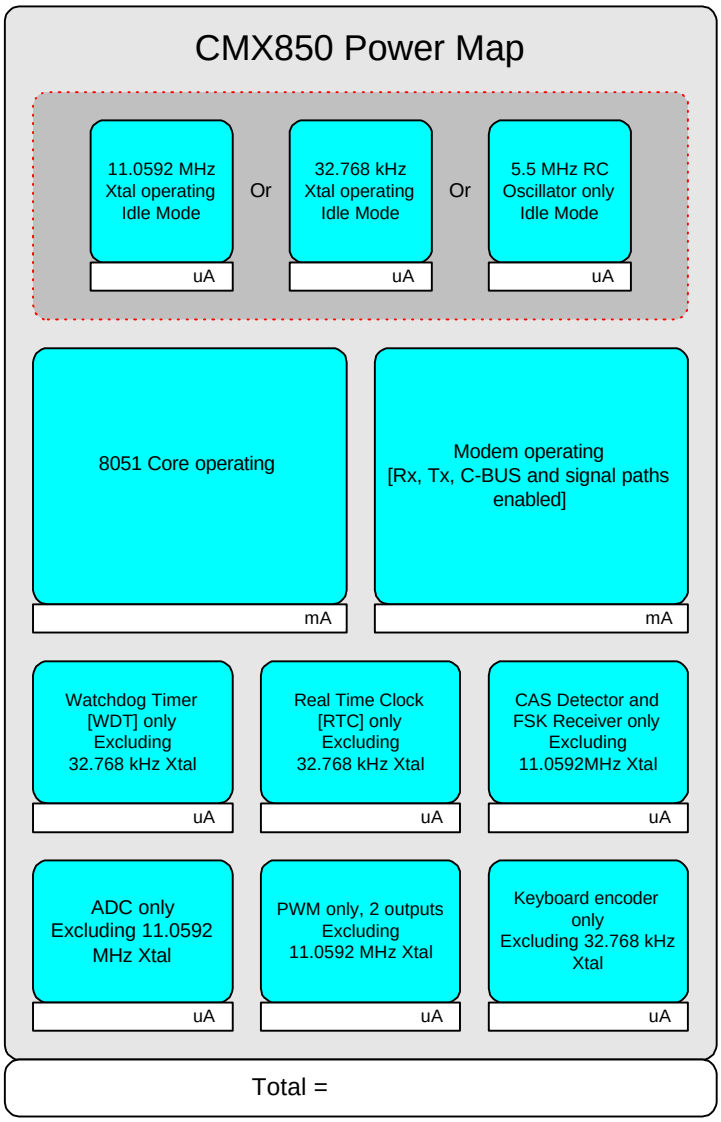


Figure 7: CMX850 Pin Alternate Functions

Other power saving schemes and methods are also possible within the CMX850 such as Burst Mode Memory Access, reduced operating voltage and selection of external analogue components. Please refer to the appropriate sections within the CMX850 data sheet or this application note for further details.

Timers and Counters

The CMX850 has a built in Real Time Clock (RTC) and Watchdog Timer (WDT). This section aims to give the user some insight of RTC and WDT system control to operate the CMX850 device correctly.

Real Time Clock (RTC) and Alarm Registers

The Real Time Clock (RTC) circuit in the CMX850 consists of a 32bit counter clocked once per second which is divided into four time registers (TIME0-3) and compared against four alarm registers (ALM0-3) to generate an alarm interrupt as required. Tables 8, 9 and 10 summarize the available RTC timers and alarm registers.

The RTC Control Register (RTCCON) is used to control the TIME0-3 RTC registers. This register enables and controls the RTC and the selection of the time interval set-up between regular INT6 interrupts.

Register Name	Register Address	Register bit (0 to 31) Allocation	Comments / Notes
RTCCON	$FB	0 to 7	Controls RTC TIME0-3

Table 8: RTC Control Register (RTCCON)

The TIME0-3 registers clock at 1 per second regardless of INT6 setup.
The RTC stores the time in four Special Function Register (SFR): TIME0, TIME1, TIME2 and TIME3. Each register holds eight bits, with TIME0 holding the least significant eight bits and TIME3 holding the most significant eight bits.

Register Name	Register Address	bit (0 to 31) Allocation
TIME0	$FC	0(lsb) to 7
TIME1	$FD	8 to 15
TIME2	$FE	16 to 23
TIME3	$FF	24 to 31(msb)

Table 9: RTC Time Registers

The RTC stores the 32 bit alarm time in four SFRs: ALM0, ALM1, ALM2 and ALM3. Each register holds eight bits, with ALM0 holding the least significant eight bits and ALM3 holding the most significant eight bits.

Register Name	Register Address	bit (0 to 31) Allocation
ALM0	$F4	0(lsb) to 7 of TIME0
ALM1	$F5	8 to 15 of TIME1
ALM2	$F6	16 to 23 of TIME2
ALM3	$F7	24 to 31(msb) of TIME3

Table 10: RTC Alarm Registers

Using the alarm facility is simply a case of setting the required alarm time in ALM0…3. The RTC waits until the current 32-bit time value is equal to or greater than the 32-bit alarm time, then on the next active edge of the 32.768kHz clock will assert the Int7 interrupt line to the 8051 µC.

Notes:

- As it is recommended that INT7 be configured as a level sensitive interrupt, it is the responsibility of the interrupt service routine to clear the alarm interrupt request either by clearing IE_1 bit 5 or by writing a new 32-bit alarm value to the ALM0…3 SFRs (writing to any of the ALM0…3 registers negates the Int7 signal). If this is not done, the alarm interrupt will re-trigger upon exit of the service routine.
- During the process of writing to the registers a spurious interrupt may be generated. It is suggested that the interrupt registers are masked and cleared during the writing process to stop any new interrupts.
- It is recommended that the alarm interrupt enable bit (IE_1 bit 5) be cleared whenever the ALM0…3 registers are being modified.
- There is no status bit in RTCCON for the alarm interrupt signal, therefore should it be necessary to poll the alarm interrupt it must be done from the flag bit in the interrupt control register ICON1B.

- To read the RTC TIME0, 1, 2 & 3, four reads must be performed. One for each of the registers. As the read time takes a finite time 2 read operations are recommended, when the 2 values are equal then this is deemed as the correct time value.

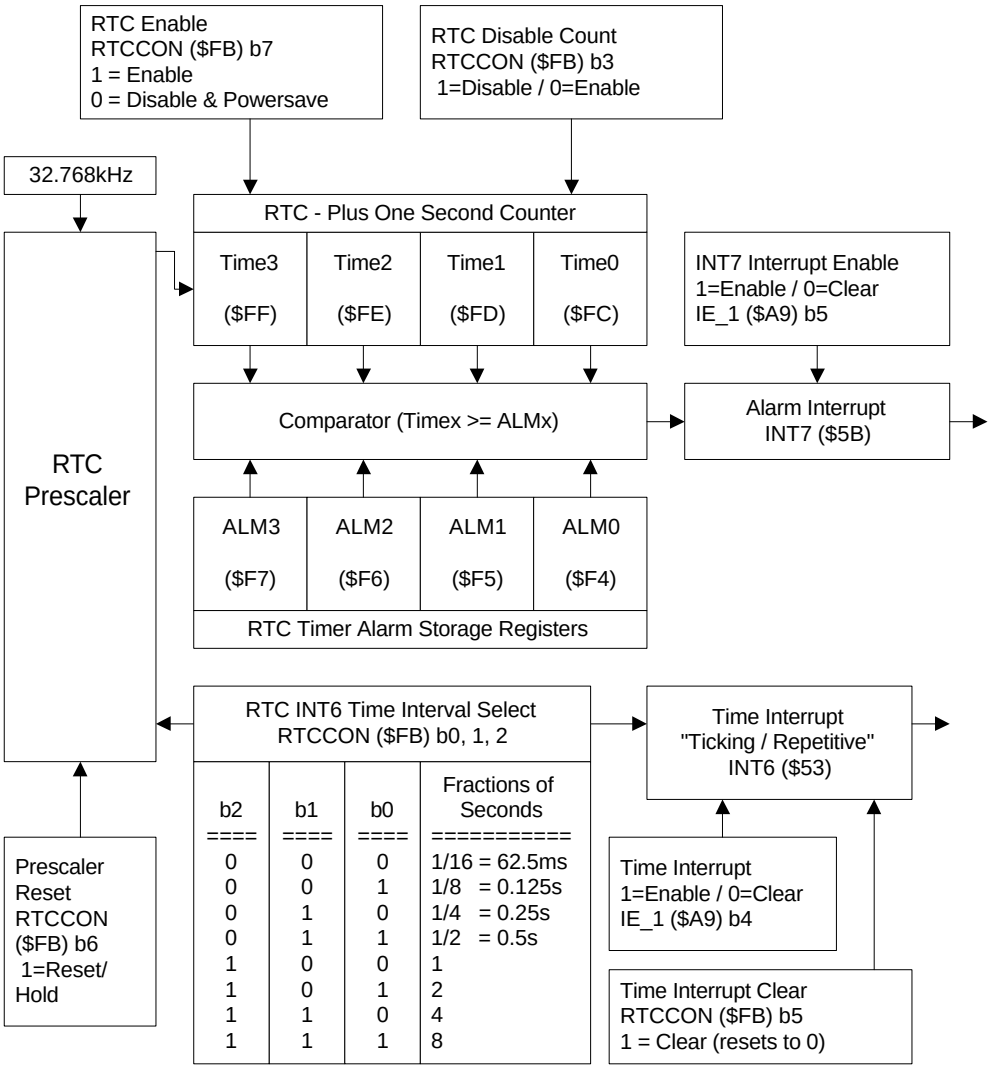


Figure 8: Block Diagram of the RTC Timer and the Alarm Timer

Please refer to the appendix for Real Time Clock sample “C” code.

Watch-Dog Timers (WDT)

The Watchdog Timer (WDT) can be used to monitor the operation of the CMX850 system. This is achieved by creating a regular WDT refresh within the software; if this refresh does not occur on time, the WDT will assume that the system has hung and will cause a system reset, preceded by an optional interrupt request to the 8051 µC.

When the interrupt before watchdog is activated the operation of the watchdog is modified. The watchdog will expire in the programmed interval as normal but instead of resetting the device it will trigger an interrupt (INT8). This interrupt then has a period of (prescale/128) seconds to determine if a watchdog reset is required. If a reset is not required then the watchdog should be refreshed and either the watchdog counter reloaded or the watchdog disabled. If the watchdog is not refreshed by the interrupt service routine then the device will reset after another (prescale/128) seconds have passed.

```
; Sample code to refresh and then disable the WDT

ORL     WDTCON,#0x02        ; Refresh the WDT
MOV     WDTCON,#0x00        ; Disable the WDT

; Sample code to refresh and then reload the WDT

ORL     WDTCON,#0x02        ; Refresh the WDT
MOV     WDTLD,#WDT_RELOAD   ; Reload the WDT with reload value
```

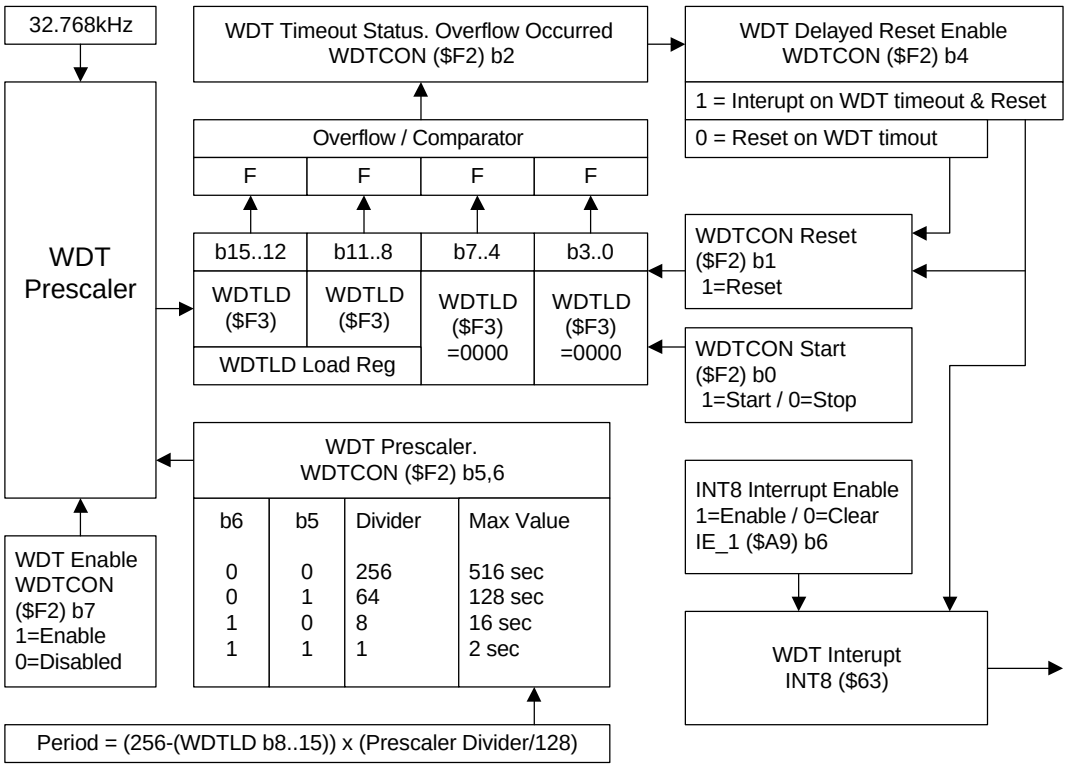


Figure 9: Block Diagram of the Watchdog Timer

Serial Port Interface

One of the 8051's many powerful features is its integrated UART, also known as a serial port. Port 3 has the alternative option of being set up as a serial data port. The fact that the 8051 has an integrated serial port means that you may very easily read and write values to the serial port and interface with a standard PC.

Setting the Serial Port Mode

The first thing we must do when using the CMX850's integrated serial port is configure it. This lets us tell the 8051 how many data bits will be used, the desired baud rate and how the baud rate will be determined.

The "Serial Control" and SCON SFR (listed in Table 12) is used to configure the serial port. Port 3 bits 7-0 are available as I/O pins, but each pin also has an alternative output function, as shown in Table 11.

CMX850 Pin No.	PORT/BIT	ADDRESS	**NAME**	ALTERNATIVE FUNCTION
9	P3.0	$B0	RXD	Serial port receive data
10	P3.1	$B1	TXD	Serial port transmit data
11	P3.2	$B2	Int0	Int0 (External interrupt 0)
12	P3.3	$B3	Int1	External interrupt 1
13	P3.4	$B4	T0	Timer/counter 0 external input
14	P3.5	$B5	T1	Timer/counter 1 external input
15	P3.6	$B6	PWM1	Pulse-Width Modulator 1 output
16	P3.7	$B7	PWM2	Pulse-Width Modulator 2 output

Table 11: P3 Port Pin Usage

Note that pins 15 (P3.6) and pin 16 (P3.7) are automatically configured as an output when the associated PWM block is enabled. The other alternative pin functions require the correct pin direction to be explicitly configured using the Port 3 P3DIR register (SFR $B1).

NAME	ADDRESS	DESCRIPTION
SM0	$9F	Serial port mode select bit 0 & bit 1 These two bits determine the serial port-operating mode, as illustrated in table 13. The serial port-operating mode selects the mode of operation (8-bit/9-bit, UART or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillator's frequency. In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows.
SM1	$9E	
SM2	$9D	Serial port mode select bit 2 This bit is a flag for "Multiprocessor communication". Generally, whenever a byte has been received the 8051 will set the Receive Interrupt (RI) flag, which lets the program know that a byte has been received and that it needs to be read & processed. However, when SM2 is set, the "RI" flag will only be triggered if the 9th bit received was a "1". That is to say, if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set. This can be useful in certain advanced serial applications.
REN	$9C	Serial port receive enable. - (This bit must be set in order to receive characters). "Receiver Enable". If you want to receive data via the serial port, set this bit. You will almost always want to set this bit.
TB8	$9B	9th Transmit data bit. - (The 9th bit to transmit in mode 2 and 3). This bit is used as the 9th "data bit" in modes 2 and 3. If TB8 is set and a value is written to the serial port, the data bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear". This is used to implement parity systems.

RB8	$9A	8th Receive data bit. - (The 9th bit received in mode 2 and 3). This bit is relevant for modes 2 and 3 and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In this case, the first eight bits received are the data of the serial byte received and the value of the ninth bit received will be placed in RB8. This is used to implement parity systems.
TI	$99	Transmit interrupt Flag - (Set when a byte has been completely transmitted.) "Transmit Interrupt". When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" of the serial port. If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the 8051 lets the program know that it has "clocked out" the last bit by setting the TI bit. When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte.
RI	$98	Receive interrupt Flag. - (Set when a byte has been completely received) "Receive Interrupt". It functions similarly to the "TI" bit, but it indicates that a byte has been received. That is to say, whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value before another byte is received.

Table 12: SCON Register

Bit 7 (SM0)	Bit 6 (SM1)	Serial Port mode
0	0	Mode 0 = Shift register, baud rate = (fosc / 12) **SM2=0
0	1	Mode 1 = 8-bit UART, baud rate = variable
1	0	Mode 2 = 9-bit UART, baud rate = (fosc / 64) or (fosc / 12)
1	1	Mode 3 = 9-bit UART, baud rate = variable
Bit 5 (SM2)	Enables the microprocessor communications features. In modes 2 & 3, if SM2=1 then RI will not be activated if the 9th received data bit (RB8) is 0 In mode 1, If SM2=1 then RI will not be activated if a valid stop bit was not received. **In Mode 0 then SM2=0.	

Table 13: SCON Configuration for Serial Port Operating Modes

Setting the Serial Port Baud Rate

In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if your crystal is 11.059Mhz, mode 0 baud rate will always be 921,583 baud.

In mode 2 the baud rate is always the oscillator frequency divided by 64, so an 11.059Mhz crystal speed will yield a baud rate of 172,797.

If the Serial Port mode has been configured as mode 1 or 3, then the program must configure the serial port's baud rate.

The baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode 2) and set a reload value (TH1) that causes Timer 1 to

overflow at a frequency appropriate to generate a baud rate. The resulting baud rate is variable and is calculated from:

$$BAUD\_RATE = \frac{(K \times f_{osc})}{(32 \times 12 \times (256 - TH1))}$$

where:

K = 1 if SMOD = 0
K = 2 if SMOD = 1
SMOD = bit 7 of the PCON reg ($87)
fosc = Crystal frequency
TH1 = Timer/Counter 1 high byte register ($8D)

This situation only applies to Serial Port modes 1 and 3 otherwise the Baud Rate is determined based on the oscillator's frequency.

To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation (assuming PCON bit 7 (SMOD) is clear).

TH1 = 256 - (Crystal / (384 x Baud Rate)) ……..(1)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - (Crystal / (192 x Baud Rate)) ……..(2)

For example, if we have an 11.059Mhz crystal and we want to configure the serial port to 19,200 baud:

TH1 = 256 - (Crystal / (384 x Baud))
TH1 = 256 - (11059000 / (384 x 19200))
TH1 = 256 - 1.5 = 254.5

As you can see, to obtain 19,200 baud on a 11.059Mhz crystal we'd have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud…Therefore...

To achieve a baud rate of 19,200: we need to set the Power Control PCON bit 7 (SMOD) register. When we do this we double the baud rate and utilize the second equation mentioned above.

Thus we have:

TH1 = 256 - (Crystal / (192 x Baud))
TH1 = 256 - (11059000 / (192 x 19200))
TH1 = 256 - 3 = 253

Here we are able to calculate a nice, even TH1 value. Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1. Configure Serial Port mode 1 or 3.
2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
3. Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
4. Set the Power Control PCON bit 7 (SMOD) register to double the baud rate.

To achieve a baud rate of 9,600: we need to clear the Power Control PCON bit 7 (SMOD) register and utilize the first equation mentioned above:

TH1 = 256 - (Crystal / (384 x Baud))
TH1 = 256 - (11059000 / (384 x 9600))
TH1 = 256 - 3 = 253

Here we are able to calculate a nice, even TH1 value. Therefore to obtain 9600 baud rate with an 11.059MHz crystal we must:

1. Configure Serial Port mode 1 or 3.
2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
3. Set TH1 to 253 to reflect the correct frequency for 9,600 baud.
4. Clear the Power Control PCON bit 7 (SMOD) register.

TH1 = 256 – (fosc / (384 x Baud Rate))……..for K=1		
	SMOD = 0	SMOD = 0
	K = 1	K = 1
	fosc = 11,059,200	fosc = 12,288,000*
Baud Rate	TH1 (dec / hex)	TH1 (dec / hex)
1200	232 / E8	203 / CB
2400	244 / F4	243 / F3
4800	250 / FA	249 / F9
9600	253 / FD	253 / FD

Table 14: Baud Rate Settings with PCON.7=0

For a baud rate of 19200 the PCON.7 (SMOD) must be set '1' and the equation changes to:

TH1 = 256 – (fosc / (192 x Baud Rate))……..for K=2		
	SMOD = 1	SMOD = 1
	K = 2	K = 2
	fosc = 11,059,200	fosc = 12,288,000
Baud Rate	TH1 (dec / hex)	TH1 (dec / hex)
19200	250 / FA	249 / F9

* The TH1 values for the fosc = 12.288MHz are rounded to the nearest whole figure.

Table 15: Baud Rate Settings with PCON.7=1

Writing to the Serial Port

Once the Serial Port has been properly configured as explained above, the serial port is ready to send data and receive data.

To write a byte to the serial port, write the value to the SBUF ($99) SFR. For example, if you wanted to send the character "A" to the serial port, it could be accomplished with:

MOV SBUF,#'A'

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Since the 8051 does not have a serial output buffer we need to be sure that the character is completely transmitted before we try to transmit the next character.

The 8051 lets us know when it is done transmitting a character by setting the TI bit in SCON. When this bit is set the last character has been transmitted and the next character, if any, can be sent. Consider the following code segment:

```
JNB TI,$          ;Wait for the TI bit to set thus ready for  a new byte.
CLR TI            ;Clear TI for the next character to be transmitted
MOV SBUF,#'A'     ;Send the character 'A' to the serial port SBUF Register.
JNB TI,$          ;Pause until byte is transmitted, the TI bit is set.
```

The above three instructions will wait for the serial port to transmit the last character, clear the TI transmit ready flag and load the character "A" into the serial port to start transmission.

Reading the Serial Port

To read a byte from the serial port one just needs to read the value stored in the SBUF ($99) SFR after the 8051 has automatically set the RI flag in the SCON SFR.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

```
JNB RI,$          ;Wait for the 8051 to set the RI flag, byte received.
MOV A,SBUF        ;Read the character from the serial port.
CLR RI            ;Indicate character read from serial port.
```

The first line of the above code segment waits for the 8051 to set the RI flag; again, the 8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously.

Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction that reads the value.

Lastly the RI flag is cleared to allow the next character reception to be recognised.

RS232 Connection to a PC

The connection to a PC is fairly straightforward but as the CMX850 is operated at 3.3Vdc the output voltages are below the RS232c standard levels, therefore a level-shifting device will be required to change the CMX850 output to a RS232 standard.

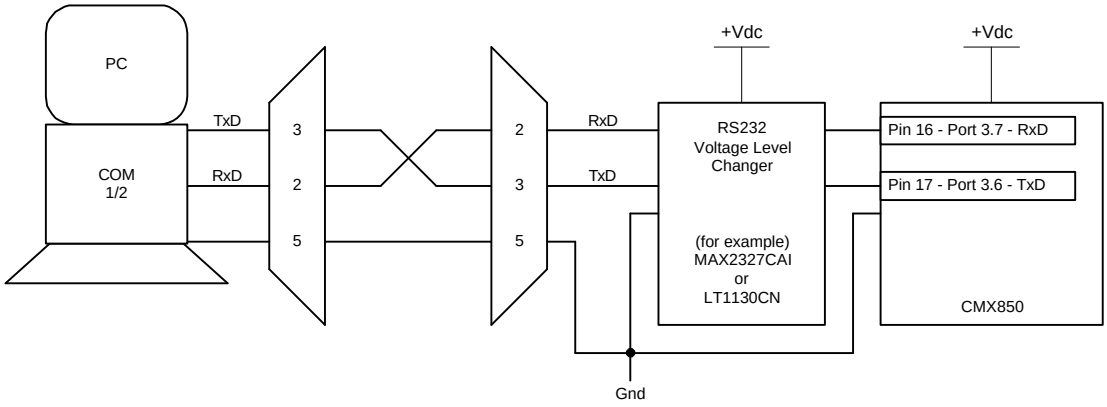


Figure 10: Serial Port Wiring Connection - For a more comprehensive example of

RS232 interface circuit diagram, please refer to the CML Microcircuits' EV8500 circuit schematics.

CAS/FSK Detector Block

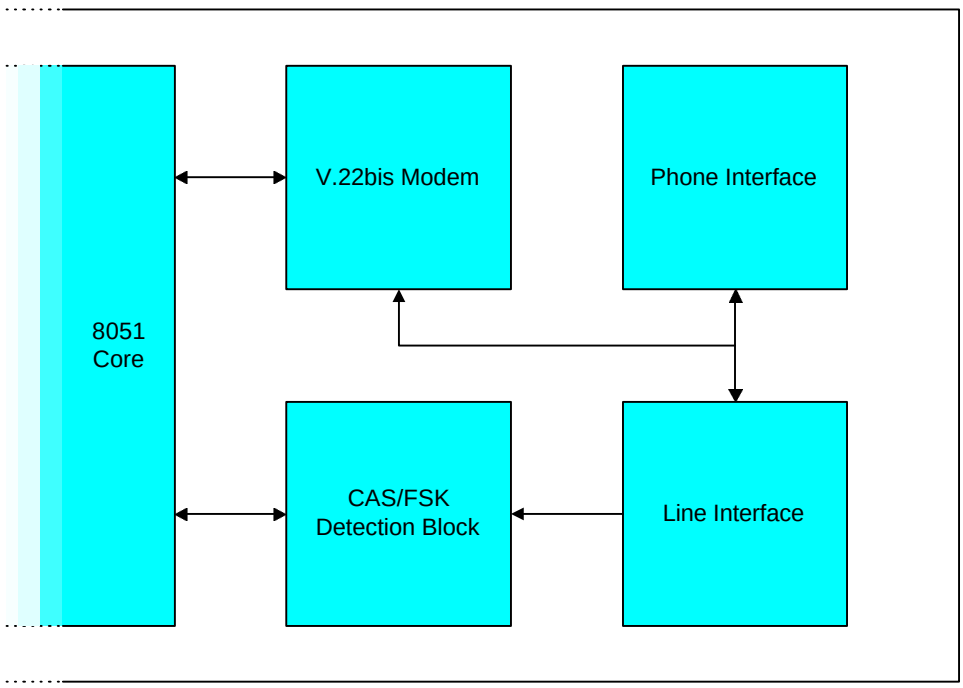


Figure 11: Block Diagram of the CAS/FSK signal interface.

The CPE (Customer Premise Equipment) Alerting Signal, or CAS Tone, is typically a 80ms burst of simultaneous 2130Hz and 2750Hz tones, although it cab be optimised a maximum of a 135ms CAS tone. The CAS tone is typically encountered in Caller ID on Call Waiting applications that allow the reception of Caller ID information while the telephone is in use (off-hook). An integrated CAS/FSK detector is provided in the CMX850. Simultaneous CAS detection capability and FSK decoding allow the CMX850 to serve in both on-hook and off-hook Caller ID applications in Europe, Scandinavia and the Americas and beyond.

The CAS/FSK detector block is configured through the CASDET SFR and can be placed into two modes of operation, CAS detection and FSK detection and formatting.

CAS Tone detection

To select the CAS detector Bit 6 of the CASDET SFR should be disabled, set low. This is the default CMX850 bit setting upon power-up or reset.

The CAS tone detector is further configured through the CASDET SFR:

- Bit 7 of CASDET is used to enable the CAS detector.
  - This bit also activates the Vbias generator and the "line" input amplifier.
- Bits 5-3 of CASDET determine the tone duration required before CAS qualification is made.

The CASDET SFR responds to the presence of a CAS tone as follows:

- CASDET b0 ("DETECT" bit) will be set as long as a CAS tone is detected.
- If the CAS tone is present for the time duration specified by CASDET b5-3:

- Once CAS tone goes away, CASDET b1 (“INTERRUPT STATUS”) will be set to indicate that a valid CAS tone has been detected.
- INT2 (external interrupt 2) will become active, which notifies the 8051 of the CAS detection.
- INT2 can be enabled or disabled with IE_1.0.

To clear the INTERRUPT STATUS bit and INT2, the INTERRUPT CLEAR bit (CASDET b2), should be toggled high.

FSK detection

To enable the FSK detector Bit 6 of the CASDET SFR should be enabled, set high.

The FSK detector block is further configured through the CASDET SFR:

- Bit 7 of CASDET is used to enable the CAS /FSK detector.
  - This bit also activates the Vbias generator and the “line” input amplifier.
- Bit 3 of CASDET selects the data receive mode. Setting this bit to “1” selects bit mode and a “0” settings selects byte mode.

Bit mode

The CASDET SFR responds to the presence of a FSK data bit as follows:

- There is no interrupt mechanism for bit mode operation so a polling mechanism should be employed in the 8051 code that reads FSKBUF SFR, b7 at regular consecutive intervals at no more than 800us approx apart.

Note: Voice and other in-band noise may also trigger the FSK detector so depending on the application it may be necessary to mute the local voice.

Byte mode

The CASDET SFR responds to the presence of a FSK data byte as follows:

- When a valid FSK character has been detected an interrupt will be generated, and the Interrupt Status bit b1 of the CASDET Register should be read. Int2 will also be set if it is enabled.
- As soon as the interrupt has been read and the 8 data bits read from the FSKBUF SFR all the interrupts will be cleared, the process will be repeated as long as valid data is being received.

Note: This mechanism is designed for asynchronous data that has 1 start bit, 8 data bits and a stop bit. Noise or synchronous data reception can cause indeterminate data to be stored in the FSKBUF SFR. 55h preamble pattern will be correctly loaded as 55h

Note: Since the CAS/FSK detector block operates independently of the modem block, the 8051 µC must “wake up” the modem portion of the CMX850 upon indication of CAS tone detection for the subsequent data transfer.

Interrupts

Interrupt Sources and Vector Addresses

The CMX850 has thirteen vectored interrupt sources, including the original five interrupt sources from the original 8051 Microcontroller and eight additional interrupt sources. The thirteen-interrupt sources available in the CMX850 are listed below (interrupt sources used in standard 8051 Microcontrollers are in parentheses):

Interrupt signal	Hardware source	Vector address	Bit to enable interrupt	Bit to assign priority	Priority within level (lower # has higher priority)
(Int0)	Interrupt 0 input pin (P3.2)	$03	IE.0	IP.0	2
(Timer0)	8051 timer 0	$0B	IE.1	IP.1	4
(Int1)	Interrupt 1 input pin (P3.3)	$13	IE.2	IP.2	6
(Timer1)	8051 timer 1	$1B	IE.3	IP.3	8
(Serial)	RI or TI from 8051 serial port	$23	IE.4	IP.4	10
Int2	CAS Detect	$33	IE_1.0	IP_1.0	1 (Highest)
Int3	DSP Modem	$3B	IE_1.1	IP_1.1	3
Int4	Keyboard encoder	$43	IE_1.2	IP_1.2	5
Int5	A/D converter	$4B	IE_1.3	IP_1.3	7
Int6	Real-time clock (RTC) time interrupt	$53	IE_1.4	IP_1.4	9
Int7	RTC alarm interrupt	$5B	IE_1.5	IP_1.5	11
Int8	Watchdog timeout	$63	IE_1.6	IP_1.6	12 (Lowest)
Int9	Interrupt 9 input pin (D6)*	$6B	IE_1.7	N/A (SUPER PRIORITY)	N/A (SUPER PRIORITY)

Table 16: CMX850 Interrupts

Note*, only available in multiplexed address mode

Enabling and Disabling Interrupts

The interrupts flag must first be enabled before any interrupt other than INT9 can be activated; this is done by setting bit 7 in the IE (Interrupt Enable) SFR. Once bit 7 in the IE SFR is set, individual interrupts can be enabled or disabled by manipulating the appropriate bits in either the IE or the IE_1 (Interrupt Enable_1) SFR. The five "standard" 8051 interrupts (Serial, Timer 0, Timer 1, external interrupt 1, external interrupt 2) are enabled/disabled with the IE SFR, and the CMX850 specific interrupts are controlled through the IE_1 SFR.

Int9 is a "super priority" interrupt source that is intended for use in debugging with an external PROM emulator. Int9 is the only interrupt not affected by bit 7 of the IE SFR; instead, it is enabled/disabled by bit 7 of the IE_1 SFR.

When an interrupt is enabled and an "interruptible event" occurs at the interrupt source (e.g. timer expires, serial data byte has finished transmission, etc.), an interrupt request is generated and the CMX850 uC will cause program execution to vector to the predefined memory address listed above ("Vector Address" column in table). An interrupt service routine (ISR) should be located at the vectored address, the ISR is executed and program control is then returned to the main program at the point where program flow was interrupted when the ISR completes.

Interrupt Priorities

The CMX850 priority handling scheme is two-tiered and is identical to that used in the original 8051: high/low priority (top tier) and priority-within-level (lower tier).

Each interrupt can be assigned as either a high or low priority by manipulating bits in the IP and IP_1 (Interrupt Priority & Interrupt Priority_1) SFR. Priorities (e.g. high or low) for the five "standard" 8051 interrupts (Serial, Timer 0, Timer 1, external interrupt 1, external interrupt 2) are set via the IP SFR, while the priorities for the CMX850 specific interrupts are set via the IP_1 SFR.

The "priority within level" rankings determine which interrupt takes precedence if interrupt requests of the same priority occur simultaneously. For example, if INT3 and TIMER0 occur simultaneously and are both high priority, INT3 will be serviced first because it has a higher priority-within-level than does TIMER0.

All interrupts, with the exception of Int9, are controlled by the CMX850 interrupt logic circuitry described above. Int9 is permanently configured with "super priority" and will interrupt any other interrupt regardless of priority.

Interrupt Handling

The CMX850 CPU samples the condition of its thirteen-interrupt sources at the completion of every machine cycle. This information is analysed during the next machine cycle (i.e. M1). If an interrupt has occurred a call to the associated vector address will take place over the next two machine cycles (i.e. M2-M3). The start of the associated ISR will be situated at this vector address.

The ISR will begin execution with the fourth machine cycle after the initial interrupt conditions were recorded (i.e. M4). ISR execution will continue until the ISR is complete or until a higher priority interrupt occurs. When the ISR is complete the program will return to the instruction immediately following the one that was being processed prior to the interrupt.

There are three conditions under which an interrupt flag will not cause an immediate call to its ISR (e.g. interrupt vector address):

1) An interrupt of equal or higher priority is already in progress. In this condition, the newly activated interrupt is considered "pending" and its own ISR will be called after the current interrupt has been fully serviced.

2) The current machine cycle is not the final machine cycle of the instruction being executed. When an interrupt flag occurs in this situation, the current instruction must be fully completed before the ISR can be called.
3) The current instruction is either (a) “return from interrupt” (RETI) or (b) an instruction that modifies the IE, IE_1, IP, or IP_1 register. When this happens, at least one additional instruction (beyond the current instruction) must be completed before the ISR can be called.

Interrupt Activation Levels

Most interruptible events can be signalled in one of two ways; a change of logic state for the interrupt source (“edge triggered”), or a low logic level condition for the interrupt source (“level triggered”). These types of interrupts require the manipulation of “type control” bits to determine whether edge triggering or level triggering is to be used.

Edge triggered interrupt flags are automatically cleared when the corresponding interrupt service routine is called. A level triggered interrupt flag is cleared when the event that caused the interrupt is cleared.

Some interruptible events, such as timer interrupts, are by definition purely edge triggered (interrupts occur when counter register rolls over from all 1s to 0s) and therefore do not require type control bits.

It is recommended that the type control bits for INT2-8 be cleared to 0 to configure them as level triggered interrupts.

Interrupt	Hardware Source	“Type Control” bit location	Interrupt Flag location
TIMER0	Internal timer 0	N/A	TCON.5
TIMER1	Internal timer 1	N/A	TCON.7
SERIAL	Serial port (RI or TI)	N/A	SCON.0 (RI) and SCON.1 (TI)
INT0	External interrupt 0	TCON.0	TCON.1
INT1	External interrupt 1	TCON.2	TCON.3
INT2	External interrupt 2	ICON1A.0	ICON1A.1
INT3	External interrupt 3	ICON1A.2	ICON1A.3
INT4	External interrupt 4	ICON1A.4	ICON1A.5
INT5	External interrupt 5	ICON1A.6	ICON1A.7
INT6	External interrupt 6	ICON1B.0	ICON1B.1
INT7	External interrupt 7	ICON1B.2	ICON1B.3
INT8	External interrupt 8	ICON1B.4	ICON1B.5
INT9	Super Priority	ICON1B.6	ICON1B.7

Table 17: CMX850 Interrupt Configuration Locations

Instructions and Addressing

CMX850 Instruction Set

The instructions used to control the CMX850 are listed below.

Instruction	**Description**	# OF BYTES	# OF OSC CYCLES
ACALL addr11	ABSOLUTE (SUBROUTINE) CALL command. This command (1) increments the program counter (PC) twice, (2) pushes the 16-bit PC result onto the stack, (3) increments the stack pointer (SP) twice. The destination address is then determined by concatenating the upper five bits of the PC with bits 7-5 of the ACALL opcode, and the second instruction byte. Destination must be in the same 2Kbyte block as the jump. The 2Kbyte blocks are fixed in position.	2	24
ADD A, @Ri	ADD operation. This command adds the contents of the RAM location whose address is in bank register “i” to the contents of the accumulator, then leaves the result in the accumulator.	1	12
ADD A, direct	ADD operation. This command adds the contents of RAM address “direct” to the contents of the accumulator, then leaves result in accumulator. (“Direct” is an 8-bit address for either internal RAM or SFR.).	2	12
ADD A, Rn	ADD operation. This command adds the byte contained in bank register “n” to the contents of the accumulator, then leaves result in accumulator.	1	12
ADD A, #data	ADD operation. This command adds the immediate data byte to the contents of the accumulator, then leaves the result in the accumulator.	2	12
ADDC A, #data	ADD WITH CARRY operation. This command adds the carry flag and the immediate data byte to the contents of the accumulator, then leaves result in accumulator. This instruction is helpful for addition of 16-bit or larger numbers.	2	12
ADDC A, @Ri	ADD WITH CARRY operation. This command adds the carry flag and the contents of the RAM location whose address is in bank register “i” to the contents of the accumulator, then leaves result in accumulator. This instruction is helpful for addition of 16-bit or larger numbers.	1	12

ADDC A, direct	ADD WITH CARRY operation. This command adds the carry flag and contents of RAM address “direct” to the contents of the accumulator, then leaves result in accumulator. This instruction is helpful for addition of 16-bit or larger numbers. (“Direct” is an 8-bit address for either internal RAM or SFR.).	2	12
ADDC A, Rn	ADD WITH CARRY operation. This command adds the carry flag and the byte contained in bank register “n” to the contents of the accumulator, then leaves result in accumulator. This instruction is helpful for addition of 16-bit or larger numbers.	1	12
AJMP addr11	ABSOLUTE JUMP command. This command increments the PC by 2 and then transfers program execution to the indicated address. The destination address is then determined by concatenating the upper five bits of the PC with bits 7-5 of the ACALL opcode, and the second instruction byte. Destination must be in the same 2Kbyte block as the jump. The 2Kbyte blocks are fixed in position.	2	24
ANL A, #data	LOGICAL AND byte operation. This command performs a bitwise-AND operation between the accumulator value and the immediate data byte, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	2	12
ANL A, @Ri	LOGICAL AND byte operation. This command performs a bitwise-AND operation between the accumulator value and the contents of the RAM location whose address is in bank register “i”, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	1	12
ANL A, direct	LOGICAL AND byte operation. This command performs a bitwise-AND operation between the accumulator value and the contents of RAM address “direct”, with the result stored in the accumulator. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	2	12
ANL A,Rn	LOGICAL AND byte operation. This command performs a bitwise-AND operation between the accumulator value and the value in bank register “n”, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	1	12

ANL C, /bit	LOGICAL AND bit command. This command performs an AND operation on the carry bit of the PSW and the inverse of the indicated bit, with the result stored in the carry bit. The indicated bit (source bit) is not altered.	2	24
ANL C, bit	LOGICAL AND bit command. This command performs an AND operation on the carry bit of the PSW and the indicated bit, with the result stored in the carry bit. The indicated bit (source bit) is not altered.	2	24
ANL direct, #data	LOGICAL AND byte operation. This command performs a bitwise-AND operation between the the contents of RAM address “direct” and the immediate data byte, with the result stored in RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins. This instruction can be used to mask combinations of bits in the “direct” RAM register or SFR.	3	24
ANL direct, A	LOGICAL AND byte operation. This command performs a bitwise-AND operation between the the contents of RAM address “direct” and the accumulator value, with the result stored in RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins. This instruction can be used to mask combinations of bits in the “direct” RAM register or SFR.	2	12
CJNE @Ri, #data, rel	COMPARE AND JUMP IF NOT EQUAL command. This command compares the magnitudes of the contents of the RAM location whose address is stored in bank register “i” and the immediate data byte. If the magnitudes are not equal, the PC is incremented by 3 (to form the address of the next instruction) and program execution transfers to the address whose sum is PC + “rel”. Otherwise, program execution continues with the next instruction. If the unsigned integer value of the contents of RAM location whose address is stored in bank register “i” are less than the unsigned integer value of the immediate data byte, the carry flag (PSW) is set. Otherwise, the carry flag will be cleared. The contents of the RAM location whose address is stored in bank register “i” contents are not altered.	3	24

CJNE A, #data, rel	COMPARE AND JUMP IF NOT EQUAL command. This command compares the magnitudes of the accumulator contents and the immediate data byte. If the magnitudes are not equal, the PC is incremented by 3 (to form the address of the next instruction) and program execution transfers to the address whose sum is PC + "rel". Otherwise, program execution continues with the next instruction. If the unsigned integer value of the accumulator is less than the unsigned integer value of the immediate data byte, the carry flag (PSW) is set. Otherwise, the carry flag will be cleared. The accumulator contents are not altered.	3	24
CJNE A, direct, rel	COMPARE AND JUMP IF NOT EQUAL command. This command compares the magnitudes of the accumulator contents and the "direct" RAM address. ("Direct" is an 8-bit address for either internal RAM or SFR.) If the magnitudes are not equal, the PC is incremented by 3 (to form the address of the next instruction) and program execution transfers to the address whose sum is PC + "rel". Otherwise, program execution continues with the next instruction. If the unsigned integer value of the accumulator is less than the unsigned integer value of the "direct" RAM address, the carry flag (PSW) is set. Otherwise, the carry flag will be cleared. The accumulator contents and "direct" value are not altered.	3	24
CJNE Rn, #data, rel	COMPARE AND JUMP IF NOT EQUAL command. This command compares the magnitudes of the bank register "n" contents and the immediate data byte. If the magnitudes are not equal, the PC is incremented by 3 (to form the address of the next instruction) and program execution transfers to the address whose sum is PC + "rel". Otherwise, program execution continues with the next instruction. If the unsigned integer value of the bank register "n" contents are less than the unsigned integer value of the immediate data byte, the carry flag (PSW) is set. Otherwise, the carry flag will be cleared. The bank register "n" contents and the immediate data byte are not altered.	3	24
CLR A	CLEAR operation. This command sets all bits in the accumulator to zero.	1	12
CLR bit	CLEAR command. This command clears the indicated bit. No other bits are affected.	2	12
CLR C	CLEAR command. This command clears the carry bit of the PSW. No other bits are affected.	1	12

CPL A	COMPLEMENT operation. This command logically complements all bits in the accumulator. Bits, which were zeros before command execution, become ones after command execution, and vice versa.	1	12
CPL bit	COMPLEMENT command. This command complements the indicated bit (changes a one to a zero, and vice versa). No other bits are altered.	2	12
CPL C	COMPLEMENT command. This command complements the carry bit (changes a one to a zero, and vice versa). No other bits are altered.	1	12
DA A	DECIMAL ADJUST ACCUMULATOR FOR ADDITION operation. This command applies a correction factor that transforms the accumulator value into two correct 4-bit BCD (binary coded decimal) nibbles. DA command will cause six to be added to either the high-order or low-order accumulator nibble if either nibble is greater than nine. (BCD numbers cannot be greater than nine.) Additionally, the DA command will cause six to be added to the high-order accumulator nibble if the CY flag (PSW.7 bit) is set, or to the low-order accumulator nibble if the AC flag (PSW.6 bit) is set.	1	12
DEC @Ri	DECREMENT operation. This command decrements the contents of the RAM location whose address is in bank register “i” by one.	1	12
DEC A	DECREMENT operation. This command decrements the contents of the accumulator by one.	1	12
DEC direct	DECREMENT operation. This command decrements the contents of RAM address “direct” by one. (“Direct” is an 8-bit address for either internal RAM or SFR.)	2	12
DEC Rn	DECREMENT operation. This command decrements the data byte stored in bank register “n” by one.	1	12
DIV AB	DIVIDE operation. This command divides the contents of the accumulator by the contents of the B register. The integer portion (quotient) of the result is stored in the accumulator while the remainder is stored in the B register.	1	48

DJNZ direct, rel	DECREMENT AND JUMP IF NOT ZERO command. This command decrements the contents of RAM address “direct” by one. An original register value of 00h will underflow to FFh. (“Direct” is an 8-bit address for either internal RAM or SFR.) If these decremented contents are not zero, the PC is incremented by 2 and program execution transfers to the address of (PC + “rel”.) If the decremented contents are zero, program execution continues with the next instruction. When the DJNZ command is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	3	24
DJNZ Rn, rel	DECREMENT AND JUMP IF NOT ZERO command. This command decrements the contents of bank register “n” by one. An original register value of 00h will underflow to FFh. If these decremented contents are not zero, the PC is incremented by 2 and program execution transfers to the address of (PC + “rel”.) If the decremented contents are zero, program execution continues with the next instruction.	2	24
INC @Ri	INCREMENT operation. This command increments the contents of the RAM location whose address is in bank register “i” by one.	1	12
INC A	INCREMENT operation. This command increments the accumulator contents by one.	1	12
INC direct	INCREMENT operation. This command increments the contents of RAM address “direct” by one. (“Direct” is an 8-bit address for either internal RAM or SFR.).	2	12
INC DPTR	INCREMENT operation. This command increments the contents of the currently selected data pointer by one. (NOTE: The DPS SFR determines which data pointer is currently selected.)	1	24
INC Rn	INCREMENT operation. This command increments the data byte stored in bank register “n” by one.	1	12
JB bit, rel	JUMP IF BIT SET command. This command will cause the program to branch if the indicated bit is set; otherwise, the program will continue with the next instruction. The program counter (PC) is incremented to the first byte of the next instruction, and then added to the signed relative-displacement byte in the third instruction byte, to arrive at the branching location address. The indicated bit is not altered.	3	24

JBC bit, rel	JUMP IF BIT SET AND CLEAR command. If the indicated bit is set, this command will cause the indicated bit to be cleared and the program to branch; otherwise, the program will continue with the next instruction. The program counter (PC) is incremented to the first byte of the next instruction, and then added to the signed relative-displacement byte in the third instruction byte, to arrive at the branching location address.	3	24
JC rel	JUMP IF CARRY SET command. This command will cause the program to branch if the carry bit is set; otherwise, the program will continue with the next instruction. The program counter (PC) is incremented twice and then added to the signed relative-displacement byte in the second instruction byte to arrive at the branching location address.	2	24
JMP @A + DPTR	JUMP INDIRECT command. This command adds the accumulator contents to the DPTR value, and the sum is then passed to the PC. The next instruction fetches will be use the address held in the PC. The accumulator and DPTR contents are not altered.	1	24
JNB bit, rel	JUMP IF BIT NOT SET command. This command will cause the program to branch if the indicated bit is not set; otherwise, the program will continue with the next instruction. The program counter (PC) is incremented to the first byte of the next instruction, and then added to the signed relative-displacement byte in the third instruction byte, to arrive at the branching location address. The indicated bit is not altered.	3	24
JNC rel	JUMP IF CARRY NOT SET command. This command will cause the program to branch if the carry bit is not set; otherwise, the program will continue with the next instruction. The program counter (PC) is incremented twice and then added to the signed relative-displacement byte in the second instruction byte to arrive at the branching location address.	2	24
JNZ rel	JUMP IF ACCUMULATOR NOT ZERO command. If any accumulator bit is a one, the PC is incremented twice and program execution branches to the sum of the PC and the “rel” address. If all accumulator bits are zero, program execution continues with the next instruction. The accumulator contents are not modified.	2	24
JZ rel	JUMP IF ACCUMULATOR ZERO command. If all accumulator bits are zero, the PC is incremented twice and program execution branches to the sum of the PC and the “rel” address. If any accumulator bit is one, program execution continues with the next instruction. The accumulator contents are not modified.	2	24

LCALL addr16	LONG (SUBROUTINE) CALL command. This command first adds 3 to the PC to provide the address for the next instruction. The command then increments the SP by one and pushes the low-order byte of the modified PC address onto the SP. The SP is then incremented by one again, and the high-order byte of the modified PC address is pushed onto the SP. The designated "addr16", which corresponds to the desired subroutine, is then loaded into the PC and program execution begins at that location. The called subroutine can begin anywhere in the 64kbyte program memory space.	3	24
LJMP addr16	LONG JUMP command. This command unconditionally branches to the indicated address, which can be anywhere within the 64k program memory space. The "addr16" high and low order bytes are loaded into the PC.	3	24
MOV @Ri, #data	MOVE byte command. This command copies the contents of the immediate data byte into the RAM location whose address is in bank register "i". The immediate data byte is not altered.	2	12
MOV @Ri, A	MOVE byte command. This command copies the contents of the accumulator into the RAM location whose address is in bank register "i". The accumulator contents are not altered.	1	12
MOV @Ri, direct	MOVE byte command. This command copies the contents of the RAM address "direct" into the RAM location whose address is in bank register "i". ("Direct" is an 8-bit address for either internal RAM or SFR.) The contents of RAM address "direct" are not altered.	2	24
MOV A, #data	MOVE byte command. This command copies the immediate data byte into the accumulator.	2	12
MOV A, @Ri	MOVE byte command. This command copies the contents of the RAM location whose address is in bank register "i" into the accumulator. Contents of the RAM location whose address is in bank register "i" are not altered.	1	12
MOV A, direct	MOVE byte command. This command copies the contents of RAM address "direct" into the accumulator. ("Direct" is an 8-bit address for either internal RAM or SFR.) Contents of RAM address "direct" are not altered.	2	12
MOV A, Rn	MOVE byte command. This command copies the contents of bank register "n" into the accumulator. Bank register "n" contents are not altered.	1	12
MOV bit, C	MOVE bit command. This command copies the contents of the carry bit of the PSW into the indicated bit.	2	24

MOV C, bit	MOVE bit command. This command copies the contents of the indicated bit into the carry bit of the PSW.	2	12
MOV direct, #data	MOVE byte command. This command copies the immediate data byte into the RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.)	3	24
MOV direct, @Ri	MOVE byte command. This command copies the contents of the RAM location whose address is in bank register “i” into the RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) The contents of the RAM location whose address is in bank register “i” is not altered.	2	24
MOV direct, A	MOVE byte command. This command copies the contents of the accumulator into the RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) Accumulator contents are not altered.	2	12
MOV direct, direct1	MOVE byte command. This command copies the contents of the latter RAM address “direct1” into the former RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) Contents of the latter RAM address “direct1” are not altered.	3	24
MOV direct, Rn	MOVE byte command. This command copies the contents of bank register “n” into the RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) Contents of bank register “n” are not altered.	2	24
MOV DPTR, #data16	MOVE byte command. This command copies the contents of the 16-bit constant into the currently selected data pointer register. (NOTE: The DPS SFR determines which data pointer is currently selected.).	3	24
MOV Rn, #data	MOVE byte command. This command copies the immediate data byte into the bank register “n”.	2	12
MOV Rn, A	MOVE byte command. This command copies the accumulator contents into bank register “n”. The accumulator contents are not altered.	1	12
MOV Rn, direct	MOVE byte command. This command copies the contents of the RAM address “direct” into the bank register “n”. (“Direct” is an 8-bit address for either internal RAM or SFR.) The contents of the RAM address “direct” are not altered.	2	24

MOVC A, @A + DPTR	MOVE CODE byte command. This command will load the accumulator with a byte retrieved from external program memory. The memory location from which the code byte is retrieved is the sum of the accumulator contents and the 16-bit DPTR register contents. (NOTE: The DPS SFR determines which data pointer is currently selected.) The DPTR contents are not altered.	1	24
MOVC A, @A + PC	MOVE CODE byte command. This command will load the accumulator with a byte retrieved from program memory. The memory location from which the code byte is retrieved is the sum of the accumulator contents and the 16-bit program counter (PC) contents. The PC contents are incremented prior to addition to the accumulator contents.	1	24
MOVX @DPTR, A	MOVE EXTERNAL byte command. This command will copy the accumulator contents into the RAM location whose address is contained in the currently selected data pointer register (i.e. write operation). (NOTE: In addition to external memory locations, MOVX commands can also access on-chip XRAM.) (NOTE: The DPS SFR determines which data pointer is currently selected.) Bits 4-3 of the MEMCON SFR determine the memory location to which the data is written (on-chip XRAM is one possible destination). Bit 5 of the MEMCON SFR determines if this MOVX command is time stretched by one machine cycle (12 oscillator cycles).	1	24
MOVX @Ri, A	MOVE EXTERNAL byte command. This command will copy the accumulator contents into the XRAM location formed by taking P2 as the address MSB and the contents of bank register "i" as the address LSB. ("Ri" can be either R0 or R1 of the working register bank.) (NOTE: In addition to external memory locations, MOVX commands can also access on-chip XRAM.) Bits 4-3 of the MEMCON SFR determine the memory location to which the data is written (on-chip XRAM is one possible destination). Bit 5 of the MEMCON SFR determines if this MOVX command is time stretched by one machine cycle (12 oscillator cycles).	1	24

MOVX A, @DPTR	MOVE EXTERNAL byte command. This command will copy the contents of the XRAM location at the address held in DPTR into the accumulator (i.e. read operation). (NOTE: The DPS SFR determines which data pointer is currently selected.) (NOTE: In addition to external memory locations, MOVX commands can also access on-chip XRAM.) Bits 1-0 of the MEMCON SFR determine the memory location from which the data is read. Bit 2 of the MEMCON SFR determines if this MOVX command is time stretched by one machine cycle (12 oscillator cycles). (NOTE: Even though the 16-bit address contained with the DPTR is normally associated with external program memory fetches, the DPTR contents can also refer to on-chip XRAM as well; 13 bits are required to fully access all 8k of the CMX850's on-board XRAM space).	1	24
MOVX A, @Ri	MOVE EXTERNAL byte command. This command will copy the contents of the XRAM location at the address formed by taking P2 as the address MSB and Ri as the address LSB into the accumulator (i.e. read operation). ("Ri" can be either R0 or R1 of the working bank register.) Bits 1-0 of the MEMCON SFR determine the memory location from which the data is read. (NOTE: In addition to external memory locations, MOVX commands can also access on-chip XRAM.) Bit 2 of the MEMCON SFR determines if this MOVX command is time stretched by one machine cycle (12 oscillator cycles).	1	24
MUL AB	MULTIPLY operation. This command multiplies the contents of the accumulator by the contents of the B register to give a 16-bit result. The lower 8 bits of the result will be stored in the accumulator, and the upper 8 bits of the result will be stored in the B register.	1	48
NOP	NO OPERATION command. This command simply transfers program execution to the next step in program memory; only the PC is affected by this command.	1	12
ORL A, #data	LOGICAL OR byte operation. This command performs a bitwise-OR operation between the accumulator value and the immediate data byte, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	2	12

ORL A, @Ri	LOGICAL OR byte operation. This command performs a bitwise-OR operation between the accumulator value and the contents of the RAM location whose address is in bank register “i”, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	1	12
ORL A, direct	LOGICAL OR byte operation. This command performs a bitwise-OR operation between the accumulator value and the contents of RAM address “direct”, with the result stored in the accumulator. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	2	12
ORL A, Rn	LOGICAL OR byte operation. This command performs a bitwise-OR operation between the accumulator value and the value in bank register “n”, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	1	12
ORL C, /bit	LOGICAL OR bit command. This command performs an OR operation on the carry bit of the PSW and the inverse of the indicated bit, with the result stored in the carry bit. The indicated bit (source bit) is not altered.	2	24
ORL C, bit	LOGICAL OR bit command. This command performs an OR operation on the carry bit of the PSW and the indicated bit, with the result stored in the carry bit. The indicated bit (source bit) is not altered.	2	24
ORL direct, #data	LOGICAL OR byte operation. This command performs a bitwise-OR operation between the the contents of RAM address “direct” and the immediate data byte, with the result stored in RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins. This instruction can be used to set combinations of bits in the “direct” RAM register or SFR.	3	24

ORL direct, A	LOGICAL OR byte operation. This command performs a bitwise-OR operation between the the contents of RAM address “direct” and the accumulator value, with the result stored in RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins. This instruction can be used to set combinations of bits in the “direct” RAM register or SFR.	2	12
POP direct	POP command. The contents of the RAM location addressed by the stack pointer are copied into the specified “direct” RAM location. The stack pointer is then decremented by one. (“Direct” is an 8-bit address for either internal RAM or SFR.	2	24
PUSH direct	PUSH command. The stack pointer is incremented by one, and the 8-bit “direct” address is then copied into the RAM location whose address is contained in the stack pointer. (“Direct” is an 8-bit address for either internal RAM or SFR).	2	24
RET	RETURN command. This command returns program execution from a subroutine to the main program. The RET command pops the high and low order bytes of the PC off of the SP, and the SP is also decremented by 2 in the process. Program execution resumes at the resulting PC address.	1	24
RETI	RETURN FROM INTERRUPT command. The RETI command pops the high and low order PC bytes off of the SP and decrements the SP by 2. Program execution resumes at the newly popped PC address. The CMX850 interrupt logic is restored to allow processing of interrupts with the same priority as the interrupt just processed. If an interrupt of equal or lower priority was pending when RETI was performed, an additional instruction will be performed before the pending interrupt is processed. The PSW is not automatically restored to its pre-interrupt condition.	1	24
RL A	ROTATE LEFT command. This command shifts each bit in the accumulator to the left by one bit. Bit 7 is shifted into the Bit 0 position.	1	12
RLC A	ROTATE LEFT THROUGH CARRY command. The carry flag bit is shifted into accumulator Bit 0, and accumulator Bit 7 is shifted to the carry flag position.	1	12

RR A	ROTATE RIGHT command. This command shifts each bit of the accumulator to the right by one position. Bit 0 is shifted into the Bit 7 position.	1	12
RRC A	ROTATE RIGHT THROUGH CARRY command. This command shifts each bit of the accumulator to the right by one position. The carry flag bit is shifted into accumulator Bit 7, and accumulator Bit 0 is shifted to the carry flag position.	1	12
SETB bit	SET bit command. This command sets the indicated bit to one. No other bits are altered.	2	12
SETB C	SET bit command. This command sets the carry bit to one. No other bits are altered.	1	12
SJMP rel	SHORT JUMP command. This command increments the PC by 2 and unconditionally branches to an address equal to the sum of the "rel" and the PC (after incrementing).	2	24
SUBB A, #data	SUBTRACT WITH BORROW operation. This command subtracts the immediate data byte and the carry flag from the contents of the accumulator, then leaves the result in the accumulator. If the state of the carry flag is unknown prior to performing a subtraction, clear the carry flag with a CLR C instruction before performing the subtraction.	2	12
SUBB A, @Ri	SUBTRACT WITH BORROW operation. This command subtracts the contents of the RAM location whose address is in bank register "i" and the carry flag from the contents of the accumulator, then leaves the result in the accumulator. If the state of the carry flag is unknown prior to performing a subtraction, clear the carry flag with a CLR C instruction before performing the subtraction.	1	12
SUBB A, direct	SUBTRACT WITH BORROW operation. This command subtracts the contents of RAM address "direct" and the carry flag from the contents of the accumulator then leaves the result in the accumulator. If the state of the carry flag is unknown prior to performing a subtraction, clear the carry flag with a CLR C instruction before performing the subtraction. ("Direct" is an 8-bit address for either internal RAM or SFR).	2	12

SUBB A, Rn	SUBTRACT WITH BORROW operation. This command subtracts the byte contained in bank register “n” and the carry flag from the contents of the accumulator, then leaves the result in the accumulator. If the state of the carry flag is unknown prior to performing a subtraction, clear the carry flag with a CLR C instruction before performing the subtraction.	1	12
SWAP A	SWAP command. This command swaps the low-order and high-order nibbles of the accumulator.	1	12
XCH A, @Ri	EXCHANGE command. This command copies the contents of the RAM location whose address is stored in bank register “i” into the accumulator, while copying the contents of the accumulator into the RAM location whose address is stored in bank register “i”.	1	12
XCH A, direct	EXCHANGE command. This command copies the contents of the 8-bit “direct” RAM address into the accumulator while copying the contents of the accumulator into the 8-bit “direct” RAM address. (“Direct” is an 8-bit address for either internal RAM or SFR.)	2	12
XCH A, Rn	EXCHANGE command. This command copies the contents of bank register “n” into the accumulator while copying the contents of the accumulator into bank register “n”.	1	12
XCHD A, @Ri	EXCHANGE DIGIT command. This command will exchange bits 3-0 of the accumulator with bits 3-0 of the RAM location whose address is stored in bank register “i”. The upper 4 bits of both registers are not affected by this command.	1	12
XRL A, #data	LOGICAL EXCLUSIVE-OR byte operation. This command performs a bitwise-XOR operation between the accumulator value and the immediate data byte, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	2	12
XRL A, @Ri	LOGICAL EXCLUSIVE-OR byte operation. This command performs a bitwise-XOR operation between the accumulator value and the contents of the RAM location whose address is in bank register “i”, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	1	12

XRL A, direct	LOGICAL EXCLUSIVE-OR byte operation. This command performs a bitwise-XOR operation between the accumulator value and contents of RAM address “direct”, with the result stored in the accumulator. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	2	12
XRL A, Rn	LOGICAL EXCLUSIVE-OR byte operation. This command performs a bitwise-XOR operation between the accumulator value and the value in bank register “n”, with the result stored in the accumulator. When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins.	1	12
XRL direct, #data	LOGICAL EXCLUSIVE-OR byte operation. This command performs a bitwise-XOR operation between the the contents of RAM address “direct” and the immediate data byte, with the result stored in RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins. This instruction can be used to complement combinations of bits in the “direct” RAM register or SFR.	3	24
XRL direct, A	LOGICAL EXCLUSIVE-OR byte operation. This command performs a bitwise-XOR operation between the contents of RAM address “direct” and the accumulator value, with the result stored in RAM address “direct”. (“Direct” is an 8-bit address for either internal RAM or SFR.) When this instruction is used to modify an output port, the output port latches supply the original port data (for the instruction), not the port pins. This instruction can be used to complement combinations of bits in the “direct” RAM register or SFR.	2	12

Table 18: CMX850 Instruction Set

Addressing Modes

The CMX850 has five addressing modes: direct, indirect, register, immediate, and indexed addressing.

Direct Addressing

The direct addressing mode uses an 8-bit opcode followed by an 8-bit address for either internal RAM (first 128 bytes) or SFR. The address can correspond to a byte or a bit in a bit-addressable byte.

For example, if RAM address 21h contains A0h, then the command:

```
MOV A, 21;
```

…will copy A0h into the accumulator.

(Note that internal XRAM is accessed through the Ri + R2 or 16-bit data pointer.)

Indirect Addressing

With indirect addressing, the instruction specifies the address of the register that contains the actual operand. The @ symbol is used to indicate indirect addressing.

For example, if bank register R1 contains 21h, and if RAM address 21h contains the value FFh, the command:

```
MOV A, @R1;
```

…will cause FFh to be copied into the accumulator.

Indirect addressing can access internal and external RAM, and the addresses can be either 8-bits or 16-bits long. An 8-bit indirect address can refer to the Stack Pointer (SP) or to either R0 or R1 of the currently selected working bank register, which is selected with bits 4-3 of the PSW SFR. A 16-bit indirect address corresponds to the currently selected data pointer register or Ri + R2. (The CMX850 has two data pointer registers, and bit 0 of the DPS SFR determines which data pointer is currently selected.)

The CMX850 internal XRAM is accessed with indirect addressing via the MOVX command. Bits 4-3 (write destination) and bits 1-0 (read source) of the MEMCON SFR register determine whether or not the internal XRAM is accessed with the MOVX command.

Register Addressing

Commands that use register addressing use the contents of one of the working registers R0-R7 as the operand. For example, if register R0 contains C0h, then the command:

```
MOV A, R0;
```

…will copy C0h into the accumulator.

Immediate Addressing

With immediate addressing, the value to be executed is passed directly as part of the instruction. The operand can either be a numeric constant or its symbolic name and can be either an 8-bit or 16-bit number. The # symbol is used to indicate immediate addressing.

For example, the following command:

```
ADD A, #83;
```

…will add 83 to the contents of the accumulator, with the result left in the accumulator.

Indexed Addressing

Indexed addressing is used for external program memory accesses and for performing jumps to subroutines. With indexed addressing, a 16-bit base value (e.g. DPTR or PC) value is added to an 8-bit value (i.e. accumulator) to arrive at the desired memory address.

For example, the following command:

```
MOVC A, @A + DPTR
```

…will move (copy) a code byte relative to the DPTR plus the accumulator to the accumulator.

Conclusion

An enhanced 8051 Microcontroller and integrated V.22bis modem combine to make the CMX850 a versatile platform for many types of telecom applications. The information in this application note should provide clarification on the many features available in the CMX850 and assist design engineers as they integrate the CMX850 into their projects.

Appendix

RTC flow of operation

```
; Routine to configure the RTC for use

RTC_SETUP:

; Enable the RTC circuit (bit 7 set)
; Stop the RTC counter incrementing (bit 3 set)
; Set RTC time interrupt interval to 1/second (bits 0-2 set to 0x04)

        MOV     RTCCON,#0x8C


; Setup time registers with start value (0x00000000)


        CLR     A
        MOV     TIME0,A
        MOV     TIME1,A
        MOV     TIME2,A
        MOV     TIME3,A

; Setup alarm registers with start value (0x00000004)


        MOV     ALM0,#0x04
        MOV     ALM1,A
        MOV     ALM2,A
        MOV     ALM3,A

; Time interval interrupt level triggered (INT6, ICON1B bit 0 clear)
; Alarm interrupt level triggered (INT7, ICON1B bit 2 clear)

        ANL     ICON1B,#0xFA


; Enable time interval interrupt (INT6, IE_1 bit 4 set)
; Enable alarm interrupt (INT7, IE_1 bit 5 set)

        ORL     IE_1,#0x30

; Start the RTC counter incrementing (bit 3 set)

        ANL     RTCCON,#0xF7
        RET

;RCT time interval interrupt
;Interrupt is generated once every programmed time interval (once per second ;in this
example)

RTC_INT6:

; Clear time interval interrupt (RTCCON, bit 5 set)

        ORL     RTCCON,#0x20

; Perform any other required actions here
; ...
```

```
        RETI


; Alarm interrupt
; Interrupt generated once time register value is greater than or equal
; to the alarm register value (four seconds in this example)


RTC_INT7:
   PUSH    ACC             ; Save accumulator contents

; RTC alarm register still set to 0x00000004 ?

   MOV     A,#0x04
   CJNE    A,ALM0,SECOND_PASS

FIRST_PASS:

   ; This is the first alarm interrupt
   ; Change the alarm register value to 0x00000006

   MOV     ALM0,#0x06
   JMP     RTC_INT7_EXIT

SECOND_PASS:

   ; Disable alarm interrupt (INT7, IE_1 bit 5 clear)
   ; Stops alarm interrupts from re-occurring once interrupt service
   ; routine completes

   ANL     IE_1,#0xDF

   ; Perform any other required actions here
   ; ...

RTC_INT7_EXIT:
   POP     ACC             ; Restore accumulator contents
   RETI

NOTE:

In this example the interrupt sequence timeline would be as follows

0 second:   Routine RTC_SETUP run
1 second:   Time interval interrupt (INT6)
2 second:   Time interval interrupt (INT6)
3 second:   Time interval interrupt (INT6)
4 second:   Time interval interrupt (INT6)
        Alarm interval (INT7), First pass, alarm set to six seconds
5 second:   Time interval interrupt (INT6)
6 second:   Time interval interrupt (INT6)
        Alarm interval (INT7), Second pass, alarm disabled
7 second:   Time interval interrupt (INT6)
8 second:   Time interval interrupt (INT6)
9 second:   Time interval interrupt (INT6)
...
```

References

1) CMX850 Data Sheet, CML Microcircuits, http://www.cmlmicro.com/products/datasheets/Docs/cmx850ds.PDF
2) EV8500 User Manual, CML Microcircuits, http://www.cmlmicro.com/products/datasheets/Docs/ev8500ds.PDF
3) James W. Stewart and Kai X. Miao, The 8051 Microcontroller; Hardware, Software and Interfacing, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, 1999.
4) Myke Predko, Handbook of Microcontrollers, McGraw-Hill, New York, 1999.
5) John Wharton, An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family, Intel Application Note AP-69, Intel Corporation, 1980.
6) MCS®51 Microcontroller Family User's Manual, Intel Corporation, 1994.
7) Keil Software – Embedded Development Software - http://www.keil.com

www.cmlmicro.com

For FAQs see: http://www.cmlmicro.com/products/faqs/index.htm

For a full data sheet listing see:
http://www.cmlmicro.com/products/datasheets/download.htm

For detailed application notes:
http://www.cmlmicro.com/products/applications/index.htm

Oval Park, Langford, Maldon, Essex, CM9 6WG - England. Tel: +44 (0)1621 875500 Fax: +44 (0)1621 875600 Sales: sales@cmlmicro.com Technical Support: techsupport@cmlmicro.com	4800 Bethania Station Road, Winston-Salem, NC 27105 - USA. Tel: +1 336 744 5050, 800 638 5577 Fax: +1 336 744 5054 Sales: us.sales@cmlmicro.com Technical Support: us.techsupport@cmlmicro.com	No 2 Kallang Pudding Road, 09 to 05/06 Mactech Industrial Building, Singapore 349307 Tel: +65 7450426 Fax: +65 7452917 Sales: sg.sales@cmlmicro.com Technical Support: sg.techsupport@cmlmicro.com	No. 218, Tian Mu Road West, Tower 1, Unit 1008, Shanghai Kerry Everbright City, Zhabei, Shanghai 200070, China. Tel: +86 21 63174107 +86 21 63178916 Fax: +86 21 63170243 Sales: cn.sales@cmlmicro.com.cn Technical Support: sg.techsupport@cmlmicro.com