Versa Mix 8051 Mixed-Signal MCU

Overview

The VMX51C1016 is a fully integrated mixed-signal microcontroller that provides a “one-chip” solution for a broad range of control, data acquisition and processing applications. The VXM1016 is based on a powerful, single-cycle, RISC-based 8051 processor with an enhanced MULT/ACCU unit that can be used to perform complex mathematical operations. The device includes 56KB of Flash memory and 1280 bytes of RAM.

On-chip analog peripherals such as: an A/D converter, PWM outputs (that can be used as D/A converters), a voltage reference and an analog switch make the VMX51C1020 ideal for analog data acquisition applications.

The inclusion of a full set of digital interfaces such as an enhanced, fully configurable SPI, an I²C, UARTs and a J1708/RS-485 compatible differential transceiver, enables total system integration.

The VMX51C1016 operates on a 5 volt supply and is available in a QFP-44 package.

Applications

- Automotive Applications
- Industrial Controls / Instrumentation
- Consumer Products
- Medical Devices

Feature Set

- 8051 Compatible RISC Performance Processor
  Enhanced Arithmetic Unit including Barrel Shifter
- 56KB of Flash Memory
- 1280 Bytes of RAM
- 24 General Purpose I/Os
- 2 UART Serial Ports
- 2 Baud Rate Generators for UARTs
- Diff. Transceiver connected to UART1
  J1708/RS-485 compatible)
- Enhanced SPI interface (Master/Slave)
  - Fully configurable
  - Control up to 4 Slave devices
- I²C interface
- 1 External Dedicated Interrupt Input
- Interrupt on Port 1.0-P1.3 Pin Change
- 3, 16-bit Timers/Counters
- 4 Compare Units and 2 Capture Inputs
- 4 Ch. PWM, 8-bit / 16-bit resolution
- 5 Ch. 12-bit A/D Converter
  - Conversion rate configurable up to 10KHz
  - Continuous/One-Shot Operation
  - Single or 4-Channel Automatic Sequential Conversions
- On-Chip Voltage Reference
- Digitally Controlled Switch
- Power Saving Features + Clock Control
- Watchdog Timer
- Operating Temperature range (0ºC to +70ºC)
- Available in QFP-44 package

FIGURE 1: VMX51C1016 BLOCK DIAGRAM

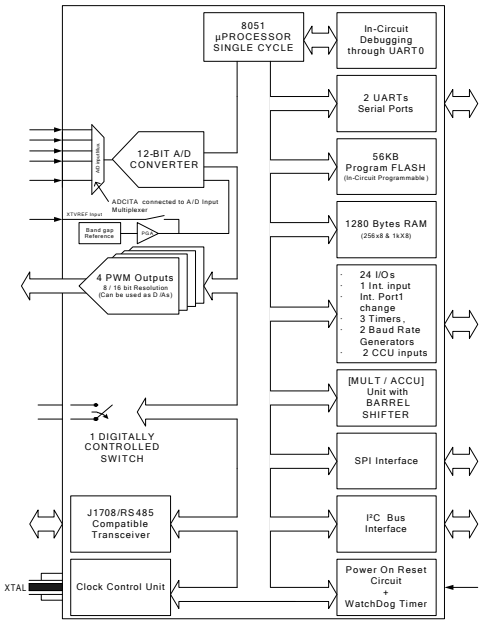


FIGURE 2: VMX51C1016-QAC14, QFP-44 PACKAGE PINOUT

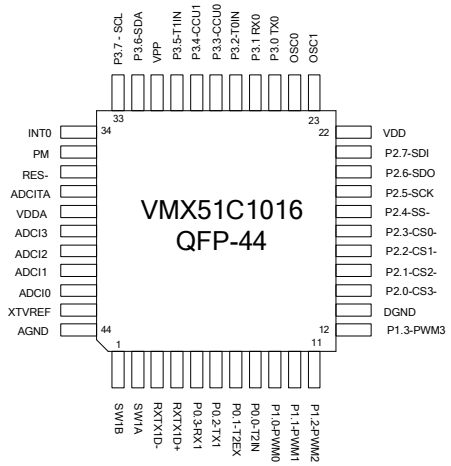
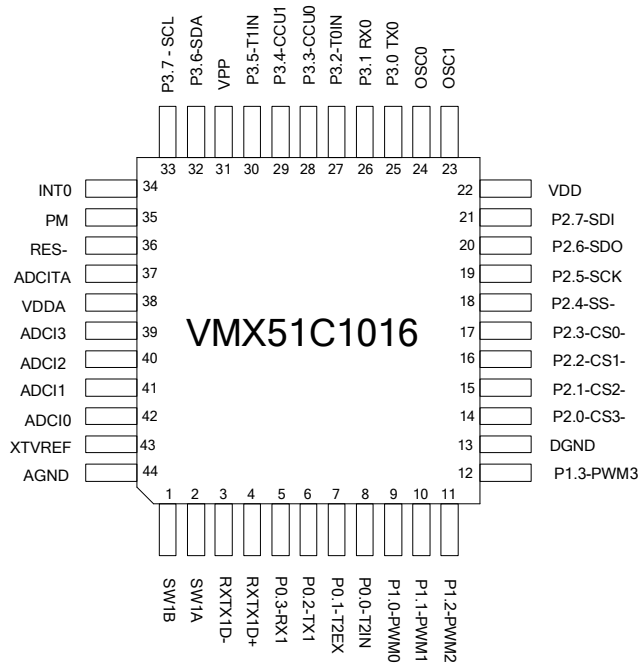


Table 1: Pinout description

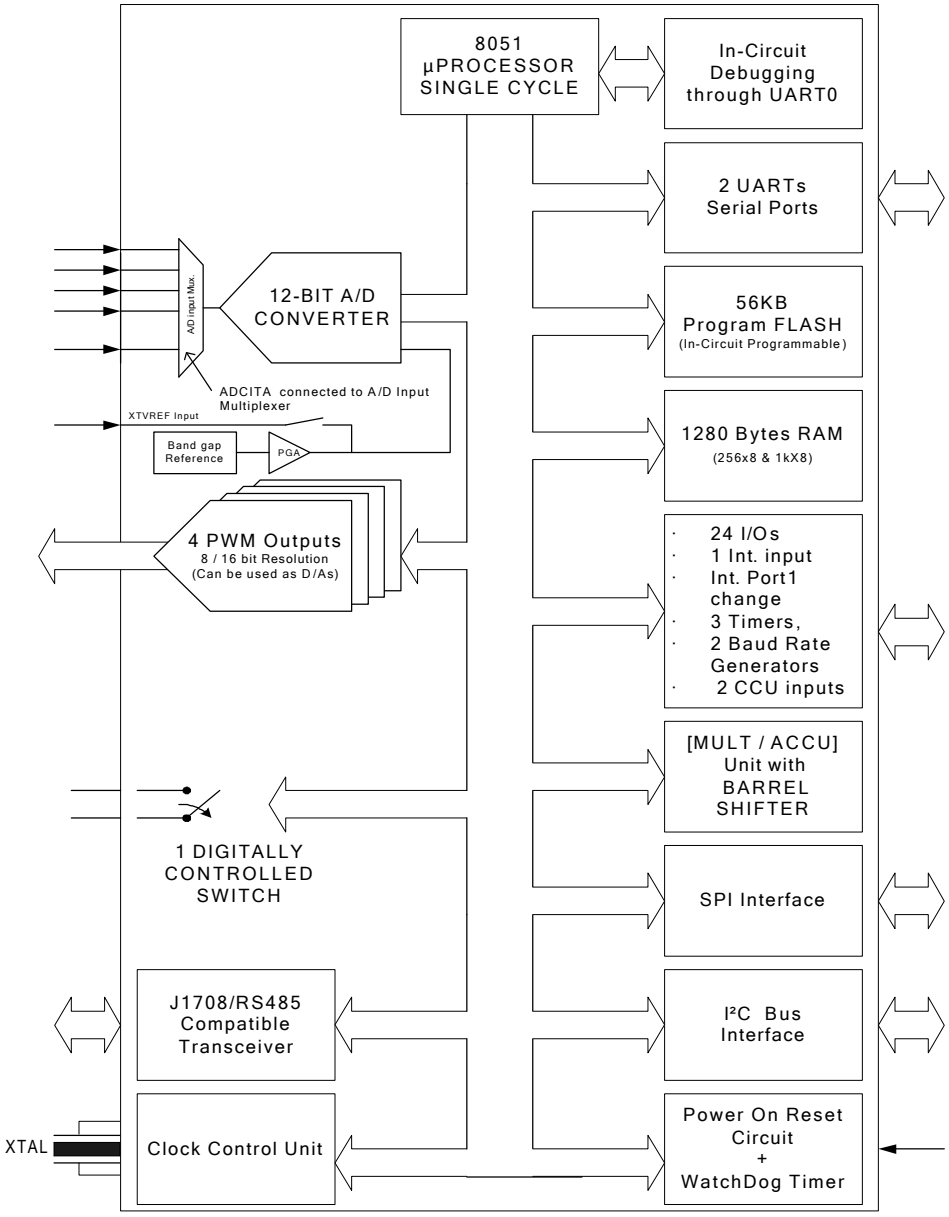
PIN	NAME	FUNCTION
1	SW1A	Digitally Controlled Switch 1A
2	SW1B	Digitally Controlled Switch 1B
3	RXTX1D-	RS-485 Compatible Differential Transmitter/Receiver, Negative side
4	RX1TXD+	RS-485 Compatible Differential Transmitter/Receiver, Positive side
5	P0.3-RX1	I/O - Asynchronous UART1 Receiver Input
6	P0.2-TX1	I/O - Asynchronous UART1 Transmitter Output
7	P0.1-T2EX	I/O -Timer/Counter 2 Input
8	P0.0-T2IN	I/O -Timer/Counter 2 Input
9	P1.0-PWM0	I/O - Pulse Width Modulator output 0
10	P1.1-PWM1	I/O - Pulse Width Modulator output 1
11	P1.2-PWM2	I/O - Pulse Width Modulator output 2
12	P1.3-PWM3	I/O - Pulse Width Modulator output 3
13	DGND	Digital Ground
14	P2.0-CS3-	I/O - SPI Chip Enable Output (Master Mode)
15	P2.1-CS2-	I/O - SPI Chip Enable Output (Master Mode)
16	P2.2-CS1-	I/O - SPI Chip Enable Output (Master Mode)
17	P2.3-CS0-	I/O - SPI Chip Enable Output (Master Mode)
18	P2.4-SS-	I/O - SPI Chip Enable Output (Slave Mode)
19	P2.5-SCK	I/O - SPI Clock (Input in Slave Mode)
20	P2.6-SDO	I/O - SPI Data Output Bus
21	P2.7-SDI	I/O - SPI Data Input Bus
22	VDD	Digital Supply
23	OSC1	Oscillator Crystal Output
24	OSC0	Oscillator Crystal input/External Clock Source Input
25	P3.0-TX0	I/O - Asynchronous UART0 Transmitter Output
26	P3.1-RX0	I/O - Asynchronous UART0 Receiver Input
27	P3.2-T0IN	I/O - Timer/Counter 0 Input
28	P3.3-CCU0	I/O - Capture and Compare Unit 0 Input
29	P3.4-CCU1	I/O - Capture and Compare Unit 1 Input
30	P3.5-T1IN	I/O - Timer/Counter 1 Input
31	VPP	Flash Programming Voltage Input
32	P3.6-SDA	I/O - I2C / Prog. Interface Bi-Directional Data Bus
33	P3.7-SCL	I/O - I2C / Prog. Interface Clock
34	INT0	External interrupt Input (Negative Level or Edge Triggered)
35	PM	Mode Control Input
36	RES-	Hardware Reset Input (Active low)
37	ADCITA	ADC input 5 and Analog Output
38	VDDA	Analog Supply
39	ADCI3	Analog to Digital Converter ext. Input 3
40	ADCI2	Analog to Digital Converter ext. Input 2
41	ADCI1	Analog to Digital Converter ext. Input 1
42	ADCI0	Analog to Digital Converter ext. Input 0
43	XTVREF	External Reference Voltage Input
44	AGND	Analog Ground

FIGURE 3: VMX51C1016 PINOUT



VMX51C1016 Block Diagram

FIGURE 4: VMX51C1016 BLOCK DIAGRAM



Absolute Maximum Ratings

V_{DD} to DGND	–0.3V, +6V	Digital Output Voltage to DGND	–0.3V, V_{DD}+0.3V
V_{DDA} to DGND	-0.3V, +6V	V_{PP} to DGND	+13V
AGND to DGND	–0.3V, +0.3V	Power Dissipation	
V_{DD} to V_{DDA}	-0.3V, +0.3V	▪ To +75°C	1000mW
ADCI (0-3) to AGND	-0.3V, V_{DDA}+0.3V	▪ Derate above +75°C	10mW/°C
XTVREF to AGND	-0.3V, V_{DDA}+0.3V	Operating Temperature Range	0° to +70°C
Digital Input Voltage to DGND	-0.3V, V_{DD}+0.3V	Storage Temperature Range	–65°C to +150°C
RS485 pin Minimum and Maximum Voltages	-2V, +7V	Lead Temperature (soldering, 10sec)	+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only. The functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Electrical Characteristics

TABLE 2: ELECTRICAL CHARACTERISTICS

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
GENERAL CHARACTERISTICS (V_{DD} = +5V, V_{DDA} = +5V, T_A = +25°C, 14.75MHz input clock, unless otherwise noted.)						
Power Supply Voltage	V_{DD}		4.75	5.0	5.5	V
	V_{DDA}		4.5	5.0	5.5	V
Power Supply Current	$I_{DD\ (14MHz)}$		5		45*	mA
	$I_{DD\ (1MHz)}$		0.6		6*	*Depends on clock speed and peripheral use and load
	I_{DDA}		0.1		5*	
Flash Programming Voltage	V_{PP}		11		13	V
DIGITAL INPUTS						
Minimum High-Level input	V_{IH}	V_{DD} = +5V		2.0		V
Maximum Low-Level input	V_{IL}	V_{DD} = +5V		0.8		V
Input Current	I_{IN}			±0.05		µA
Input Capacitance	C_{IN}			5	10	pF
DIGITAL OUTPUTS						
Minimum High-Level Output Voltage	V_{OH}	I_{SOURCE} = 4mA		4.2		V
Maximum Low-Level Output Voltage	V_{OL}	I_{SINK} = 4mA		0.2		V
Output Capacitance	C_{OUT}			10	15	pF
Tri-state Output Leakage Current	I_{OZ}				0.25	µA

ANALOG INPUTS					
ADCI(0-3) Input Voltage Range	V_{ADCI}		0	2.7	V
ADCI(0-3) Input Resistance	R_{ADCI}		100		Mohms (design)
ADCI(0-3) Input Capacitance	C_{ADCI}		7		pF
ADCI(0-3) Input Leakage Current	I_{ADCI}		TBD		nA
Channel-to-Channel Crosstalk				-72 (12 bit)	dB (design)
INTERNAL REFERENCE					
Bandgap Reference Voltage			1.18	1.23V 1.28	V
Bandgap Reference Tempco			100		ppm/°C
EXTERNAL REFERENCE					
Input Impedance	R_{XTVREF}		150		kOhms
PGA					
PGA Gain adjustment			2.11	2.29	
ANALOG TO DIGITAL CONVERTER **External Reference, TA=25C, Fosc = 16MHz**					
ADC Resolution			12		Bits
Differential Non linearity	DNL			±1.5	LSB
Integral Non linearity	INL		-1	+4	LSB
Full-Scale Error (Gain Error)		All channels, ADCI(0-3)	±4		LSB
Offset Error		All channels, ADCI(0-3)	±1		LSB
Channel-to-Channel Mismatch		All channels, ADCI(0-3)	±1		LSB
Sampling Rate		Single Channel	1	10k	Hz
		4 Channels	1	2.5k	
UART1 DIFFERENTIAL TRANSCEIVER COMPATIBLE TO J1708/ RS-485					
Common mode Input Voltage	Vc_I		-2	+7	V
Input Impedance	Z_{IN}		1		MOhms
Output Drive Current			30		mA
Differential Input			100mV		mV
DIGITAL SWITCH					
Switch on Resistance			50	100	Ohms (+/-10%)
Input capacitance			4		pF
Voltage range on Pin			0	5	V
Allowable current (DC)				5	mA
BROWN OUT / RESET CIRCUIT					
Brown-out circuit Threshold			3.7	4.0	V
RES- pin internal Pull-Up			20		KOhms

Detailed Description

The following sections describe the VMX51C1016 architecture and peripherals.

FIGURE 5: INTERFACE DIAGRAM FOR THE VERSA MIX

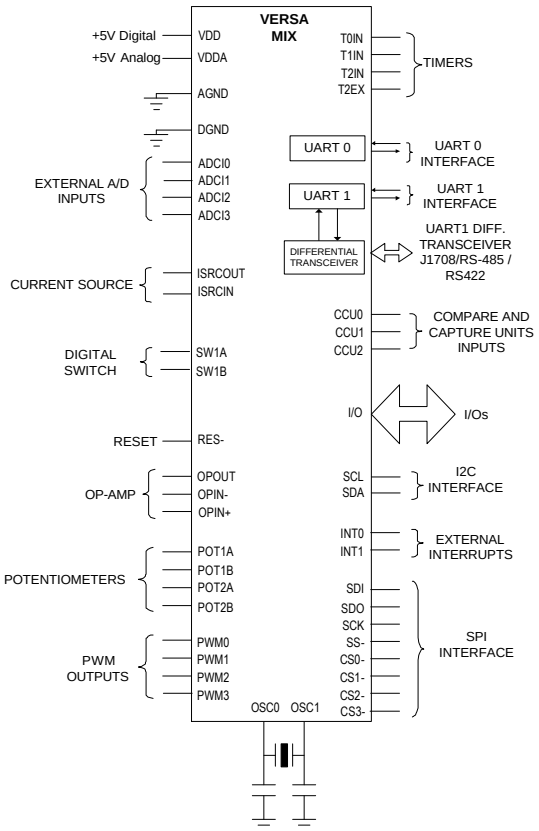
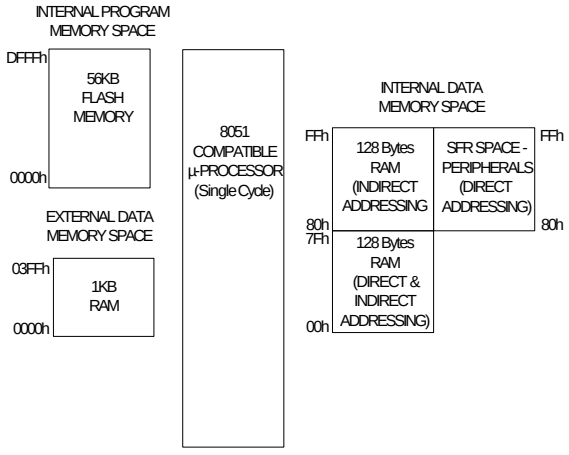


FIGURE 6: MEMORY ORGANIZATION OF THE VERSA MIX



Memory Organization

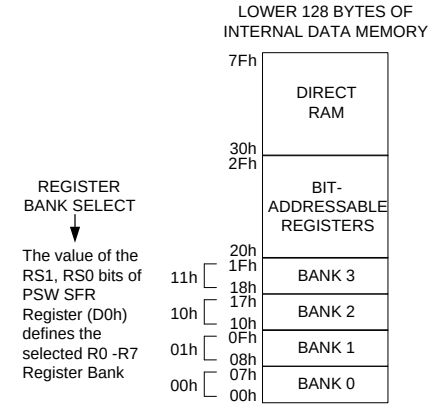
Figure 6 shows the memory organization of the VMX51C1016.

At power-up/reset, code is executed from the 56Kx8 Flash memory mapped into the processor's internal program space.

A 1KB block of RAM is also mapped into the external data memory of the VMX51C1016. This block can be used as a general-purpose scratch pad or storage memory. A 256 byte block of RAM is mapped to the internal data memory space. This block of RAM is broken into two sub-blocks, with the upper block accessible via indirect addressing only and the lower block accessible via both direct and indirect addressing.

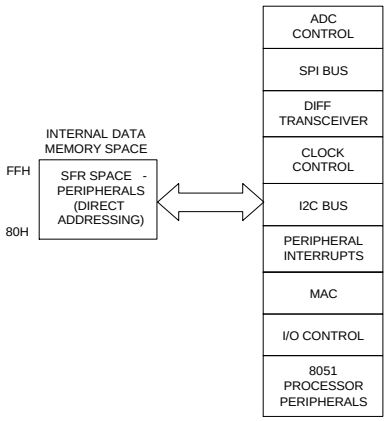
The following figure describes the access to the lower block of 128 bytes.

FIGURE 7: LOWER 128 BYTES BLOCK INTERNAL MEMORY MAP



The SFR (special function register) space is also mapped into the upper 128 bytes of internal data memory space. This SFR space is only accessible using direct-access. The SFR space provides the interface to all the on-chip peripherals. This interfacing is illustrated in Figure 8.

FIGURE 8: SFR ORGANIZATION



Dual Data Pointers

The VMX51C1016 includes two data pointers.

The first data pointer (DPTR0) is mapped into SFR locations 82h and 83h. The second data pointer (DPTR1) is mapped into SFR locations 84h and 85h. The SEL bit in the data pointer select register, DPS (SFR 86h), selects which data pointer is active. When SEL = 0, instructions that use the data pointer will use DPL0 and DPH0. When SEL = 1, instructions that use the DPTR will use DPL1 and DPH1. SEL is located in bit 0 of the DPS (SFR location 86h). The remaining bits of SFR location 86h are unused.

All DPTR-related instructions use the currently selected data pointer. In order to switch the active pointer, toggle the SEL bit. The fastest way to do this is to use the increment instruction (INC DPS).

The use of the two data pointers can significantly increase the speed of moving large blocks of data because only one instruction is needed to switch from a source address and destination address.

The SFR locations and register representations related to the dual data pointers are outlined as follows:

TABLE 3: (DPH0) DATA POINTER HIGH 0 - SFR 83H

15	14	13	12	11	10	9	8
DPH0 [7:0]							

TABLE 4: (DPL0) DATA POINTER LOW 0 - SFR 82H

7	6	5	4	3	2	1	0
DPL0 [7:0]							

Bit	Mnemonic	Function
15-8	DPH0	Data Pointer 0 MSB
7-0	DPL0	Data Pointer LSB.

TABLE 5: (DPH1) DATA POINTER HIGH 1 - SFR 85H

15	14	13	12	11	10	9	8
DPH1 [7:0]							

TABLE 6: (DPL1) DATA POINTER LOW 1 - SFR 84H

7	6	5	4	3	2	1	0
DPL1 [7:0]							

Bit	Mnemonic	Function
15-8	DPH1	Data Pointer 1 MSB.
7-0	DPL1	Data Pointer 1 LSB.

TABLE 7: (DPS) DATA POINTER SELECT REGISTER - SFR 86H

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	SEL

Bit	Mnemonic	Function
7-1	0	Always zero
0	SEL	0 = DPTR0 is selected 1 = DPTR1 is selected Used to toggle between both data pointers

MPAGE Register

The MPAGE register controls the upper 8 bits of the targeted address when the MOVX instruction is used for external RAM data transfer. This allows access to the entire external RAM content without using the data pointer.

TABLE 8: (MPAGE) MEMORY PAGE - SFR CFH

7	6	5	4	3	2	1	0
MPAGE [7:0]							

User Flags

The VMX51C1016 provides an SFR register that allows the user to define software flags. Each bit of this register is individually addressable. This register may also be used as a general-purpose storage location. Thus, the user flag feature allows the VMX51C1016 to better adapt to each specific application. This register is located at SFR address F8h.

TABLE 9: (USERFLAGS) USER FLAG - SFR F8H

7	6	5	4	3	2	1	0
UF7	UF6	UF5	UF4	UF3	UF2	UF1	UF0

Instruction Set

All VMX51C1016 instructions are function and binary code compatible with industry standard 8051s. However, the timing of instruction sets may be different. The following two tables describe the VMX51C1016 instruction set.

TABLE 10: LEGEND FOR INSTRUCTION SET TABLE

Symbol	Function
A	Accumulator
Rn	Register R0-R7
Direct	Internal register address
@Ri	Internal register pointed to by R0 or R1 (except MOVX)
rel	Two's complement offset byte
bit	Direct bit address
#data	8-bit constant
#data 16	16-bit constant
addr 16	16-bit destination address
addr 11	11-bit destination address

TABLE 11: VERSA MIX INSTRUCTION SET

Mnemonic	Description	Size (bytes)	Instr. Cycles
Arithmetic instructions			
ADD A, Rn	Add register to A	1	1
ADD A, direct	Add direct byte to A	2	2
ADD A, @Ri	Add data memory to A	1	2
ADD A, #data	Add immediate to A	2	2
ADDC A, Rn	Add register to A with carry	1	1
ADDC A, direct	Add direct byte to A with carry	2	2
ADDC A, @Ri	Add data memory to A with carry	1	2
ADDC A, #data	Add immediate to A with carry	2	2
SUBB A, Rn	Subtract register from A with borrow	1	1
SUBB A, direct	Subtract direct byte from A with borrow	2	2
SUBB A, @Ri	Subtract data mem from A with borrow	1	2
SUBB A, #data	Subtract immediate from A with borrow	2	2
INC A	Increment A	1	1
INC Rn	Increment register	1	2
INC direct	Increment direct byte	2	3
INC @Ri	Increment data memory	1	3
DEC A	Decrement A	1	1
DEC Rn	Decrement register	1	2
DEC direct	Decrement direct byte	2	3
DEC @Ri	Decrement data memory	1	3
INC DPTR	Increment data pointer	1	1
MUL AB	Multiply A by B	1	5
DIV AB	Divide A by B	1	5
DA A	Decimal adjust A	1	1
Logical Instructions			
ANL A, Rn	AND register to A	1	1
ANL A, direct	AND direct byte to A	2	2
ANL A, @Ri	AND data memory to A	1	2
ANL A, #data	AND immediate to A	2	2
ANL direct, A	AND A to direct byte	2	3
ANL direct, #data	AND immediate data to direct byte	3	4
ORL A, Rn	OR register to A	1	1
ORL A, direct	OR direct byte to A	2	2
ORL A, @Ri	OR data memory to A	1	2
ORL A, #data	OR immediate to A	2	2
ORL direct, A	OR A to direct byte	2	3
ORL direct, #data	OR immediate data to direct byte	3	4
XRL A, Rn	Exclusive-OR register to A	1	1
XRL A, direct	Exclusive-OR direct byte to A	2	2
XRL A, @Ri	Exclusive-OR data memory to A	1	2
XRL A, #data	Exclusive-OR immediate to A	2	2
XRL direct, A	Exclusive-OR A to direct byte	2	3
XRL direct, #data	Exclusive-OR immediate to direct byte	3	4
CLR A	Clear A	1	1
CPL A	Compliment A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Data Transfer Instructions			
MOV A, Rn	Move register to A	1	1
MOV A, direct	Move direct byte to A	2	2
MOV A, @Ri	Move data memory to A	1	2
MOV A, #data	Move immediate to A	2	2
MOV Rn, A	Move A to register	1	2
MOV Rn, direct	Move direct byte to register	2	4
MOV Rn, #data	Move immediate to register	2	2
MOV direct, A	Move A to direct byte	2	3
MOV direct, Rn	Move register to direct byte	2	3
MOV direct, direct	Move direct byte to direct byte	3	4
MOV direct, @Ri	Move data memory to direct byte	2	4
MOV direct, #data	Move immediate to direct byte	3	3
MOV @Ri, A	Move A to data memory	1	3
MOV @Ri, direct	Move direct byte to data memory	2	5
MOV @Ri, #data	Move immediate to data memory	2	3
MOV DPTR, #data16	Move immediate 16 bit to data pointer	3	3
MOVC A, @A+DPTR	Move code byte relative DPTR to A	1	3
MOVC A, @A+PC	Move code byte relative PC to A	1	3
MOVX A, @Ri	Move external data (A8) to A	1	3-10
MOVX A, @DPTR	Move external data (A16) to A	1	3-10
MOVX @Ri, A	Move A to external data (A8)	1	4-11
MOVX @DPTR, A	Move A to external data (A16)	1	4-11
PUSH direct	Push direct byte onto stack	2	4
POP direct	Pop direct byte from stack	2	3
XCH A, Rn	Exchange A and register	1	2
XCH A, direct	Exchange A and direct byte	2	3
XCH A, @Ri	Exchange A and data memory	1	3
XCHD A, @Ri	Exchange A and data memory nibble	1	3
Branching Instructions			
ACALL addr 11	Absolute call to subroutine	2	6
LCALL addr 16	Long call to subroutine	3	6
RET	Return from subroutine	1	4
RETI	Return from interrupt	1	4
AJMP addr 11	Absolute jump unconditional	2	3
LJMP addr 16	Long jump unconditional	3	4
SJMP rel	Short jump (relative address)	2	3
JC rel	Jump on carry = 1	2	3
JNC rel	Jump on carry = 0	2	3
JB bit, rel	Jump on direct bit = 1	3	4
JNB bit, rel	Jump on direct bit = 0	3	4
JBC bit, rel	Jump on direct bit = 1 and clear	3	4
JMP @A+DPTR	Jump indirect relative DPTR	1	2
JZ rel	Jump on accumulator = 0	2	3
JNZ rel	Jump when accumulator not equal to 0	2	3
CJNE A, direct, rel	Compare A, direct JNE relative	3	4
CJNE A, #data, rel	Compare A, immediate JNE relative	3	4
CJNE Rn, #data, rel	Compare reg, immediate JNE relative	3	4
CJNE @Ri, #data, rel	Compare ind, immediate JNE relative	3	4
DJNZ Rn, rel	Decrement register, JNZ relative	2	3
DJNZ direct, rel	Decrement direct byte, JNZ relative	3	4
Bit Operations			
CLR C	Clear carry flag	1	1
CLR bit	Clear direct bit	2	3
SETB C	Set carry flag	1	1
SETB bit	Set direct bit	2	3
CPL C	Complement carry Flag	1	1
CPL bit	Complement direct bit	2	3
ANL C,bit	Logical AND direct bit to carry flag	2	2
ANL C, /bit	Logical AND between /bit and carry flag	2	2
ORL C,bit	Logical OR bit to carry flag	2	2
ORL C, /bit	Logical OR /bit to carry flag	2	2
MOC c,bit	Copy direct bit location to carry flag	2	2
MOV bit,C	Copy carry flag to direct bit location	2	3
Miscellaneous Instruction			
NOP	No operation	1	1

Special Function Registers

The special function registers (SFRs) control several features on the VMX51C1016. Many of the device's SFRs are identical to the standard 8051 SFRs. However, there are additional SFRs that control the VMX51C1016's specific peripheral features that are not available on the standard 8051.

TABLE 12: SPECIAL FUNCTION REGISTERS

SFR Register	SFR Adrs	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Value
P0	80h	-	-	-	-	-	-	-	-	1111 1111b
SP	81h	-	-	-	-	-	-	-	-	0000 0111b
DPL0	82h	-	-	-	-	-	-	-	-	0000 0000b
DPH0	83h	-	-	-	-	-	-	-	-	0000 0000b
DPL1	84h	-	-	-	-	-	-	-	-	0000 0000b
DPH1	85h	-	-	-	-	-	-	-	-	0000 0000b
DPS	86h	0	0	0	0	0	0	0	SEL	0000 0000b
PCON	87h	SMOD	-	-	-	GF1	GF0	STOP	IDLE	0000 0000b
TCON*	88h	TF1	TR1	TF0	TR0	-	-	IE0	IT0	0000 0000b
TMOD	89h	-	CT1	M11	M01	GATE0	CT0	M10	M00	0000 0000b
TL0	8Ah	-	-	-	-	-	-	-	-	0000 0000b
TL1	8Bh	-	-	-	-	-	-	-	-	0000 0000b
TH0	8Ch	-	-	-	-	-	-	-	-	0000 0000b
TH1	8Dh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	8Eh									
Reserved	8Fh									
P1*	90h	-	-	-	-	-	-	-	-	1111 1111b
IRCON	91h	T2EXIF	T2IF	ADCIF	MACIF	I2CIF	SPIRXIF	SPITXIF	Reserved	0000 0000b
ANALOGPWREN	92h	0	0	0	0	TAEN	ADCEN	PGAEN	BGAPEN	0000 0000b
DIGPWREN	93h	T2CLKEN	WDOGEN	MACEN	I2CEN	SPIEN	UART1DIFFEN	UART1EN	UART0EN	0000 0000b
CLKDIVCTRL	94h	SOFTRST	-	-	IRQNORMSPD	MCKDIV_3	MCKDIV_2	MCKDIV_1	MCKDIV_0	0000 0000b
ADCCLKDIV	95h	-	-	-	-	-	-	-	-	0000 0000b
S0RELL	96h	-	-	-	-	-	-	-	-	11011001b
S0RELH	97h	0	0	0	0	0	0	-	-	0000 0011b
S0CON*	98h	S0M0	S0M1	MCPE0	R0EN	T0B8	R0B8	T0I	R0I	0000 0000b
S0BUF	99h	-	-	-	-	-	-	-	-	0000 0000b
IEN2	9Ah	-	-	-	-	-	-	-	S1IE	0000 0000b
P0PINCFG	9Bh	-	-	-	-	P0.3/RX1INE	P0.2/TX1OE	P0.1/T2EXINE	P0.0/T2INE	0000 0000b
P1PINCFG	9Ch	-	-	-	-	PWM3EN	PWM2EN	PWM1EN	PWM0EN	
P2PINCFG	9Dh	SDIEN	SDOEN	SCKEN	SSEN	CS0EN	CS1EN	CS2EN	CS3EN	0000 0000b
P3PINCFG	9Eh	MSCLEN	MSDAEN	T1INEN	CCU1EN	CCU0EN	T0INEN	RX0EN	TX0EN	0000 0000b
PORTIRQEN	9Fh	0	0	0	0	P13IEN	P12IEN	P11IEN	P10IEN	0000 0000b
P2*	A0h	-	-	-	-	-	-	-	-	1111 1111b
PORTIRQSTAT	A1h	-	-	-	-	P13ISTAT	P12ISTAT	P11ISTAT	P10ISTAT	0000 0000b
ADCCTRL	A2h	ADCIRQCLR	XVREFCAP	1	ADCIRQ	ADCIE	ONECHAN	CONT	ONESHOT	0000 0000b
ADCCONVRLOW	A3h	-	-	-	-	-	-	-	-	0000 0000b
ADCCONVRMED	A4h	-	-	-	-	-	-	-	-	0000 0000b
ADCCONVRHIGH	A5h	-	-	-	-	-	-	-	-	0000 0000b
ADCD0LO	A6h	-	-	-	-	-	-	-	-	0000 0000b
ADCD0HI	A7h					ADCD0HI_3	ADCD0HI_2	ADCD0HI_1	ADCD0HI_0	0000 0000b
IEN0*	A8h	EA	WDT	T2IE	S0IE	T1IE	0	T0IE	INT0IE	0000 0000b
ADCD1LO	A9h	-	-	-	-	-	-	-	-	0000 0000b
ADCD1HI	AAh					ADCD1HI_3	ADCD1HI_2	ADCD1HI_1	ADCD1HI_0	0000 0000b
ADCD2LO	ABh	-	-	-	-	-	-	-	-	0000 0000b
ADCD2HI	ACh					ADCD2HI_3	ADCD2HI_2	ADCD2HI_1	ADCD2HI_0	0000 0000b
ADCD3LO	ADh	-	-	-	-	-	-	-	-	0000 0000b
ADCD3HI	AEh					ADCD3HI_3	ADCD3HI_2	ADCD3HI_1	ADCD3HI_0	0000 0000b
Reserved	AFh									
P3*	B0h	-	-	-	-	-	-	-	-	1111 1111b
Reserved	B1h									
Reserved	B2h									
BGAPCAL	B3h	-	-	-	-	-	-	-	-	0000 0000b
PGACAL	B4h	-	-	-	-	-	-	-	-	0000 0000b
INMUXCTRL	B5h	-	ADCINSEL_2	ADCINSEL_1	ADCINSEL_0	AINEN_3	AINEN_2	AINEN_1	AINEN_0	0000 0000b
OUTMUXCTRL	B6h	-	-	-	-	-	TAOUTSEL_2	TAOUTSEL_1	TAOUTSEL_0	0000 0000b
SWITCHCTRL	B7h	-	-	-	-	SWITCH1_3	SWITCH1_2	SWITCH1_1	SWITCH1_0	0000 0000b
IP0*	B8h	UF8	WDTSTAT	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0	0000 0000b
IP1	B9h	-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0	0000 0000b
Reserved	BAh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	BBh	-	-	-	-	-	-	-	-	0000 0000b

SFR Register	SFR Adrs	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Value
PGACAL0	BCh	PGACAL0	-	-	-	-	-	-	-	0000 0000b
Reserved	BDh	-	-	-	-	-	-	-	-	0000 0000b
S1RELL	BEh	-	-	-	-	-	-	-	-	0000 0000b
S1RELH	BFh	-	-	-	-	-	-	-	-	0000 0000b
S1CON*	C0h	S1M	reserved	MCPE1	R1EN	T1B8	R1B8	T1I	R1I	0000 0000b
S1BUF	C1h	-	-	-	-	-	-	-	-	0000 0000b
CCL1	C2h	-	-	-	-	-	-	-	-	0000 0000b
CCH1	C3h	-	-	-	-	-	-	-	-	0000 0000b
CCL2	C4h	-	-	-	-	-	-	-	-	0000 0000b
CCH2	C5h	-	-	-	-	-	-	-	-	0000 0000b
CCL3	C6h	-	-	-	-	-	-	-	-	0000 0000b
CCH3	C7h	-	-	-	-	-	-	-	-	0000 0000b
T2CON*	C8h	T2PS	T2PSM	T2SIZE	T2RM1	T2RM0	T2CM	T2IN1	T2IN0	0000 0000b
CCEN	C9h	COCAH3	COCAL3	COCAH2	COCAL2	COCAH1	COCAL1	COCAH0	COCAL0	0000 0000b
CRCL	CAh	-	-	-	-	-	-	-	-	0000 0000b
CRCH	CBh	-	-	-	-	-	-	-	-	0000 0000b
TL2	CCh	-	-	-	-	-	-	-	-	0000 0000b
TH2	CDh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	CEh									
MPAGE	CFh	-	-	-	-	-	-	-	-	0000 0000b
PSW*	D0h	CY	AC	F0	RS1	RS0	OV	reserved	P	0000 0000b
Reserved	D1h									
Reserved	D2h									
Reserved	D3h									
Reserved	D4h									
Reserved	D5h									
Reserved	D6h									
Reserved	D7h									
U0BAUD	D8h	BAUDSRC	-	-	-	-	-	-	-	0000 0000b
WDTREL	D9h	PRES	WDTREL_6	WDTREL_5	WDTREL_4	WDTREL_3	WDTREL_2	WDTREL_1	WDTREL_0	0000 0000b
I2CCONFIG	DAh	I2CMASKID	I2CRXOVIE	I2CRXDAVIE	I2CTXEMPIE	I2CMANACK	I2CACKMODE	I2CMSTOP	I2CMASTER	0000 0010b
I2CCLKCTRL	DBh	-	-	-	-	-	-	-	-	0000 0000b
I2CCHIPID	DCh	I2CID_6	I2CID_5	I2CID_4	I2CID_3	I2CID_2	I2CID_1	I2CID_0	I2CWID	0100 0010b
I2CIRQSTAT	DDh	I2CGOTSTOP	I2CNOACK	I2CSDAS	I2CDATACK	I2CIDLE	I2CRXOV	I2CRXAV	I2CTXEMP	0010 1001b
I2CRXTX	DEh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DFh									
ACC*	E0h	-	-	-	-	-	-	-	-	0000 0000b
SPIRX3TX0	E1h	-	-	-	-	-	-	-	-	0000 0000b
SPIRX2TX1	E2h	-	-	-	-	-	-	-	-	0000 0000b
SPIRX1TX2	E3h	-	-	-	-	-	-	-	-	0000 0000b
SPIRX0TX3	E4h	-	-	-	-	-	-	-	-	0000 0000b
SPICTRL	E5h	SPICK_2	SPICK_1	SPICK_0	SPICS_1	SPICS_0	SPICKPH	SPICKPOL	SPIMA_SL	0000 0001b
SPICONFIG	E6h	SPICSLO	-	FSONCS3	SPI LOAD	-	SPIRXOVIE	SPIRXAVIE	SPITXEMPIE	0000 0000b
SPISIZE	E7h	-								0000 0111b
IEN1*	E8h	T2EXIE	SWDT	ADCPCIE	MACOVIE	I2CIE	SPIRXOVIE	SPITEIE	reserved	0000 0000b
SPIIRQSTAT	E9h	-	-	SPITXEMPTO	SPISLAVESEL	SPISEL	SPIOV	SPIRXAV	SPITXEMP	00011001b
Reserved	EAh									
MACCTRL1	EBh	LOADPREV	PREVMODE	OVMODE	OVRDVAL	ADDSRC_1	ADDSRC_0	MULCMD_1	MULCMD_0	0000 0000b
MACC0	ECh	-	-	-	-	-	-	-	-	0000 0000b
MACC1	EDh	-	-	-	-	-	-	-	-	0000 0000b
MACC2	EEh	-	-	-	-	-	-	-	-	0000 0000b
MACC3	EFh	-	-	-	-	-	-	-	-	0000 0000b
B*	F0h	-	-	-	-	-	-	-	-	0000 0000b
MACCTRL2	F1h	MACCLR2_2	MACCLR2_1	MACCLR2_0	MACOV32IE	-	-	MACOV16	MACOV32	0000 0000b
MACA0	F2h	-	-	-	-	-	-	-	-	0000 0000b
MACA1	F3h	-	-	-	-	-	-	-	-	0000 0000b
MACRES0	F4h	-	-	-	-	-	-	-	-	0000 0000b
MACRES1	F5h	-	-	-	-	-	-	-	-	0000 0000b
MACRES2	F6h	-	-	-	-	-	-	-	-	0000 0000b
MACRES3	F7h	-	-	-	-	-	-	-	-	0000 0000b
USERFLAGS*	F8h	UF7	UF6	UF5	UF4	UF3	UF2	UF1	UF0	0000 0000b
MACB0	F9h	-	-	-	-	-	-	-	-	0000 0000b
MACB1	FAh	-	-	-	-	-	-	-	-	0000 0000b
MACSHIFTCTRL	FBh	SHIFTMODE	ALSHSTYLE	SHIFTAMPL_5	SHIFTAMPL_4	SHIFTAMPL_3	SHIFTAMPL_2	SHIFTAMPL_1	SHIFTAMPL_0	0000 0000b
MACPREV0	FCh	-	-	-	-	-	-	-	-	0000 0000b
MACPREV1	FDh	-	-	-	-	-	-	-	-	0000 0000b
MACPREV2	FEh	-	-	-	-	-	-	-	-	0000 0000b
MACPREV3	FFh	-	-	-	-	-	-	-	-	0000 0000b

* Bit addressable

Peripheral Activation Control

Digital Peripherals Power Enable

To save power upon reset, many of the digital peripherals of the VMX51C1016 are not activated. The peripherals affected by this are:

- Timer 2 / Port1
- Watchdog timer
- MULT/ACCU unit
- I²C interface
- SPI interface
- UART0
- UART1
- Differential transceiver

Before using any of the above-listed peripherals, they must first be enabled by setting the corresponding bit of the DIGPWREN SFR register to 1.

The same rule applies when accessing a given peripheral's SFR register(s). The targeted peripheral must have been powered on (enabled) first, otherwise the SFR register content will be ignored

The following table demonstrates the structure of the DIGPWREN register.

TABLE 13: (DIGPWREN) DIGITAL PERIPHERALS POWER ENABLE REGISTER - SFR 93H

7	6	5	4
T2CLKEN	WDOGEN	MACEN	I2CEN

3	2	1	0
SPIEN	UART1DIFFEN	UART1EN	UART0EN

Bit	Mnemonic	Function
7	T2CLKEN	Timer 2 / PWM Enable 0 = Timer 2 CLK stopped 1 = Timer 2 CLK Running
6	WDOGEN	Watch Dog Enable 0 = Watch Dog Disable 1 = Watch Dog Enable
5	MACEN	1 = MULT/ACCU Unit Enable 0 = MULT/ACCU Unit Disable
4	I2CEN	1= I2C Interface Enable 0 = I2C Interface Disable This bit is merged with CLK STOP bit
3	SPIEN	1 = SPI interface is Enable 0 = SPI interface is Disable
2	UART1DIFFEN	UART1 Differential mode 0 = Disable 1 = Enable
1	UART1EN	0 = UART1 Disable 1 = UART1 Enable
0	UART0EN	0 = UART0 Disable 1 = UART0 Enable

Analog Peripherals Power Enable

The analog peripherals, such as the A/D, bandgap and PGA analog-to-digital converter, have a shared dedicated register used for enabling and disabling these peripherals. By default, these peripherals are powered down when the device is reset.

TABLE 14: (ANALOGPWREN) ANALOG PERIPHERALS POWER ENABLE REGISTER - SFR 92H

7	6	5	4
0	0	0	0

3	2	1	0
0	ADCEN	PGAEN	BGAPEN

Bit	Mnemonic	Function
7	0	Reserved, Keep at 0
6	0	Reserved, Keep at 0
5	0	Reserved, Keep at 0
4	0	Reserved, Keep at 0
3	TAEN	1 = TA Output Enable 0 = TA Output Disable
2	ADCEN	1 = ADC Enable 0 = ADC Disable
1	PGAEN	1 = PGA Enable 0 = PGA Disable
0	BGAPEN	1 = Bandgap Enable 0 = Bandgap Disable

Note: The SFR registers associated with all analog peripherals are activated when one or more analog peripherals are enabled.

General Purpose I/O

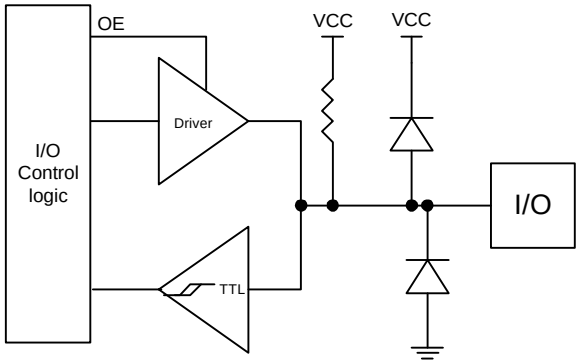
The VMX51C1016 provides 24 general-purpose I/O pins. The I/Os are shared with digital peripherals and can be configured individually.

At reset, all the VMX51C1016 I/O ports are configured as inputs. The I/O ports are bi-directional and the CPU can write or read data through any of these ports.

I/O Port Structure

The VMX51C1016 I/O port structure is shown in the following figure.

FIGURE 9 – I/O PORT STRUCTURE



Each I/O pin includes pull-up circuitry (represented by the internal pull-up resistor) and a pair of internal protection diodes internally connected to VCC and ground, providing ESD protection.

The I/O operational configuration is defined in the I/O control logic block.

I/O Port Drive Capability

Each I/O port pin, when configured as an output, can source or sink up to 4mA. The following graphs show typical I/O output voltage versus source and I/O output voltage versus sink current.

FIGURE 10: TYPICAL I/O VOUT VS. SOURCE CURRENT

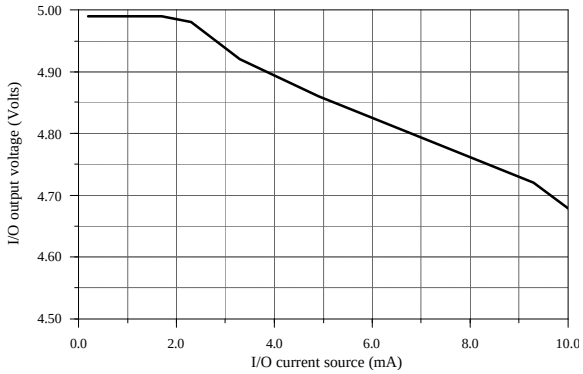
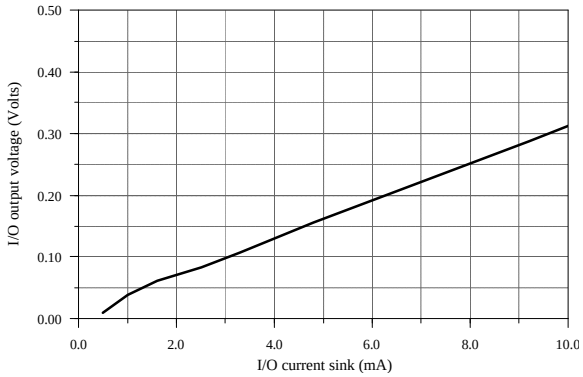


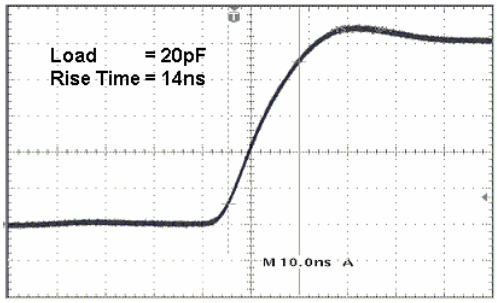
FIGURE 11: TYPICAL I/O VOUT VS. SINK CURRENT



The maximum recommended driving current of a single I/O on a given port is 10mA. The recommended limit when more than one I/O on a given port is driving current is 5mA on each I/O. The total current drive of all I/O ports should be limited to 40mA

The following figure shows a typical I/O rise time when driving a 20pF capacitive load. In this case, rise time is about 14ns.

FIGURE 12: I/O RISE TIME WITH A 20PF LOAD



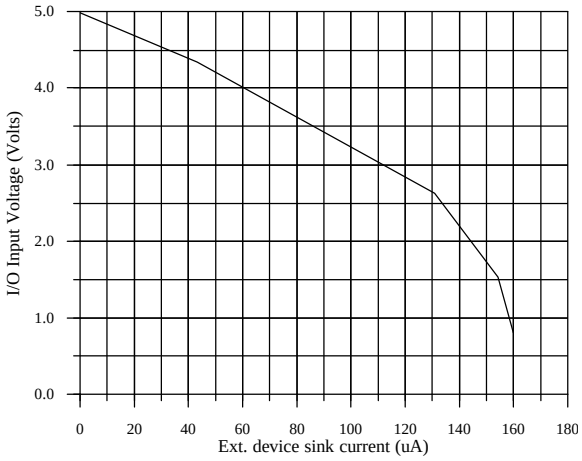
Input Voltage vs. Ext. Device Sink

The I/Os of the VMX51C1016, when configured as inputs, include an internal pull-up resistor made up of a transistor, which ensures that the level present at input is stable when the I/O pins are disconnected.

Due to the presence of the pull-up resistor on the digital inputs, the external device driving the I/O must be able to sink enough current to bring the I/O pin low.

The following figure shows the VMX51C1016 input port voltage versus the external device sink current.

FIGURE 13: INPUT PORT VOLTAGE VS. EXT DEVICE SINK CURRENT



I/O Port Configuration Registers

The VMX51C1016's I/O port operation is controlled by two sets of four registers:

- Port pin configuration registers
- Port access registers

The port pin configuration registers combined with specific peripheral configuration will define whether a given pin acts as a general purpose I/O or provides the alternate peripheral functionality.

Before using a peripheral that is shared with I/Os, the pin corresponding to the peripheral output must be configured as an output and the pins that are shared with the peripheral inputs must be configured as inputs.

The following registers are used to configure each of the ports as either general-purpose inputs, outputs or alternate peripheral functions. For example, when bit 5 of Port 2 is configured as an output, it will output the SCK signal if the SPI interface is enabled and working.

The only exception to this rule is the I^2C clock and data bus signals. In these two cases, the VMX51C1016 configures the pins automatically as inputs or outputs.

The P0PINCFG register controls the I/O access to UART1 and Timer 2's input and output, and defines the direction of P0 when used as a general purpose I/O.

TABLE 15: (P0PINCFG) PORT 0 PORT CONFIGURATION REGISTER - SFR 9BH

7	6	5	4
P07IO	P06IO	P05IO	P04IO

3	2	1	0
P0.3/RX1INE	P0.2/TX1OE	P0.1/T2EXINE	P0.0/T2INE

Bit	Mnemonic	Function
7:4	P0xIO	Not connected on VMX51C1016
3	P0.3/RX1INE	0: General purpose input or UART1 RX 1: General purpose output ***When using UART1 you must set this bit to 0.***
2	P0.2/TX1OE	0: General purpose input 1: General purpose output or UART1 TX ***When using UART1 you must set this bit to 1.***
1	P0.1/T2EXINE	0: General purpose input or Timer 2 EX 1: General purpose output ***When using Timer 2EX input you must set this bit to 0.***
0	P0.0/T2INE	0: General purpose input or Timer 2 IN 1: General purpose output ***When using Timer 2 input you must set this bit to 0.***

The P1PINCFG register controls access from the PWM to the I/O pins and defines the direction of P1 when the PWM's are not used.

TABLE 16: (P1PINCFG) PORT 1 PORT CONFIGURATION REGISTER - SFR 9CH

7	6	5	4
P1[7:4]			

3	2	1	0
P1.3/PWM3EN	P1.2/PWM2EN	P1.1/PWM1EN	P1.0/PWM0EN

Bit	Mnemonic	Function
7:4	P1[7:4]	Not connected on VMX51C1016
3	P1.3/PWM3OE	0: General purpose input 1: General purpose output or PWM bit 3 output ***When using PWM you must set this bit to 1.***
2	P1.2/PWM2OE	0: General purpose input 1: General purpose output or PWM bit 2 output ***When using PWM you must set this bit to 1***
1	P1.1/PWM1OE	0: General purpose input 1: General purpose output or PWM bit 1 output ***When using PWM you must set this bit to 1***
0	P1.0/PWM0OE	0: General purpose input 1: General purpose output or PWM bit 0 output ***When using PWM you must set this bit to 1***

The P2PINCFG register controls I/O access to the SPI interface and defines the direction of P2 when used as a general purpose I/O.

TABLE 17: (P2PINCFG) PORT 2 PORT CONFIGURATION REGISTER - SFR 9DH

7	6	5	4
P2.7/SDIEN	P2.6/SDOEN	P2.5/SCKEN	P2.4/SSEN

3	2	1	0
CS0EN	CS1EN	CS2EN	CS3EN

Bit	Mnemonic	Function
7	P2.7/SDIEN	0: General purpose input or SDI 1: General purpose output ***When using SPI you must set this bit to 0.***
6	P2.6/SDOEN	0: General purpose input 1: General purpose output or SDO ***When using SPI you must set this bit to 1.***
5	P2.5/SCKEN	0: General purpose input or SCK 1: General purpose output ***When using SPI you must set this bit to 0.***
4	P2.4/SSEN	0: General purpose input or Slave Select 1: General purpose output ***When using SPI SS you must set this bit to 0.***
3	P2.3/CS0EN	0: General purpose input 1: General purpose output or Chip Select bit 0 output ***When using SPI CS0 you must set this bit to 1.***
2	P2.2/CS1EN	0: General purpose input 1: General purpose output or Chip Select bit 1 output ***When using SPI CS1 you must set this bit to 1.***
1	P2.1/CS2EN	0: General purpose input 1: General purpose output or Chip Select bit 2 output ***When using SPI CS2 you must set this bit to 1.***
0	P2.0/CS3EN	0: General purpose input 1: General purpose output or Chip Select bit 3 output ***When using SPI CS3 you must set this bit to 1.***

The P3PINCFG register controls I/O access to UART0, the I²C interface, capture and compare inputs 0 and 1 and timer 0/1's inputs, and defines the direction of P3 when used as a general purpose I/O.

TABLE 18: (P3PINCFG) PORT 3 PORT CONFIGURATION REGISTER - SFR 9EH

7	6	5	4
P3.7/MSCLEN	P3.6/MSDAEN	P3.5/T1INEN	P3.4/CCU1EN

3	2	1	0
P3.3/CCU0EN	P3.2/T0INEN	P3.1/RX0EN	P3.0/TX0EN

Bit	Mnemonic	Function
7	P3.7/MSCLEN	0: General purpose input 1: General purpose output or Master I2C SCL output ***When using the I2C you must set this bit to 1.***
6	P3.6/MSDAEN	0: General purpose input 1: General purpose output or Master I2C SDA ***When using the I2C you must set this bit to 1.***
5	P3.5/T1INEN	0: General purpose input or Timer1 Input 1: General purpose output ***When using Timer 1 you must set this bit to 0.***
4	P3.4/CCU1EN	0: General purpose input or CCU1 Input 1: General purpose output ***When using the Compare and Capture unit you must set this bit to 0.***
3	P3.3/CCU0EN	0: General purpose input or CCU0 Input 1: General purpose output ***When using the Compare and Capture unit you must set this bit to 0.***
2	P3.2/T0INEN	0: General purpose input or Timer 0 Input 1: General purpose output ***When using Timer 0 you must set this bit to 0.***
1	P3.1/RX0EN	0: General purpose input or UART0 Rx 1: General purpose output ***When using UART0 you must set this bit to 0.***
0	P3.0/TX0EN	0: General purpose input 1: General purpose output or UART0 Tx ***When using UART0 you must set this bit to 1.***

Using General Purpose I/O Ports

The VMX51C1016's 24 I/Os are grouped into four ports. For each port an SFR register location is defined. These registers are bit addressable, providing the ability to control the I/O lines individually. The upper 4 bits of Port 0 and Port 1 are not pinned out.

When the port pin configuration register value defines the pin as an output, the value written into the port register will be reflected at the pin level.

Reading the I/O pin configured as input is done by reading the contents of its associated port register.

TABLE 19:
PORT 0 - SFR 80H

7	6	5	4	3	2	1	0
P0 [7:0]							

PORT 1 - SFR 90H

7	6	5	4	3	2	1	0
P1 [7:0]							

PORT 2 - SFR A0H

7	6	5	4	3	2	1	0
P2 [7:0]							

PORT 3 - SFR B0H

7	6	5	4	3	2	1	0
P3 [7:0]							

Bit	Mnemonic	Function
7-0	P0, 1, 2, 3	When the Port is configured as an output, setting a port pin to 1 will make the corresponding pin to output logic high. When set to 0, the corresponding pin will set a logic low.

I/O usage example

The following example demonstrates the configuration of the VMX51C1016 I/Os.

```
//-------------------------------------------------------------------------
//This example continuously reads the P0 and writes its contents into //P1 and it toggle P2 and P3.
//-------------------------------------------------------------------------
#pragma TINY
#pragma UNSIGNEDCHAR

#include <VMIXReg.h>

at 0x0000 void main (void)
{
  DIGPWREN = 0x80;              // Enable Timer 2 to activate P1
                                //Output
  P0PINCFG = 0x00;              // Configure all P0 as Input
  P1PINCFG = 0x0F;              //Configure P1 as Output
  P2PINCFG = 0xFF;              //Configure P2 as Output
  P3PINCFG = 0xFF;              //Configure P3 as Output

  while(1)
   {
     P1 = P0;                   //Write P0 into P1
     P2 = ~P2;                  //Toggle P2 & P3
     P3 = ~P3;
   }
}//end of main() function
```

Using Port 1.0-3 as General Purpose Output

Port 1.0-P1.3 can be used as standard digital output. In order to do this, the Timer 2 clock must be enabled by setting the T2CLKEN bit of the DIGPWREN register. In addition, the Timer 2 CCEN register must also have the reset value.

Interrupt on Port 1 Change Feature

The VMX51C1016 includes ann interrupt on Port 1 change feature. This can be used to monitor the activity on each I/O Port 1 pin (individually), and trigger an interrupt when the state of the pin on which this feature has been activated changes. This is equivalent to having eight individual external interrupt inputs. The interrupt on Port 1 change shares the interrupt vector of the ADC peripheral at address 006Bh.

See the interrupt section for more details on how to use this feature.

MULT/ACCU - Multiply Accumulator Unit

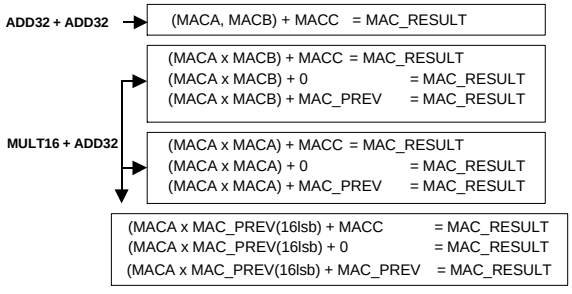
MULT/ACCU Features

The VMX51C1016 includes a hardware based multiply-accumulator unit, which allows the user to perform fast and complex arithmetic operations.

MULT/ACCU unit features:

- Hardware calculation engine
- Calculation result is ready as soon as the input registers are loaded
- Signed mathematical calculations
- Unsigned MATH operations are possible if the MUL engine operands are limited to 15 bits in size
- Auto/Manual reload of MAC_RES
- Enhanced VMX51C1016 MULT/ACCU unit
- Easy implementation of complex MATH operations
- 16-bit and 32-bit overflow flag
- 32-bit overflow can trigger an interrupt
- MULT/ACCU operand registers can be cleared individually or all together
- Overflow flags can be configured to stay active until manually cleared
- Can store and use results from previous operations
- MULT/ACCU can be configured to perform the following operations:

FIGURE 13: VMX51C1016 MULT/ACCU OPERATION



Where MACA (multiplier), MACB (multiplicand), MACACC (accumulator) and MACRESULT (result) are 16, 16, 32 and 32 bits, respectively.

MULT/ACCU Control Registers

With the exception of the barrel shifter, the MULT/ACCU unit operation is controlled by two SFR registers:

- MACCTRL1
- MACTRLC2

The following two tables describe these control registers:

TABLE 20: (MACCTRL1) MULT/ACCU UNIT CONTROL REGISTER - SFR EBH

7	6	5	4
LOADPREV	PREVMODE	OVMODE	OVRDVAL

3	2	1	0
ADDSRC [1:0]		MULCMD [1:0]	

Bit	Mnemonic	Function
7	LOADPREV	MACPREV manual Load control 1 = Manual load of the MACPREV register content if PREVMODE = 1
6	PREVMODE	Loading method of MACPREV register 0 = Automatic load when MACA0 is written. 1 = Manual Load when 1 is written into LOADPREV
5	OVMODE	0 = Once set by math operation, the OV16 and OV32 flag will remain set until the overflow condition is removed. 1= Once set by math operation, the OV16 and OV32 flag will stay set until it is cleared manually.
4	OVRDVAL	0 = The value on MACRES is the calculation result. 1 = the value on MACRES is the 32LSB of the MACRES when the OV32 overflow occurred
3:2	ADDSRC[1:0]	32-bit Addition source **B Input** 00 = 0 (No Add) 01 = C (std 32-bit reg) 10 = RES –1 11 = C (std 32-bit reg) **A Input** 00=Multiplication 01=Multiplication 10=Multiplication 11= Concatenation of {A, B} for 32-bit addition
1:0	MULCMD[1:0]	Multiplication Command 00 = MACA x MACB 01 = MACA x MACA 10 = MACA x MACPREV (16 LSB) 11 = MACA x MACB

TABLE 21: (MACCTRL2) MULT/ACCU UNIT CONTROL REGISTER 2 -SFR F1H

7	6	5	4
MACCLR2 [2:0]			MACOV32IE

3	2	1	0
-	-	MACOV16	MACOV32

Bit	Mnemonic	Function
7:5	MACCLR[2:0]	MULT/ACCU Register Clear 000 = No Clear 001 = Clear MACA 010 = Clear MACB 011 = Clear MACC 100 = Clear MACPREV 101 = Clear All MAC regs + Overflow Flags 110 = Clear Overflow Flags only
4	MACOV32IE	MULT/ACCU 32-bit Overflow IRQ Enable
3	-	-
2	-	-
1	MACOV16	16-bit Overflow Flag 0 = No 16 overflow 1 = 16-bit MULT/ACCU Overflow occurred
0	MACOV32	32-bit Overflow Flag 1 = 32-bit MULT/ACCU Overflow This automatically loads the MAC32OV register. The MACOV32 can generate a MULT/ACCU interrupt when enabled.

MULT/ACCU Unit Data Registers

The MULT/ACCU data registers include operand and result registers that serve to store the numbers being manipulated in mathematical operations. Some of these registers are uniquely for addition (such as MACC), while others can be used for all operations. The MULT/ACCU operation registers are represented below.

MACA and MACB Multiplication (Addition) Input Registers

The MACA and MACB register serves as 16-bit input operands when performing multiplication. When the MULT/ACCU is configured to perform 32-bit addition, the MACA and the MACB registers are concatenated to represent a 32-bit word. In such cases, the MACA register contains the upper 16-bit of the 32-bit operand and the MACB contains the lower 16 bits.

TABLE 22: (MACA0) MULT/ACCU UNIT A OPERAND, LOW BYTE - SFR F2H

7	6	5	4	3	2	1	0
MACA0 [7:0]							

Bit	Mnemonic	Function
7:0	MACA0	Lower segment of the MACA operand

TABLE 23: (MACA1) MULT/ACCU UNIT A OPERAND, HIGH BYTE - SFR F3H

7	6	5	4	3	2	1	0
MACA1 [15:8]							

Bit	Mnemonic	Function
15:8	MACA1	Upper segment of the MACA operand

TABLE 24: (MACB0) MULT/ACCU UNIT B OPERAND, LOW BYTE - SFR F9H

7	6	5	4	3	2	1	0
MACB0 [7:0]							

Bit	Mnemonic	Function
7:0	MACB0	Lower segment of the MACB operand

TABLE 25: (MACB1) MULT/ACCU UNIT B OPERAND, HIGH BYTE - SFR FAH

7	6	5	4	3	2	1	0
MACB1 [7:0]							

Bit	Mnemonic	Function
7:0	MACB1	Upper segment of the MACB operand

MACC Input Register

The MACC register is a 32-bit register used to perform 32-bit addition.

It is possible to substitute the MACPREV register for the MACC register or 0 in the 32-bit addition.

TABLE 26: (MACC0) MULT/ACCU UNIT C OPERAND, LOW BYTE - SFR ECH

7	6	5	4	3	2	1	0
MACC0 [7:0]							

Bit	Mnemonic	Function
7:0	MACC0	Lower segment of the 32-bit addition register

TABLE 27: (MACC1) MULT/ACCU UNIT C OPERAND, BYTE 1 - SFR EDH

7	6	5	4	3	2	1	0
MACC1 [15:8]							

Bit	Mnemonic	Function
15:8	MACC1	Lower middle segment of the 32-bit addition register

TABLE 28: (MACC2) MULT/ACCU UNIT C OPERAND, BYTE 2 - SFR EEH

7	6	5	4	3	2	1	0
MACC2 [23:16]							

Bit	Mnemonic	Function
23:16	MACC2	Upper middle segment of the 32-bit addition register

TABLE 29: (MACC3) MULT/ACCU UNIT C OPERAND, HIGH BYTE - SFR EFH

7	6	5	4	3	2	1	0
MACC3 [31:24]							

Bit	Mnemonic	Function
31:24	MACC3	Upper segment of the 32-bit addition register

MACRES Result Register

The MACRES register, which is 32 bits wide, contains the result of the MULT/ACCU operation. In fact, the MACRES register is the output of the barrel shifter.

TABLE 30: (MACRES0) MULT/ACCU UNIT RESULT, LOW BYTE - SFR F4H

7	6	5	4	3	2	1	0
MACRES0 [7:0]							

Bit	Mnemonic	Function
7:0	MACRES0	Lower segment of the 32-bit MULT/ACCU result register

TABLE 31: (MACRES1) MULT/ACCU UNIT RESULT, BYTE 1 - SFR F5H

7	6	5	4	3	2	1	0
MACRES1 [15:8]							

Bit	Mnemonic	Function
15:8	MACRES1	Lower middle segment of the 32-bit MULT/ACCU result register

TABLE 32: (MACRES2) MULT/ACCU UNIT RESULT, BYTE 2 - SFR F6H

7	6	5	4	3	2	1	0
MACRES2 [23:16]							

Bit	Mnemonic	Function
23:16	MACRES2	Upper middle segment of the 32-bit MULT/ACCU result register

TABLE 33: (MACRES3) MULT/ACCU UNIT RESULT, HIGH BYTE - SFR F7H

7	6	5	4	3	2	1	0
MACRES3 [31:24]							

Bit	Mnemonic	Function
31:24	MACRES3	Upper segment of the 32-bit MULT/ACCU result register

MACPREV Register

The MACPREV register provides the ability to automatically or manually save the contents of the MACRES register and re-inject it into the calculation. This feature is especially useful in applications where the result of a given operation serves as one of the operands of the next one.

As previously mentioned, there are two ways to load the MACPREV register controlled by the PREVMODE bit value:

PREVMODE = 0:
Auto MACPREV load, by writing into the MACA0 register. Selected when PREVMODE = 0.

PREVMODE = 1:
Manual load of MACPREV when the LOADPREV bit is set to 1.

A good example demonstrating the auto loading of the MACPREV feature is the implementation of a FIR filter. In that specific case, it is possible to save a total of 8 MOV operations per tap calculation.

TABLE 34: (MACPREV0) MULT/ACCU UNIT PREVIOUS OPERATION RESULT, LOW BYTE - SFR FCH

7	6	5	4	3	2	1	0
MACPREV0 [7:0]							

Bit	Mnemonic	Function
7:0	MACPREV0	Lower segment of 32-bit MULT/ACCU previous result register

TABLE 35: (MACPREV1) MULT/ACCU UNIT PREVIOUS OPERATION RESULT, BYTE 1 - SFR FDH

7	6	5	4	3	2	1	0
MACPREV1 [7:0]							

Bit	Mnemonic	Function
15:8	MACPREV1	Lower middle segment of 32-bit MULT/ACCU previous result register

TABLE 36: (MACPREV2) MULT/ACCU UNIT PREVIOUS OPERATION RESULT, BYTE 2 - SFR FEH

7	6	5	4	3	2	1	0
MACPREV2 [15:8]							

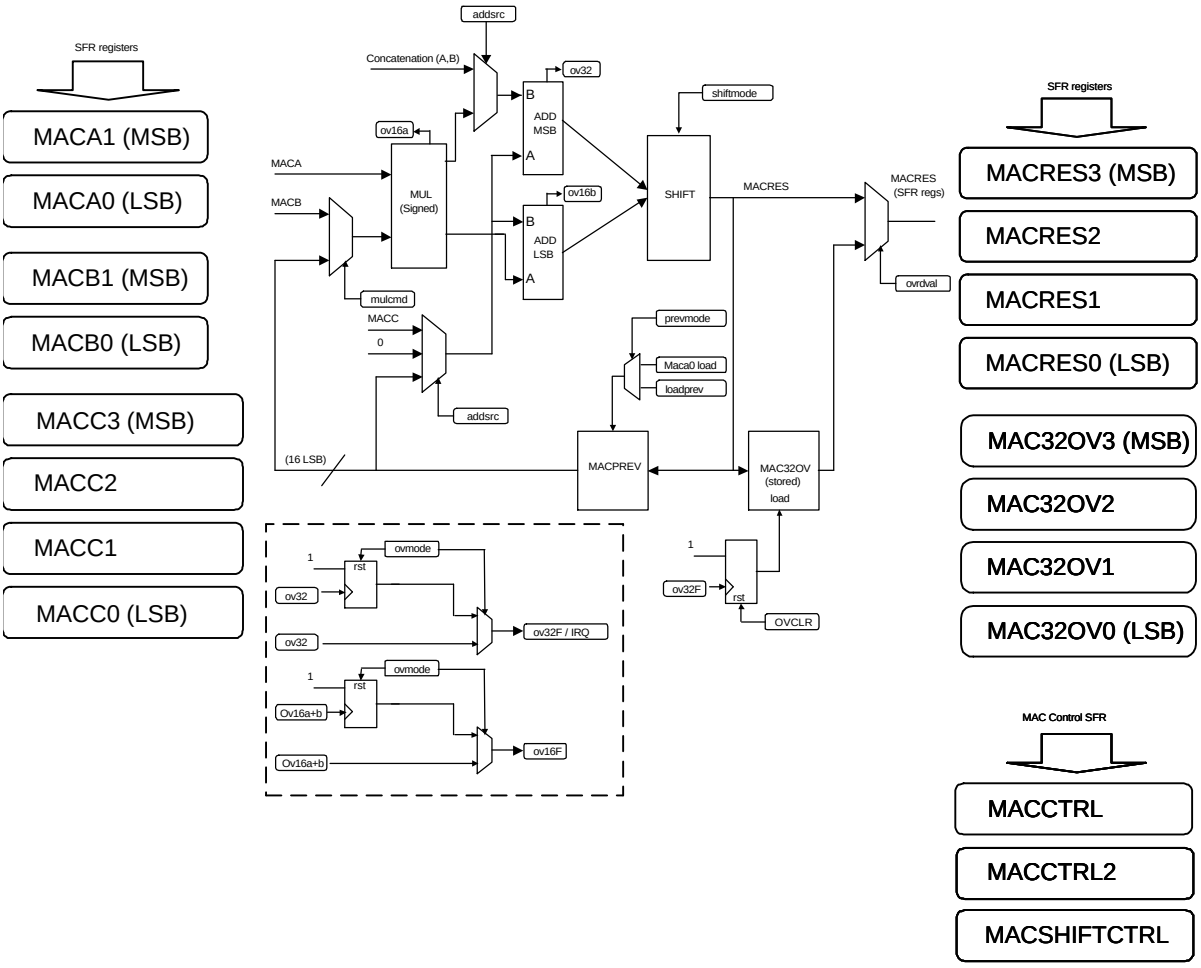
Bit	Mnemonic	Function
23:16	MACPREV2	Upper middle segment of 32-bit MULT/ACCU previous result register

TABLE 37: (MACPREV3) MULT/ACCU UNIT PREVIOUS OPERATION RESULT, HIGH BYTE - SFR FFH

7	6	5	4	3	2	1	0
MACPREV3 [7:0]							

Bit	Mnemonic	Function
31:24	MACPREV3	Upper segment of 32-bit MULT/ACCU previous result register

FIGURE 14: VMX51C1016 MULT/ACCU FUNCTIONAL DIAGRAM



The block diagram above shows the interaction between the registers and the other components that comprise the MULT/ACCU unit on the VMX51C1016.

MULT/ACCU Barrel Shifter

The MULT/ACCU includes a 32-bit barrel shifter at the output of the 32-bit addition unit. The barrel shifter can perform right/left shift operations in one cycle, which is useful for scaling the output result of the MULT/ACCU.

The shifting range is adjustable from 0 to 16 in both directions. The "shifted" addition unit output can be routed to:

- MACRES
- MACPREV
- MACOV32

The barrel shifter can perform both arithmetic and logical shifts: The shift left operation can be configured as an arithmetic or logical shift. In the latter, the sign bit is discarded.

TABLE 38: (MACSHIFTCTRL) MULT/ACCU UNIT BARREL SHIFTER CONTROL REGISTER - SFR FBH

7	6	5	4	3	2	1	0
SHIFTMODE	ALSHSTYLE	SHIFTAMPL [5:0]					

Bit	Mnemonic	Function
7	SHIFTMODE	0 = Logical SHIFT 1 = Arithmetic SHIFT
6	ALSHSTYLE	Arithmetic Shift Left Style 0= Arithmetic Left Shift: Logical Left 1= Arithmetic Left Shift: Keep sign bit
5:0	SHIFTAMPL[5:0]	Shift Amplitude 0 to 16 (5 bits to provide 16 bits shift range) Neg. Number = Shift Right (2 complements) Pos. Number = Shift Left

MULT/ACCU Unit Setup and OV32 Interrupt Example

In order to use the MULT/ACCU unit, the user must first set up and configure the module. The following provides setup code examples. The first part of the code is the interrupt setup and module configuration, while the second part is the interrupt function itself.

Sample C code for MULT/ACCU unit interrupt setup and module configuration:

```
//-------------------------------------------------------------------------
// Sample C code to setup the MULT/ACCU unit
//-------------------------------------------------------------------------

//--- Program initialisation omitted…
(…)
void main(void){
// MULT/ACCU setup
IEN0 |= 0x80;                   // Enable all interrupts
IEN1 |= 0x10;                   // Enable MULT/ACCU interrupt
DIGPWREN |= 0x20;               // Enable MULT/ACCU unit
MACCTRL1 = 0x0C;                // {A,B}+C
MACCTRL2 = 0x10;                // Enable INT overflow_32

// MULT/ACCU example use

MACA0 = 0xFF;
MACA1 = 0x7F;
MACB0 = 0xFF;
MACB1 = 0xFF;
MACC0 = 0xFF;
MACC1 = 0xFF;
MACC2 = 0xFF;
MACC3 = 0x7F;

//--- as soon as the MAC input registers are loaded the result is available in the
MACRESx registers.
}//end of main

//-------------------------------------------------------------------------
// MAC 32 bit overflow Interrupt Function

void int_5_mac (void) interrupt 12
{
IEN0 &= 0x7F;                   // Disable all interrupts

//Put MAC 32 bit Overflow Interrupt code here.*/
//Note that when a 32bit overflow occurs, the 32 least significant bit of the current
//result are stored into the MAC32OVx registers and can be read at the location
of MACRESx by setting to 1 the OVRDVAL bit of the MACCTRL register

IRCON &= 0xEF;                  // Clear flag (IEX5)
IEN0 |= 0x80;                   // Enable all interrupts
}
//-------------------------------------------------------------------------
```

MULT/ACCU Application Example: FIR Filter Function

The following ASM code shows the implementation of an FIR filter computation function for one iteration, followed by the data shifting operation and the definition of the FIR filter coefficient table. The FIR computation is simple to implement, however, it is quite demanding in terms of processing power. For each new data point, the multiplication with associated coefficients plus addition operation must be performed N times (N=number of filter tapps).

Since it is hardware based and provides an automatic reload of the result of the previous operation feature, the VMX51C1016 MULT/ACCU unit is very efficient in performing operations such as FIR filter computation.

In the code example below, the COMPUTEFIR loop forms the "heart" of the FIR computation. It is clear that use of the MULT/ACCU unit implies very few instructions to perform mathematical operations. The net result is a dramatic performance improvement when compared with manual calculations done solely via the standard 8051 instruction set.

VMX51C1016 FIR Filter Example

The example below shows how to use the VMX51C1016's MULT/ACCU unit to perform FIR filter computing. To minimize the example size, only the FIR computing function and the coefficient table are presented.

```
;-----------------------------------------------------//
;** FIR Filter Computing Function          //
;-----------------------------------------------------//
FIRCOMPUTE:    MOV    R0,#NPOINTSBASEADRS
                                                ;INPUT ADC RAW DATA
                                                ;AT Xn LOCATIONS...

;Saving acquired data from calling function into RAM for computation

               MOV          VARH,DATAH
               MOV          VARL,DATAL
               MOV          @R0,VARH            ;(MSB)
               INC          R0
               MOV          @R0,VARFL           ;(LSB)

;** Prepare to compute Yn...
;***Define Base ADRS of input values
               MOV          R0,#NPOINTSBASEADRS

;***Define Base Address of coefficients
               MOV          R1,#COEFBASEADRS
               MOV          R7,#NPOINTS         ;DEFINE COUNTER

;***Configure the MULT/ACCU unit as Follow:

               MOV          MACCTRL,#00001000B

               ;BIT7 LOADPREV = 0      No manual Previous result
               ;BIT6 PREVMODE = 0      Automatic Previous result save when
               ;                       MULT/ACCUA0 is loaded
               ;BIT5 OVMODE   = 0      Overflow flag remains ON until overflow
               ;                       condition exist
               ;BIT4 OVRDVAL  = 0      The value of MACRES is the calculation
               ;                       result
               ;BIT3:2 ADDSRC = 10     MACPREV is the Addition Source
               ;BIT1:0 MULCMD = 00     Mul Operation = MACAxMACB


;**Clear the MULT/ACCU registers content
               MOV                    MACCTRL2,#0A0H
;** COMPUTE Yn...

COMPUTEFIR:    MOVMACB1,@R1           ;Put a given Coefficient into
                                      ;MULT/ACCUB
               INC          R1
               MOV          MACB0,@R1
               INC          R1

               MOV          MACA1,@R0   ; Put a given Xn Input into
               INC          R0
               MOV           MACA0,@R0
               ;This last instruction load the MACPREV register for next Operation
               INC          R0
               DJNZ         R7,COMPUTEFIR  ;Do the Computation for N taps
```

```
;*** Second part
;-----------------------------------------------------------------------------------------------//
;** SHIFT PREVIOUS INPUT VALUES TO LET PLACE FOR NEXT ONE...
;-----------------------------------------------------------------------------------------------//
SHIFTPAST:
               MOV          R7,#(NPOINTS-1)*2      ;Define # of datashift
                                                   ;To perform (N-1)*2
;***COMPUTE FIRST FETCH ADDRESS
               MOV          R0,#(NPOINTSBASEADRS - 1 + 2*(NPOINTS-1))

;***COMPUTE FIRST DESTINATION ADDRESS
               MOV          R1,#(NPOINTSBASEADRS + 1 + 2*(NPOINTS-1))
SHIFTLOOP:  MOV             A,@R0       ;Shift Given LSB input...
               MOV          @R1,A       ;To next location
               DEC          R0          ;Prepare pointer for moving LSB
               DEC          R1
               DJNZ         R7,SHIFTLOOP

;** PERFORM TRANSFORMATION OF Yn HERE AND PUT INTO BINH, BINL
;** IN THIS CASE THE COEFFICIENTS HAVE BEEN MULTIPLIED BY 65536
;** SO THE RESULT IS ON 32-BITS
;** DIVISING YN BY 65536 MEAN ONLY TAKING THE UPPER 16-BITS

               MOV          DATAH,MACRES3
               MOV          DATAL,MACRES2

               LCALL        SENDLTC1452
               MOV          P3,#00
               RET

;-----------------------------------------------
;* FIR Filter Coefficients Table          *
;-----------------------------------------------
;FSAMPLE 480HZ, N=16, LOW PASS 0.1HZ -78DB @ 60HZ

COEFTABLE:     DW     023DH
               DW     049DH
               DW     086AH
               DW     0D2DH
               DW     1263H
               DW     1752H
               DW     1B30H
               DW     1D51H
               DW     1D51H
               DW     1B30H
               DW     1752H
               DW     1263H
               DW     0D2DH
               DW     086AH
               DW     049DH
               DW     023DH

               DW     0FFFFH          ;END OF TABLE
```

VMX51C1016 Timers

The VMX51C1016 includes three general-purpose timer/counters

- Timer 0
- Timer 1
- Timer 2

Timer 0 and Timer 1 are general purpose timers that can operate as either a timer with a clock rate based on the system clock, or an event counter that monitors events occurring on an external timer input pin (T0IN for Timer 0 and T1IN for Timer 1).

Timers 0 and 1 are similar to standard 8051 timers. In addition to operating as a timer based on a system clock or as an event counter, Timer 2 is also the heart of the PWM counter outputs and the compare and capture units.

Each of the VMX51C1016's timers has a dedicated interrupt vector, which can be triggered when the timers overflow.

Timer 0 and Timer 1

Timer 0 and Timer 1 are similar in their structure and operation. The main differences between them is that the Timer 1 serves as a baud rate generator for UART0 and shares some of its resources when Timer 0 is used in Mode 3.

Timer 0 and Timer 1 each consist of a 16-bit register, for which content is accessible as two independent SFR registers: TLx and THx.

TABLE 39: (TL0) TIMER 0 LOW BYTE - SFR 8AH

7	6	5	4	3	2	1	0
TL0 [7:0]							

TABLE 40: (TH0) TIMER 0 HIGH BYTE - SFR 8CH

7	6	5	4	3	2	1	0
TH0 [7:0]							

TABLE 41: (TL1) TIMER 1 LOW BYTE - SFR 8BH

7	6	5	4	3	2	1	0
TL1 [7:0]							

TABLE 42: (TH1) TIMER 1 HIGH BYTE - SFR 8DH

7	6	5	4	3	2	1	0
TH1 [7:0]							

With the exception of their associated interrupts, the configuration and control of timers 0 and 1 are performed via the TMOD and TCON SFR registers.

The following table shows the TCON special function register of the VMX51C1016. This register contains the Timer 0/1 overflow flags, Timer 0/1 run control bits, INT0 edge flags and INT0 interrupt type control bits.

TABLE 43: (TCON) TIMER 0, TIMER 1 TIMER/COUNTER CONTROL - SFR 88H

7	6	5	4
TF1	TR1	TF0	TR0

3	2	1	0
-	-	IE0	IT0

Bit	Mnemonic	Function
7	TF1	Timer 1 overflow flag. Set by hardware when Timer 1 overflows. It is automatically cleared when the Timer 1 interrupt is serviced. This flag can also be cleared by software.
6	TR1	Timer 1 Run control bit. TR1 = 0, Stop Timer 1 TR1 = 1, Start Timer 1
5	TF0	Timer 0 overflow flag. Set by hardware when Timer 0 overflows. It is automatically cleared when the Timer 0 interrupt is serviced. This flag can also be cleared by software.
4	TR0	Timer 0 Run control bit. TR0 = 0, Stop Timer 0 TR0 = 1, Start Timer 0
3	-	Reserved
2	-	Reserved
1	IE0	INT0 edge flag configuration Set by hardware when falling edge on external pin INT0 is observed. It is cleared when interrupt is processed.
0	IT0	INT0 interrupt event type control bit. IT0 = 0, interrupt will be caused by a Low Level on INT0 IT0 = 1, Interrupt will be caused by a High to Low transition on INT0.

The TMOD register is used mainly to set the timers' operating mode and allows the user to enable the external gate control as well as select a timer or counter operation.

TABLE 44: (TMOD) TIMER MODE CONTROL - SFR 89H

7	6	5	4
-	CT1	M11	M01

3	2	1	0
GATE0	CT0	M10	M00

Bit	Mnemonic	Function
7	Reserved	
	CT1	Selects TIMER1 Operation. CT1 = 0, Sets the Timer 1 as a Timer which value is incremented by SYSCLK events. CT1 = 1, The Timer 1 operates as a counter which counts the High to Low transition on that occurs on the T1IN input.
5	M11	Selects mode for Timer/Counter 1, as shown in the Table below.
4	M01	
3	GATE0	GATE0 = 0, The level present on the INT0 pin has no affect on Timer1 operation. GATE0 = 1, The level of INT0 pin serves as a Gate control on to Timer/Counter operation provided the TR1 bit is set. Applying a Low Level on the INT0 pin makes the Timer stop.
2	CT0	Selects Timer 0 Operation. CT1 = 0, Sets the Timer 0 as a Timer which value is incremented by SYSCLK events. CT1 = 1, The Timer 0 operates as a counter which counts the High to Low transition on that occurs on the T1IN input.
1	M10	Selects mode for Timer/Counter 0, as shown in the Table below.
0	M00	

Timer 0/Timer 1/Counter Operation

The CT0 and CT1 bits of the TMOD register control the clock source for Timer 0 and Timer 1, respectively. When the CT bit is set to 0 (timer mode), the timer is sourced from the system clock divided by 12.

Setting the CTx bit to 1 configures the timer to operate in event counter mode. In this mode, high to low transitions on the TxIN pin of the VMX51C1016 increment the timer value.

Note that when Timer 0 and Timer 1 operate in timer mode, they use the system clock as their source. Therefore, configuring the CLKDIVCTRL register will affect the timers' operation.

Timer 0, Timer 1 Gate Control

Gate control makes it possible for an external device to control the Timer 0 operation through the interrupt (INT0) pins.

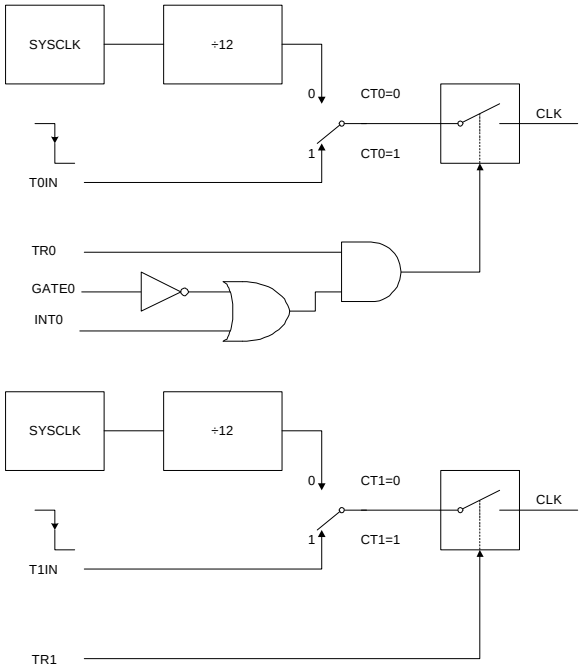
When the GATE0 and TR0 bits of the TMOD register are set to 1:

- INT0 = Logic low, Timer 0 stops
- INT0 = Logic high, Timer 0 runs

When the gate bit equals 0, the logic level presented at the INT0 pin has no affect on the Timer 0 operation.

Gate control is not possible on Timer 1 because the INT1 is not pinned out and an internal pull-up resistor keeps its level high.

FIGURE 15: TIMER 0, TIMER 1 CTX & GATE CONTROL



Timer 0, Timer 1 Operation Modes

The operating mode of Timer 0 and Timer 1 is determined by the value of the M1x and M0x bits in the TMOD register. The table below summarizes the four modes of operation for timers 0 and 1.

TABLE 45: TIMER/COUNTER MODE DESCRIPTION SUMMARY

M1	M0	Mode	Function
0	0	**Mode 0**	**13-bit Timer / Counter**, with 5 lower bits in TL0 or TL1 register and bits in TH0 or TH1 register (for timer 0 and timer 1, respectively). The 3 high order bits of TL0 and TL1 are held at 0.
0	1	**Mode 1**	**16-bit Timer / Counter**
1	0	**Mode 2**	**8-bit auto reload Timer / Counter**. The reload value is kept in TH0 or TH1, while TL0 or TL1 is incremented every machine cycle. When TLx overflows, a value from THx is copied to TLx.
1	1	**Mode 3**	If Timer 1 M1 and M0 bits are set to 1, Timer 1 stops. If Timer 0 M1 and M0 bits are set to 1, Timer 0 acts as two independent 8-bit Timers / Counters.

Mode 0, 13-bit Timer/Counter

Mode 0 operation is the same for Timer 0 and Timer 1.

In Mode 0, the timer is configured as a 13-bit counter that uses bits 0-4 of the TLx register and all 8 bits of the THx register. The timer run bit (TRx) of the TCON SFR starts the timer. The value of the CTx bit defines if the timer will operate as a timer (CTx = 0), deriving its source from the system clock, or if the timer will count the high to low transitions (CTx = 1) that occur on the external timer input pin (TxIN). When the 13-bit count increments from 1FFFh (all ones) to all zeros, the TF0 (or TF1) bit will be set in the TCON SFR.

The state of the upper 3 bits of the TLx register is indeterminate in Mode 0 and must be masked when the software evaluates the register's contents.

Timers 0, Timer 1: Mode 0 - Overflow Rate (Hz)

CTx = 0

$$\text{Timer overflow rate (Hz)} = \frac{f_{SYSCLK}}{12 \times [8192-(THx, TLx)]}$$

CTx = 1

$$\text{Timer overflow rate (Hz)} = \frac{f_{TxIN}}{[8192-(THx,TLx)]}$$

Mode 1 (16-bit)

The Mode 1 operation is the same for Timer 0 and Timer 1. In Mode 1, the timer is configured as a 16-bit counter. Besides rollover at FFFFh, Mode 1 operation is the same as Mode 0.

FIGURE 16 : TIMER 0 MODE 0 & MODE 1

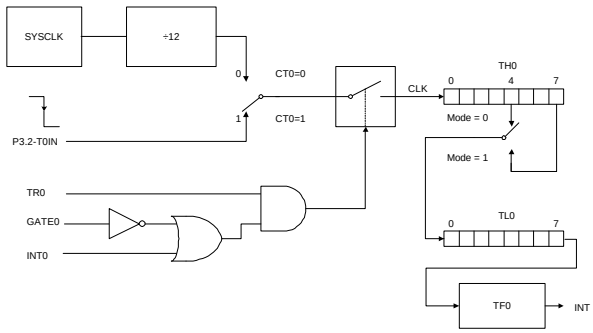
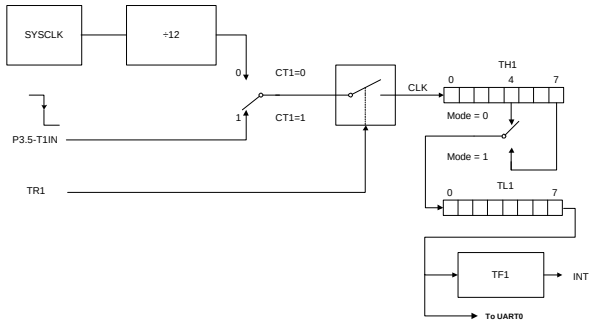


FIGURE 17: TIMER 1 MODE 0 & MODE 1



The Timer 0 and Timer 1 overflow rate in Mode 1 can be calculated using the following equations:

Timers 0, Timer 1: Mode 1 - Overflow Rate (Hz)

CTx = 0

$$\text{Timer overflow rate (Hz)} = \frac{f_{SYSCLK}}{12 \times [65536-(THx, TLx)]}$$

CTx = 1

$$\text{Timer overflow rate (Hz)} = \frac{f_{TxIN}}{[65536-(THx, TLx)]}$$

Mode 2 (8-bit)

The operation of Mode 2 is the same for Timer 0 and Timer 1. In Mode 2, the timer is configured as an 8-bit counter, with automatic reload of the start value. The LSB of the timer register, TLx, is the counter itself and the MSB portion of the timer (THx) stores the timer reload value.

The Mode 2 counter control is the same as for Mode 0 and Mode 1. However, in Mode 2, when TLx rolls over from FFh, the value stored in THx is reloaded into TLx.

FIGURE 18 : TIMER 0 MODE 2

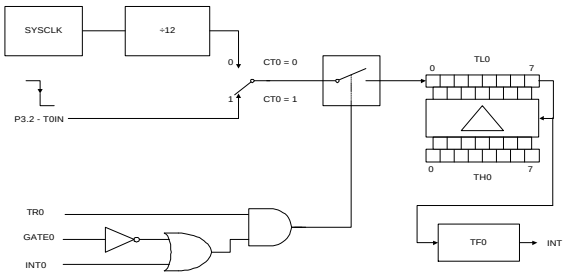
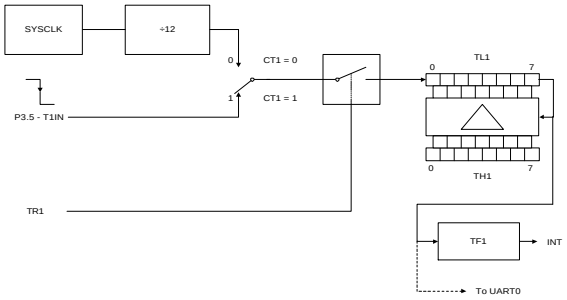


FIGURE 19: TIMER 1 MODE 2



The Timer 0 and Timer 1 overflow rate in Mode 2 can be calculated using the following equations:

Timers 0, Timer 1: Mode 2 - Overflow Rate (Hz)
CTx = 0
Timer overflow rate (Hz) = $\frac{f_{SYSCLK}}{12 \times [256-(THx)]}$
CTx = 1
Timer overflow rate (Hz) = $\frac{f_{TxIN}}{[256--(THx)]}$

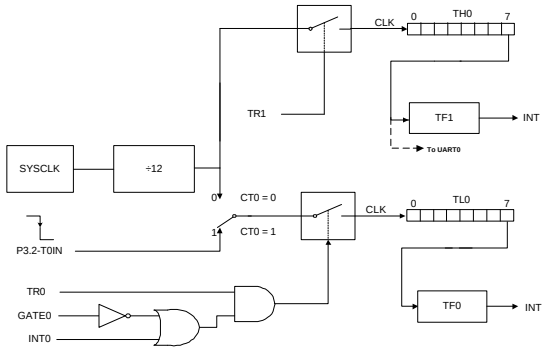
Using the Timer 1 as Baud Rate generator

Using Timer 1 in Mode 2 is recommended as the best approach when using Timer 1 as the UART0 baud rate generator.

Mode 3 (2 x 8-bit)

In Mode 3, Timer 0 operates as two 8-bit counters and Timer 1 stops counting and holds its value.

FIGURE 20: TIMER0, TIMER 1 STRUCTURE IN MODE 3USING



The Timer 0 overflow rate in Mode 3 can be calculated using the following equations:

Timers 0, Timer 1: Mode 3 - Overflow Rate (Hz)
TH0, CTx = 0 or 1
Timer overflow rate (Hz) = $\frac{f_{SYSCLK}}{12 \times 256}$
TL0, CTx = 0
Timer overflow rate (Hz) = $\frac{f_{SYSCLK}}{12 \times 256}$
TL0, CTx = 1
Timer overflow rate (Hz) = $\frac{f_{TxIN}}{256}$

In Mode 3, the values present in the TH1 and TL1 registers and CT1 control bits have no impact on the timer operation.

Timer 0 and Timer 1 Interrupts

Timer 0 and Timer 1 each have dedicated interrupt vectors located at:

- **000Bh** for the **Timer 0**
- **001Bh** for the **Timer 1**

The natural priority of Timer 0 is higher than that of Timer 1.

The following table summarizes the interrupt control and flag bits associated with the Timer 0 and Timer 1 interrupts.

Bit Name	Location	Description
EA	IEN0.7	General interrupt control bit 0, Interrupt Disabled 1, Enabled Interrupt active
T0IE	IEN0.1	Timer 0 Overflow Interrupt 1 = Enable 0 = Disable
T1IE	IEN0.3	Timer 1 Overflow Interrupt 1 = Enable 0 = Disable
TF0	TCON.5	TF0 Flag is set when Timer 0 Overflow occurs. Automatically cleared when Timer 0 interrupt is serviced. This flag can also be cleared by software
TF1	TCON.7	TF1 Flag is set when Timer 1 Overflow occurs. Automatically cleared when Timer 1 interrupt is serviced. This flag can also be cleared by software

Setting Up Timer 0 Example

To use Timer 0, first setup the interrupt and then configure the module. This is described in the following code example.

Sample C code to set up Timer 0:

```
//---------------------------------------------------------------------------
// Sample C code to setup Timer 0
//---------------------------------------------------------------------------
// (…) PROGRAM INITIALIZATION OMITTED

AT 0X0100 VOID MAIN(VOID){

// INTERRUPT + TIMER 0 SETUP
IEN0 |= 0x80;                   // ENABLE ALL INTERRUPTS
IEN0 |= 0x02;                   // ENABLE INTERRUPT TIMER 0
TMOD = 0x02;                    // TIMER 0 MODE 2
TCON = 0x10;                    // START TIMER 0

DO{}WHILE(1);           //WAIT FOR TIMER 0 INTERRUPT

}//END OF MAIN()

//---------------------------------------------------------------------------
// INTERRUPT FUNCTION

VOID INT_TIMER_0 (VOID) INTERRUPT 1
        {
IEN0 &= 0X7F;                   // DISABLE ALL INTERRUPTS
        /*-----------------------*/
        /*Put Interrupt code here*/
        /*-----------------------*/
IEN0 |= 0x80;                   // Enable all interrupts
}
//---------------------------------------------------------------------------
```

Setting Up Timer 1 Examples

The following code provides an example of how to configure Timer 1 (the first part of the code is the interrupt setup and module configuration, while the second part is the interrupt function).

Example1: Delay function

```
//---------------------------------------------------------------------------
// Sample C code using the Timer 1: Delay function
//---------------------------------------------------------------------------
VOID DELAY1MS(UNSIGNED CHAR DLAIS) {
        IDATA UNSIGNED CHAR X=0;
        TMOD = 0x10;
        TL1 = 0x33;
        TH1 = 0xFB;
        ;//TIMER1 RELOAD VALUE FOR
        TCON = 0x40;

        WHILE (DLAIS > 0)
        {
  DO{
  X=TCON;
  X= X&0X80;
  }WHILE(X==0);

        TCON = TCON&0X7F;
        TL1 = 0x33;
        TH1 = 0xFB;
        ;//TIMER1 RELOAD VALUE FOR

        DLAIS = DLAIS-1;
        }
}//END OF DELAY 1MS
```

Example2: Timer 1 interrupt example

```
//---------------------------------------------------------------------------
// Sample C code using the Timer 1:  Interrupt
//---------------------------------------------------------------------------
//  (…) PROGRAM INITIALIZATION OMITTED
at 0xo100 void main(void){

// TIMER 1 setup

IEN0 |= 0x80;                   // Enable all interrupts
IEN0 |= 0x08;                   // Enable interrupt Timer1
TMOD = 0x20;                    // Timer 1 mode 2
TCON = 0x40;                    // Start Timer 1
TL1 = 0xFC;                     // Timer1 offset

do {
        }while(1);              //Wait Timer 1 interrupt

}//end of main() function
//---------------------------------------
// Timer 1 Interrupt function
//---------------------------------------
void int_timer_1 (void) interrupt 3
{
IEN0 &= 0x7F;                   // Disable all interrupts

/* Put Interrupt code here*/

IEN0 |= 0x80;                   // Enable all interrupts
}
```

Timer 2

The VMX51C1016 Timer 2 and its associated peripherals include the following capabilities:

- o 16-bit timer
- o 16-bit auto-reload timer
- o Compare and capture units
- o 8/16 PWM outputs

TABLE 46: (TL2) TIMER 2, LOW BYTE - SFR CCH

7	6	5	4	3	2	1	0
TL2 [7:0]							

TABLE 47: (TH2) TIMER 2, HIGH BYTE - SFR CDH

7	6	5	4	3	2	1	0
TH2 [7:0]							

Figure 21 shows the Timer 2/compare and capture unit block diagram. The following paragraphs will describe how these blocks work.

Timer 2 Registers

Timer 2 consists of a 16-bit register, whose upper and lower bytes are accessible via two independent SFR registers (TL2, TH2).

TABLE 48: (TL2) TIMER 2 LOW BYTE - SFR CCH

7	6	5	4	3	2	1	0
TL2 [7:0]							

TABLE 49: (TH2) TIMER 2 HIGH BYTE - SFR CDH

7	6	5	4	3	2	1	0
TH2 [7:0]							

Timer 2 Control Register

Most of Timer 2's control is accomplished via the T2CON register located at SFR address C8h.

The T2CON register controls:

- o T2 clock source prescaler
- o T2 count size (8/16-bits)
- o T2 reload mode
- o T2 input selection

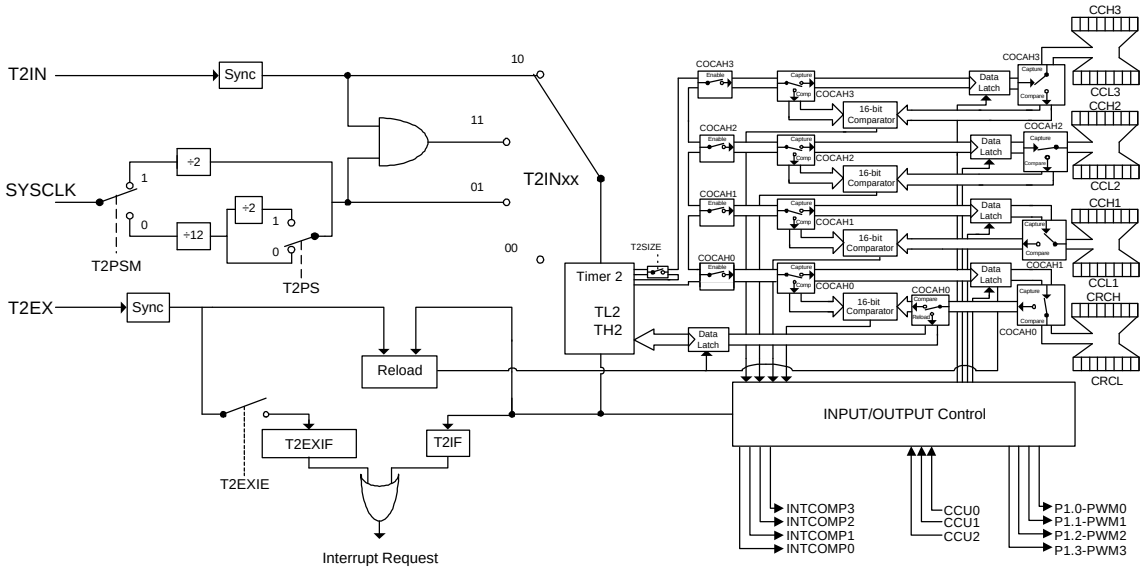
TABLE 50: (T2CON) TIMER 2 CONTROL REGISTER -SFR C8H

7	6	5	4
T2PS	T2PSM	T2SIZE	T2RM1

3	2	1	0
T2RM0	T2CM	T2IN1	T2IN0

Bit	Mnemonic	Function
7	T2PS	Prescaler select bit: 0 = Timer 2 is clocked with 1/12 of the oscillatory frequency 1 = Timer 2 is clocked with 1/24 of the oscillatory frequency
6	T2PSM	0 = Prescaler 1 = clock/2
5	T2SIZE	Timer 2 Size 0 = 16-bit 1 = 8-bit
4	T2RM1	Timer 2 reload mode selection
3	T2RM0	0X = Reload disabled 10 = Mode 0 11 = Mode 1
2	T2CM	Timer 2 compare mode selection 0 = Mode 0 1 = Mode 1
1	T2IN1	Timer 2 input selection
0	T2IN0	00 = Timer 2 stops 01 = Input frequency f/2, f/12 or f/24 10 = Timer 2 is incremented by external signal at pin T2IN 11 = Internal clock is gated to the T2IN input.

FIGURE 21: TIMER 2 AND COMPARE/CAPTURE UNIT



Timer 2 Clock Sources

As previously stated, Timer 2 can operate in timer mode, in which case it derives its source from the system clock (SYSCLK), or it can be configured as an event counter where the high to low transition on the T2IN input causes Timer 2 to increment.

The T2IN0 and T2IN1 bits of the T2CON register serve to define the selected Timer 2 input and the operating mode (see the following table):

TIMER 2 CLOCK SOURCE

T2IN1	T2IN0	Selected Timer 2 input
0	0	Timer 2 Stop
0	1	Standard Timer mode using internal clock with or without prescaler
1	0	External T2IN pin clock Timer2
1	1	Internal Clock is gated by the T2IN input When T2IN = 0, the Timer2 stop

When in timer mode, Timer 2 derives its source from the system clock and the CLKDIVCTRL register will affect Timer 2's operation.

Timer 2 Stop

When both the T2IN1 and T2IN0 bits are set to 0, Timer 2 is in STOP mode.

Timer 2 Operating Modes

When the T2IN1 bit is set to 0 and the T2IN0 bit is set to 1, the Timer 2 register may or may not derive its source from the internal pre-scaled clock, depending on the T2PSM bit value.

Event Counter Mode

When operating in event counter mode, the timer is incremented as soon as the external signal T2IN transitions from a 1 to a 0. A sample of the T2IN input is taken at every machine cycle. Timer 2 is incremented in the cycle following the one in which the transition was detected.

Gated Timer Mode

In the gated timer mode, the internal clock, which serves as the Timer 2 clock source, is gated by the external signal T2IN. In other words, when T2IN is high, the internal clock is allowed to pass through the AND gate. A low value of T2IN will disable the clock pulse. This allows an external device to control the Timer 2 operation or to use Timer 2 to monitor the duration of an event.

Timer 2 Clock Prescaler

When Timer 2 is configured to derive its clock source from the system clock, the clock prescaling value can be controlled by software using the t2psm and T2PS bits of the T2CON register.

The different system clock prescaling values are shown in the table below:

T2PSM	T2PS	Timer 2 input clock
1	X	SYSCLK / 2
0	0	SYSCLK / 12
0	1	SYSCLK / 24

Timer 2 Count Size

Timer 2 can be configured to operate in 8-bit or 16-bit format. The T2SIZE bit of the T2CON register selects the Timer 2 count size.

- If T2SIZE = 0, Timer 2 size is 16 bits
- If T2SIZE = 1, Timer 2 size is 8 bits

Timer 2 Reload Modes

The Timer 2 reload mode is selected by the T2RM1 and T2RM0 bits of the T2CON register. The following figure shows the reload operation.

Timer 2 must be configured as a 16-bit timer/counter for the reload modes to be operational by clearing the T2SIZE bit.

Timer 2 Mode 0

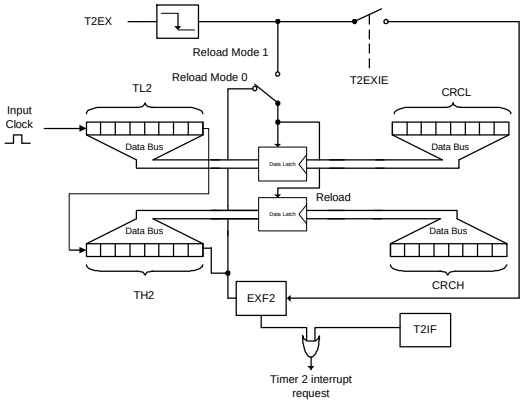
When the timer overflows, the T2IF overflow flag is set. Concurrently, this overflow causes Timer 2 to be reloaded with the 16-bit value contained in the CRCx register, (which has been preset by software). This reload operation will occur during the same clock cycle in which T2IF was set.

Timer 2 Mode 1

In Mode 1, a 16-bit reload from the CRCx register on the falling edge of T2EX occurs. This transition will set T2EXIF if T2EXIE is set. This action will cause an interrupt (providing that the Timer 2 interrupt is enabled) and the T2IF flag value will not be affected.

The value of T2SIZE does not affect reload in Mode 1. Also, the reload operation is performed independently of the state of the T2EXIE bit.

FIGURE 22: TIMER 2 RELOAD MODE



Timer 2 Overflows and Interrupts

Timer 2's interrupt is enabled when the Timer 2 counter, the T2IF flag, is set and a Timer 2 interrupt occurs.

A Timer 2 interrupt may also be raised from T2EX if the T2EXIE bit of the IEN1 register is set.

The exact source of a Timer 2 interrupt can be verified by checking the value of the T2IF and T2EXIF bits of the IRCON register.

Timer 2's interrupt vector is located at address 002Bh.

Timer 2 Setup Example

To use Timer 2, the user must first set up and configure the module (see the following code example).

```
//-------------------------------------------------------------------------
// Sample C code to setup Timer 2
//-------------------------------------------------------------------------
// (…) PROGRAM INITIALIZATION OMITTED

at 0x100 void main(void){

// TIMER 2 & Interrupt setup
DIGPWREN = 0x80;                    // Enable Timer2,
T2CON = 0x01;                       // Set timer 2 to OSC/12
TL2 = 0xE0;
TH2 = 0xFF;

IEN0 |= 0x80;                       // Enable all interrupts
IEN0 |= 0x20;                       // Enable interrupt Timer 2

        do{                         //wait for Timer 2 interrupt
        }while(1);

}//end of main()

//-------------------------------------------------------------------------
// Timer 2 Interrupt Function
//-------------------------------------------------------------------------
void int_timer_2 (void) interrupt 5
{
IEN0 &= 0x7F;                       // Disable all interrupts

        /*-------------------------*/
        /*Interrupt code here*/
        /*-------------------------*/

IEN0 |= 0x80;                       // Enable all interrupts
}
```

Timer 2 Special Modes

For general timing/counting operations, the VMX51C1016 Timer 2 includes four compare and capture units that can be used to monitor specific events and drive PWM outputs. Each compare and capture unit provides three specific operating modes that are controlled by the CCEN register:

- Compare modes enable
- Capture on write into CRCL/CCLx registers
- Capture on transitions at CCU input pins level

TABLE 51: (CCEN) COMPARE/CAPTURE ENABLE REGISTER -SFR C9H

7	6	5	4
COCAH3	COCAL3	COCAH2	COCAL2

3	2	1	0
COCAH1	COCAL1	COCAH0	COCAL0

The CCEN register bits are grouped in pairs of COCAHx/COCALx bits. Each pair corresponds to one compare and capture unit. The compare and capture unit operating mode versus the configuration bit is described in the following table:

Bit		
Mnemonic	Mnemonic	Function
COCAH0	**COCAL0**	Compare and Capture mode for CRC register
0	0	Compare/capture disabled
0	1	Capture on a falling edge at pin CCU0 (1 cycle)
1	0	Compare enabled (PWM0)
1	1	Capture on write operation into register CRC1
COCAH1	**COCAL1**	Compare/capture mode for CC register 1
0	0	Compare/capture disabled
0	1	Capture on a rising edge at pin CCU1 (2 cycles)
1	0	Compare enabled (PWM1)
1	1	Capture on write operation into register CCL1
COCAH2	**COCAL2**	Compare/capture mode for CC register 2
0	0	Compare/Capture disabled
0	1	Capture on a rising edge at pin CCU2 (2 cycles)
1	0	Compare enabled (PWM2)
1	1	Capture on write operation into register CCL2
COCAH3	**COCAL3**	Compare/Capture mode for CC register 3
0	0	Compare/capture disabled
0	1	N/A - CCU3 not pinned out
1	0	Compare enabled (PWM)
1	1	Capture on write operation into register CCL3

This allows individual configuring and operation of each compare and capture unit..

Compare/Capture, Reload Registers

Each compare and capture unit has a specific 16-bit register accessible via two SFR addresses.

Note that the CRCHx/CRCLx registers associated with Compare/Capture Unit 0 are the only ones that can be used to perform a reload of the Timer 2 operation.

The following tables describe the different registers that may be captured or compared to the value of Timer 2.

TABLE 52: (CRCL) COMPARE/RELOAD/CAPTURE REGISTER, LOW BYTE - SFR CAH

7	6	5	4	3	2	1	0
CRCL [7:0]							

TABLE 53: (CRCH) COMPARE/RELOAD/CAPTURE REGISTER, HIGH BYTE - SFR CBH

7	6	5	4	3	2	1	0
CRCH [7:0]							

TABLE 54: (CCL1) COMPARE/CAPTURE REGISTER 1, LOW BYTE - SFR C2H

7	6	5	4	3	2	1	0
CCL1 [7:0]							

TABLE 55: (CCH1) COMPARE/CAPTURE REGISTER 1, HIGH BYTE - SFR C3H

7	6	5	4	3	2	1	0
CCH1 [7:0]							

TABLE 56: (CCL2) COMPARE/CAPTURE REGISTER 2, LOW BYTE - SFR C4H

7	6	5	4	3	2	1	0
CCL2 [7:0]							

TABLE 57: (CCH2) COMPARE/CAPTURE REGISTER 2, HIGH BYTE - SFR C5H

7	6	5	4	3	2	1	0
CCH2 [7:0]							

TABLE 58: (CCL3) COMPARE/CAPTURE REGISTER 3, LOW BYTE - SFR C6H

7	6	5	4	3	2	1	0
CCL3 [7:0]							

TABLE 59: (CCH3) COMPARE/CAPTURE REGISTER 3, HIGH BYTE - SFR C7H

7	6	5	4	3	2	1	0
CCH3 [7:0]							

Compare/Capture Data Line Width

The VMX51C1016 is capable of comparing and capturing data using data lines up to 16 bits wide. When comparing two registers or capturing one register, the T2SIZE bit of the T2CON register must be set to 1. This adjusts the line width to 8 bits.

When comparing two pairs of registers, for example, CCH1 and CCL1 to TH2 and TL2, the T2SIZE bit must be set to 0. This adjusts the line width to 16 bits.

Timer 2 Capture Modes

Timer 2 capture modes makes it possible to acquire and store the 16-bit content of Timer 2 into a capture/compare registers following a MOV SFR operation or the occurrence of an external event on one of the CCU pins, as described below:

Capture input	Timer2 Capture triggering event
CCU0	High to Low Transition on CCU0
CCU1	Low to High Transition on CCU1
CCU2	Low to High Transition on CCU2

Timer 2 capture is done without affecting the Timer 2 operation.

Each individual compare and capture unit can be configured for capture mode by configuring the appropriate bit pair of the CCEN register.

The two capture modes are:

Capture Mode 0

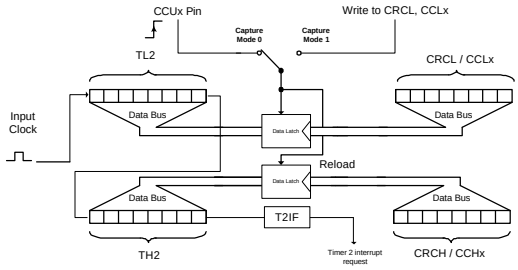
In Capture Mode 0, the transition on a given CCU pin triggers the latching of Timer 2's data into the associated compare/capture register.

Capture Mode 1

In Capture Mode 1, a capture of the Timer 2 value will occur upon writing to the low byte of the chosen capture register.

Note: On the VMX51C1016, the CCU2 and CCU3 input is not pinned out.

FIGURE 23:TIMER 2 CAPTURE MODE 0 FOR CRCL AND CRCH BLOCK DIAGRAM



The capture modes can be especially useful for external event duration calculations with the ability to latch the timer value at a given time (computation can be performed at a later time).

When operating in capture mode, the compare and capture units do not affect the VMX51C1016 interrupts.

Timer 2 Compare Modes

In compare mode, a Timer 2 count value is compared to a value that is stored in the CCHxx/CCLx or CRCHx/CRCLx registers. If the values compared match (i.e. when the pulse changes state), a compare/capture interrupt is generated, if enabled. Varying the value of the CCHx/CCLx or the CRCHx/CRCLx allows a variation of the rectangular pulse generated at the output. This variation can be used to perform pulse width modulation. (See PWM in the following section.)
To activate the compare mode on one of the four compare and capture units, the associated

COCAHx bit must be set to 1 and the associated COCALx bit must be set to 0.

When compare mode is enabled, the corresponding output pin value is controlled by the internal timer circuitry.

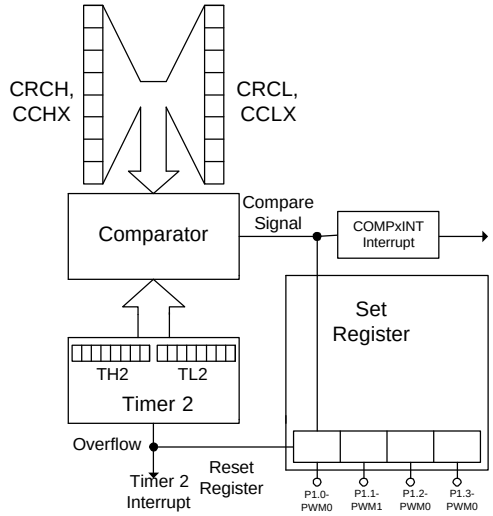
On the VMX51C1016, two compare modes are possible. In both modes, the new value arrives at Port Pin 1 in the same clock cycle as the internal compare signal is activated. The T2CM of the T2CON register defines the compare mode and is described below:

Compare Mode 0

A functional diagram of Compare Mode 0 is shown below. A comparison is made between the 16-bit value of the compare/capture registers and the TH2, TL2 registers. When the Timer 2 value exceeds the value stored in the CRCH, CRCL/CCHx and CCLx registers, a high compare signal is generated and a compare/capture interrupt is activated if enabled. If T2SIZE = 1, the comparison is made between the TL2 and CRCL/CCLx registers.

This compare signal is then propagated to the corresponding P1.x Pin(s) and to the associated COMPINTx interrupt (if enabled). The corresponding P1.x pin is reset when a Timer 2 overflow occurs.

FIGURE 24: TIMER 2 COMPARE MODE 0 BLOCK DIAGRAM

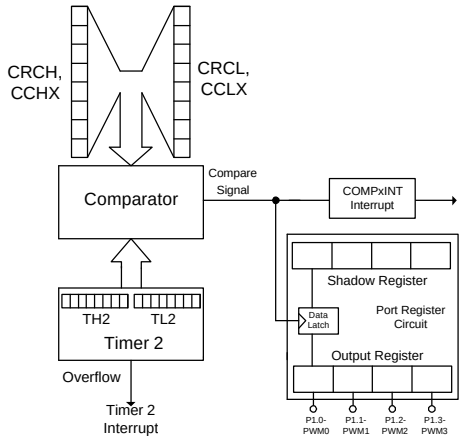


Compare Mode 1

When a given compare capture unit is operating in Mode 1, any write operations to the corresponding output register of the port P1.x (x=0 to3) will not appear on the physical port pin until the next compare match occurs.
Like Compare Mode 0, the compare signal in Mode 1 can also generate an interrupt (if enabled).

The figure below shows the operating structure of a given capture and compare unit operating in Compare Mode 1.

FIGURE 25: TIMER 2 COMPARE MODE 1 BLOCK DIAGRAM

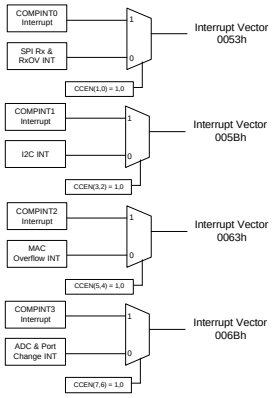


Timer 2 Compare Mode Interrupt

Configuration of the compare and capture units for the “Compare Mode” through the CCEN register impacts on the interrupt structure of the VMX51C1016. In that specific mode each compare and capture unit controls one interrupt line.

When using the PWM output device, care must be exercised to avoid other peripheral interrupts from being blocked by this mechanism.

FIGURE 26: COMPARE AND CAPTURE UNIT INTERRUPT CONTROL



Using Timer 2 for PWM Outputs

Configuring the compare and capture units in Compare Mode 0 allows PWM output generation on the Port1 I/O pins. This mode can be used for PWM applications, such as:

- o D/A conversion
- o Motor control
- o Light control, etc.

When a specific compare and capture unit is configured for this mode, its associated I/O pin is reserved for this operation only and any write operations to the associated I/O pin of the P1 register will have no affect.

The following table shows the association between the compare and capture units, associated registers and I/O pins.

TABLE 60: COMPARE AND CAPTURE UNIT PWM ASSOCIATION

Compare Capture Unit	Registers	I/O pin
0	CRCH / CRCL	P1.0
1	CCH1 / CCL1	P1.1
2	CCH2 / CCL2	P1.2
3	CCH3 / CCL3	P1.3

PWM signal generation is derived from the comparison result between the values stored in the compare and capture registers and the Timer 2 value.

When a digital value is written into one of the compare and capture registers, a comparison is performed between this register and the Timer 2 value (providing that Timer 2 is in compare mode). As long as the value present in the compare and capture register is greater than the Timer 2 value, the compare unit will output a logic low.

When the value of Timer 2 equals the value of the compare and capture register, the compare unit will change from a logic low to a logic high.

The clock source for the PWM is derived from Timer 2, which is incremented at every signal pulse of the appropriate source. The source is selected by the T2IN1 and T2IN0 bits of the T2CON register.

The T2SIZE bit of the T2CON register allows configuring the PWM output for 8 or 16-bit operation. The Timer 2 size affects all the PWM outputs.

When the Timer 2 size is 8 bits, the comparison is performed between Timer 2 and the LSB of the compare and capture unit register. The resulting PWM resolution is 8 bits.

When the Timer 2 size is configured for a 16-bit operation, the comparison is performed between Timer 2 and the contents of the entire compare and capture unit register. The resulting PWM resolution is 16 bits, but the PWM frequency is consequently low.

When the system clock is used as the Timer 2 clock source, the PWM output frequency equals the Timer 2 overflow rate. Note that the CLKDIVCTRL register content affects the Timer 2 operation and, thus, the PWM output frequency.

Fosc	T2CON T2PSM	T2CON T2PS	T2CON T2SIZE	Freq PWM
14.74MHz	1	X	0	112.5Hz
	1	x	1	28.8KHz
	0	0-12	1-8	4.8KHz
	0	0-12	0-16	18.8Hz
	0	1-24	1-8	2.4KHz
	0	1-24	0-16	9.38Hz

The duty cycle of the PWM output is proportional to the ratio of the compare and capture unit register's content versus Timer 2's maximum number of cycles before overflow: 256 or 65536 depending on the T2SIZE bit value.

PWM Duty Cycle Calculation: 8-bit
PWM duty cycle CCU0 (%) = 100% x $\frac{(256\text{-}CRCL)}{256}$
PWM duty cycle CCU1-3 (%) = 100% x $\frac{(256\text{-}CCLx)}{256}$

PWM Duty Cycle Calculation: 16-bit
PWM duty cycle CCU0 (%) = 100% x $\frac{65536–(CRCH, CRCL)}{(CRCH, CRCL)}$
PWM duty cycle CCU1-3 (%) = 100% x $\frac{65536–(CRCH, CRCL)}{(CRCH, CRCL)}$

PWM Configuration Example

The following example shows how to configure the Timer 2 based PWM in 8-bit mode.

```
(…)
DIGPWREN = 0x80;                //ENABLE TIMER 2 MODULE
T2CON = 0x61;                   //BIT 7 - Select 0=1/12, 1=1/24 of Fosc
                                //BIT 6 - T2 clk source:  0 = Presc,
                                1=clk/2
                                //BIT 5 - T2 size: 0=16-bit, 1=8-bit
                                //BIT 4,3 - T2 Reload mode:
                                //BIT 2   - T2 Compare mode
                                //BIT 1,0 - T2 input select: 01= input
                                derived from osc.

//WHEN THE PWM IS CONFIGURED IN 16-BIT FORMAT, THE PWM OUTPUT
FREQUENCY IS GIVEN BY //THE FOLLOWING EXPRESSION:
// PWM Freq = [(FOSC/2)] / 65536
// WITH A 14.7456MHZ CRYSTAL PWM FREQUENCY = 112.5HZ

//When the PWM is configured in 8-bit its output freq  = [(Fosc/2)] / 256
//USING A 14.7456MHZ CRYSTAL PWM FREQUENCY = 28.8KHZ

CCEN = 0x0AA;                   //Enable Compare on 4 PWM outputs

// In 16-bit PWM resolution both LSB and MSB of compare unit are used

//In 8-bit PWM Resolution, only the LSB of compare units are used
// and MSB is kept to 00h

CRCL = 0x0E6;                   //PWM0 duty = [(256-CRCL)/256]
x100%
CRCH = 0x000;                   //E6h => 10.1%
CCL1 = 0x0C0;                   //PWM1 duty  = [(256-CCL1)/256]
x100%
CCH1 = 0x000;                   //C0h => 25%
CCL2 = 0x080;                   //PWM2 duty = [(256-CCL2)/256] x100%
CCH2 = 0x000;                   //80h => 50%
CCL3 = 0x033;                   //PWM3 duty = [(256-CCL3)/256] x100%
CCH3 = 0x000;                   //33h => 80%
P1PINCFG = 0x0F;                //Configure P1 LSQ as output to enable
                                PWM

(…)
```

Using the PWM as a D/A Converter

One of the popular uses of the PWM is to perform D/A conversion by low pass filtering its modulated square wave output. The greater the duty cycle of the square wave, the greater the DC value is at the output of the low pass filter and vice versa.

Variations in the duty cycle of the PWM when filtered can therefore generate arbitrary waveforms.

Serial UART Interfaces

The VMX51C1016 includes two serial UART interface ports (UART0 and UART1). Each serial port has a 10-bit timer devoted to baud rate generation.

Both serial ports can operate in full duplex asynchronous mode. The VMX51C1016 also includes a double buffer, enabling the UART to accept an incoming word before the software has read the previous value.

UART0 Serial Interface

The operation of the UART0 on the VMX51C1016 is similar to a standard 8051 UART.

UART0 can derive its clock source from a 10-bit dedicated baud rate generator or from the Timer 1 overflow.

UART0 transmit and receive buffers are accessed through a unique SFR register (S0BUF).

UART0 S0BUF has a double buffering feature on reception, which allows the UART to accept an incoming word before the software has read the previous value from the S0BUF.

TABLE 61: (S0BUF) SERIAL PORT 0, DATA BUFFER - SFR 99H

7	6	5	4	3	2	1	0
S0BUF [7:0]							

UART0 Control Register

UART0 configuration is performed mostly via the S0CON SFR register located at address 98h.

TABLE 62: (S0CON) SERIAL PORT 0, CONTROL REGISTER - SFR 98H

7	6	5	4
S0M0	S0M1	MPCE0	R0EN

3	2	1	0
T0B8	R0B8	T0I	R0I

Bit	Mnemonic	Function
7	S0M0	Sets Serial Port Operating Mode See Table
6	S0M1	
5	MPCE	1 = Enables the multiprocessor communication feature.
4	R0EN	1 = Enables serial reception. Cleared by software to disable reception.
3	T0B8	The 9th transmitted data bit in Modes 2 and 3. Set or cleared by the CPU, depending on the function it performs (parity check, multiprocessor communication etc.)
2	R0B8	In Modes 2 and 3, it is the 9th data bit received. In Mode 1, if sm20 is 0, RB80 is the stop bit. In Mode 0, this bit is not used. Must be cleared by software.
1	T0I	Transmit interrupt flag set by hardware after completion of a serial reception. Must be cleared by software.
0	R0I	Receive interrupt flag set by hardware after completion of a serial reception. Must be cleared by software.

UART0 Operating Modes

UART0 can operate in four distinct modes, which are defined by the SM0 and SM1 bits of the S0CON register (see the following table):

TABLE 63: SERIAL PORT 0 MODES

SM0	SM1	MODE	DESCRIPTION	BAUD RATE
0	0	0	Shift Register	Fosc/12
0	1	1	8-bit UART	Variable
1	0	2	9-bit UART	Fclk/32 or /64
1	1	3	9-bit UART	Variable

**Note that the speed in mode 2 depends on SMOD bit in the Special Function Register PCON when SMOD = 1 fclk/32

UART0 - Mode 0

In this mode, pin RX0 is used as an input and an output, while TX0 is used only to output the shift clock. For an operation in this mode, 8 bits are transmitted with the LSB as the first bit. In addition, the baud rate is fixed at 1/12 of the crystal oscillator frequency. In order to initialize reception in this mode, the user must set bits R0I and R0EN in the S0CON register to 0 and 1, respectively. Note that in other modes, when R0EN=1, the interface begins to receive data.

UART0 - Mode 1

In this mode, the RX0 pin serves uniquely as an input and the TX0 pin serves as a serial output and no external shift clock is used. In Mode 0, 10 bits are transmitted:

- One*** Start bit (active low)
- 8 bits of data starting with the LSB;
- One logical high Stop bit

The Start bit synchronizes data reception with the 8 bits of received data available in the S0BUF register. Reception is complete once the Stop bit sets the R0B8 flag in the S0CON register.

UART0 - Mode 2

In this mode, the RX0 pin is used as an input and as an output, while the TX0 pin is used to output the shift clock. In Mode 2, 11 bits are transmitted or received. These 11 bits consist of:

- One logic low Start bit
- 8 bits of data (LSB first)
- One programmable 9th bit
- One logic high Stop bit

The 9th bit is used for parity. In the case of data transmission, bit TB80 of the S0CON is output as the 9th bit. For reception, the 9th bit is captured in the RB80 bit of the S0CON register.

UART0 - Mode 3

Mode 3 is essentially identical to Mode 2, the difference being that in Mode 3, the internal baud rate generator or Timer 1 can be used to set the baud rate.

UART0 - Baud Rate Generator Source

As mentioned previously, the UART0 baud rate clock can be sourced from either Timer 1 or the 10-bit dedicated baud rate generator

Selection between these two clock sources is enabled via the BAUDSRC bit of the U0BAUD register (see the following table).

TABLE 64: (U0BAUD) UART0 BAUD RATE SOURCE SELECT - SFR D8H

7	6	5	4	3	2	1	0
BAUDSRC	-	-	-	-	-	-	-

7	BAUDSRC	Baud rate generator clock source 0 = Timer 1 1 = Use UART0 dedicated baud rate generator
,6:0	-	-

Using the UART0 dedicated baud rate generator frees up Timer 1 for other uses.

The S0RELH and S0REL registers are used to store the 10-bit reload value of the UART0 baud rate generator.

TABLE 65: (S0RELL) SERIAL PORT 0, RELOAD REGISTER, LOW BYTE - SFR 96H

7	6	5	4	3	2	1	0
S0RELL [7:0]							

TABLE 66: (S0RELH) SERIAL PORT 0, RELOAD REGISTER, HIGH BYTE - SFR 97H

7	6	5	4	3	2	1	0
S0RELH [15:8]							

The following equations should be used to calculate the reload value for the SOREL register (examples follow).

Mode 3: For BAUDSRC=1

$$SOREL = 1024 - \frac{2^{SMOD} \times f_{clk}}{64 \times \text{Baud Rate}}$$

$$\text{Baud Rate} = \frac{2^{SMOD} \times f_{clk}}{64 \times (1024 - S0REL)}$$

TABLE 67: SERIAL 0 BAUD RATE SAMPLE VALUES BAUDSRC = 1, SMOD = 1

Desired Baud Rate	S0REL @ f_{clk}= 11.059 MHz	S0REL @ f_{clk}= 14.746 MHz
500.0 kbps	-	-
460.8 kbps	-	3FFh
230.4 kbps	-	3FEh
115.2 kbps	3FDh	3FCh
57.6 kbps	3FAh	3F8h
19.2 kbps	3EEh	3E8h
9.6 kbps	3DCh	3D0h
2.4 kbps	370h	340h
1.2 kbps	2E0h	280h
300 bps	-	-

TABLE 68: SERIAL 0 BAUD RATE SAMPLE VALUES BAUDSRC =1, SMOD = 0

Desired Baud Rate	S0REL @ f_{clk}= 11.059 MHz	S0REL @ f_{clk}= 14.746 MHz
115.2 kbps	-	3FEh
57.6 kbps	3FDh	3FCh
19.2 kbps	3F7h	3F4h
9.6 kbps	3EEh	3E8h
2.4 kbps	3B8h	3A0h
1.2 kbps	370h	340h
300 bps	1C0h	100

Timer 1 can also be used as the baud rate generator for the UART0. Set BAUDSRC to 0 and assign Timer 1's output to UART0.

When the baud rate clock source is derived from Timer 1, the baud rate and the timer reload values can be calculated using the following formulas (examples follow):

TABLE 69: EQUATION TO CALCULATE BAUD RATE FOR SERIAL 0

Serial 0: mode 1 and 3

Mode 1: ForU0BAUD.7=0 (standard mode)

$$\text{Baud Rate} = \frac{2^{SMOD} \times f_{clk}}{32 \times 12 \times (256 - TH1)}$$

$$TH1 = 256 - \frac{2^{SMOD} \times f_{clk}}{32 \times 12 \times \text{Baud Rate}}$$

TABLE 70: UART0 BAUD RATE SAMPLE VALUES BAUDSRC =0, SMOD = 1

Desired Baud Rate	TH1 @ f_{clk}= 11.059 MHz	TH1 @ f_{clk}= 14.746 MHz
115.2 kbps	-	-
57.6 kbps	FFh	-
19.2 kbps	FDh	FCh
9.6 kbps	FAh	F8h
2.4 kbps	E8h	E0h
1.2 kbps	D0h	C0h
300 bps	40h	-

TABLE 71:UART0 BAUD RATE SAMPLE VALUES BAUDSRC =0, SMOD = 0

Desired Baud Rate	TH1 @ f_{clk}= 11.059 MHz	TH1 @ f_{clk}= 14.746 MHz
115.2 kbps	-	-
57.6 kbps	-	-
19.2 kbps	-	FEh
9.6 kbps	FDh	FCh
2.4 kbps	F4h	F0h
1.2 kbps	E8h	E0h
300 bps	A0h	80h

Example of UART0 Setup and Use

In order to use UART0, the following operations must be performed:

- Enable UART0 interface
- Set I/O pad direction TX=output, RX=Input
- Enable reception (if required)
- Configure the UART0 controller S0CON

The following are configuration and transmission code examples for UART0.

```
//-------------------------------------------------------------------------------------//
// UART0 CONFIG with S0REL
//
// Configure the UART0 to operate in RS232 mode at 19200bps
// with a crystal of 14.7456MHz
//
//-------------------------------------------------------------------------------------//
void uart0ws0relcfg()
        {
        P3PINCFG |= 0x01;               // pads for uart 0
        DIGPWREN |= 0x01;               // enable uart0/timer1
        S0RELL = 0xF4;                  //com speed = 19200bps
        S0RELH = 0x03;
        S0CON = 0x50;           // Uart0 in mode1, 8 bit, var. baud rate
        U0BAUD = 0x80;          //Set S0REL is source for UART0
                                //Baud rate clock

        }//end of uart0ws0relcfg() function

//-------------------------------------------------------------------------------------//
// UART0 CONFIG with Timer 1
//
// Configure the UART0 to operate in RS232 mode at 19200bps
// with a crystal of 14.746MHz
//
//-------------------------------------------------------------------------------------//
void uart0wTimer1cfg()
        {
        P3PINCFG |= 0x01;               // pads for uart0
        DIGPWREN |= 0x01;               // enable uart0/timer1
        TMOD &= 0x0F;
        TMOD =0x20;             //Set Timer 1, Gate 0, Mode 2
        TH1 = 0xFE;                     //Com Speed = 19200bps
        TCON &= 0x0F;
        TCON =0x40;                     //Start Timer 1
        U0BAUD = 0x00;                  //Set Timer 1 Baud rate
                                        //generator for UART0

        PCON = 0x00;                    //Set SMOD = 0
        S0CON = 0x50;                   // Config Uart0 in mode 1,
                                        //8 bit, variable baud rate
        }//end of uart1Config() function

//-------------------------------------------------------------------------------------//
// Txmit0()
//
// One Byte transmission on UART0
//-------------------------------------------------------------------------------------//

// - Constants definition
sbit UART_TX_EMPTY = USERFLAGS^1;

void txmit0( unsigned char charact){
        S0BUF = charact;
        USERFLAGS = S0CON;
        //Wait TX EMPTY flag to be raised
        while (!UART_TX_EMPTY) {USERFLAGS = S0CON;}S0CON =

        //clear both R0I & T0I bits
        S0CON & 0xFD;
        }//end of txmit0() function
```

See the interrupt section for examples of how to setup UART0 interrupts.

UART1 Serial Interface

The UART1 serial interface is based on a subset of UART0. It provides two operating modes and its clock source is derived exclusively from a dedicated 10-bit baud rate generator.

The UART1 transmit and receive buffers are accessed via a unique SFR register named S1BUF.

TABLE 72: (S1BUF) SERIAL PORT 1, DATA BUFFER - SFR C1H

7	6	5	4	3	2	1	0
S1BUF [7:0]							

As is the case with UART0, UART1 has a double buffering feature in order to avoid an overwriting of the receive register.

UART1 Control Register

UART1 is controlled by the S1CON register. The following table provides a description of the UART 1 control register.

TABLE 73: (S1CON) SERIAL PORT 1, CONTROL REGISTER - SFR C0H

7	6	5	4
S1M	Reserved	MPCE1	R1EN

3	2	1	0
T1B8	R1B8	T1I	R1I

Bit	Mnemonic	Function
7	S1M	Operation Mode Select
6	Reserved	-
5	MPCE1	1 = Enables multiprocessor communication feature.
4	R1EN	If set, enables serial reception. Cleared by software to disable reception.
3	T1B8	The 9th transmitted data bit in mode A. Set or cleared by the CPU, depending on the function it performs (parity check, multiprocessor communication, etc.)
2	R1B8	In Mode A, it is the 9th data bit received. In Mode B, if SM21 is 0, RB81 is the stop bit. Must be cleared by software.
1	T1I	Transmit interrupt flag, set by hardware after completion of a serial transfer. Must be cleared by software
0	R1I	Receive interrupt flag, set by hardware after completion of a serial reception. Must be cleared by software

UART1: Operating Modes

The VMX51C1016 UART1 provides two operating modes, Mode A and Mode B, which provide 8 or 9-bit operation, respectively.

Below is a summary table of operating modes of UART1.

TABLE 74: UART1 MODES

SM	MODE	DESCRIPTION	BAUD RATE
0	A	9-bit UART	Variable
1	B	8-bit UART	Variable

UART1 - Mode A

In this mode, 11 bits are transmitted or received. These 11 bits are composed of:

- A Start bit (logic low)
- 8-bits of data (LSB first)
- A programmable 9th bit
- A Stop bit (logic low)

As in modes 2 and 3 of UART0, the 9th bit is used for parity control. For data transmission, the TB81 bit of the S1CON register holds the 9th bit. In the case of reception, the 9th bit will be captured into the R1B8 bit of the S1CON register.

UART1 - Mode B

In this mode, 10 bits are transmitted and consist of:

- A Start bit (logic low)
- 8-bits of data (LSB first)
- A Stop bit (logic low)

Received data (8 bits) is read via the S1BUF register. Reception is completed once the Stop bit sets the R1B8 flag in the S1CON register.

UART1 - Baud Rate Generator

As mentioned previously, the UART1 clock source is derived from a dedicated 10-bit baud rate generator module.

The S1REL registers are used to adjust the baud rate of UART 1.

TABLE 75: (S1RELL) UART1, RELOAD REGISTER, LOW BYTE - SFR BEH

7	6	5	4	3	2	1	0
S1RELL [7:0]							

TABLE 76: (S1RELH) UART 1, RELOAD REGISTER, HIGH BYTE - SFR BFH

7	6	5	4	3	2	1	0
S1RELH [7:0]							

The following formulas are used to calculate the baud rate, S1RELL and S1RELH values:

Serial 1

$$\text{Baud Rate} = \frac{f_{clk}}{32 \times (1024 - \text{S1REL})}$$

Note: S1REL.9-0 = S1RELH.1-0 + S1RELL.7-0

$$\text{S1REL} = 1024 - \frac{f_{clk}}{32 \times \text{Baud Rate}}$$

TABLE 77: SERIAL 1 BAUD RATE SAMPLE VALUES

Desired Baud Rate	**S1REL @ f_{clk}= 11.059 MHz**	**S1REL @ f_{clk}= 14.746 MHz**
500.0 kbps	-	-
460.8 kbps	-	3FFh
230.4 kbps	-	3FEh
115.2 kbps	3FDh	3FCh
57.6 kbps	3FAh	3F8h
19.2 kbps	3EEh	3E8h
9.6 kbps	3DCh	3D0h
2.4 kbps	370h	34Fh
1.2 kbps	2E0h	280h

Setting Up and Using UART1

In order to use UART1, the following operations must be performed:

- Enable UART1 interface
- Set I/O pad direction TX= output, RX=Input
- Enable reception (if required)
- Configure UART1 controller S1CON

Example of UART1 Setup and Use

The following are C code examples of UART1 configuration, serial byte transmission and interrupt usage.

```
//-------------------------------------------------------------------------------//
// UART1 CONFIG
//
// Configure the UART1 to operate in RS232 mode at 115200bps
// with a crystal of 14.746MHz
//-------------------------------------------------------------------------------//
void uart1Config(void)
        {
        P0PINCFG |= 0x04;       // pads for uart 1
        DIGPWREN |= 0x02;       // enable uart1
        S1RELL = 0xFC;          // Set com speed = 115200bps
        S1RELH = 0x03;
        S1CON = 0x90;           // Mode B, receive enable
        }//end of uart1Config() function


//-------------------------------------------------------------------------------//
// TXMIT1  -- Transmit one byte on the UART1
//-------------------------------------------------------------------------------//
void txmit1( unsigned char charact){
        S1BUF = charact;
        USERFLAGS = S1CON;
        while (!UART_TX_EMPTY) {USERFLAGS = S1CON;}
                                        //Wait TX EMPTY flag
        S1CON = S1CON & 0xFD;           //clear both R1I & T1I bits
        }//end of txmit1() function


//-------------------------------------------------------------------------------//
// Interrupt configuration
//-------------------------------------------------------------------------------//

IEN0 |= 0x80;                   // Enable all interrupts
IEN2 |= 0x01;                   // Enable interrupt UART 1



//-------------------------------------------------------------------------------//
// Interrupt function
//-------------------------------------------------------------------------------//

void int_serial_1 (void) interrupt 16
{
IEN0 &= 0x7F;                   // Disable all interrupts

        /*----------------------*/
        /*Interrupt code here*/
        /*----------------------*/

if (S1CON&0x01==0x01)
        {
        S1CON &= 0xFE;          // Clear RI (it comes
                                // before T1I)
        }
        else
        {
        S1CON &= 0xFD;          // Clear T1I
        }
        IEN0 |= 0x80;}          // Enable all interrupts
        }
}
/-------------------------------------------------------------------
```

UART1 Driven Differential Transceiver

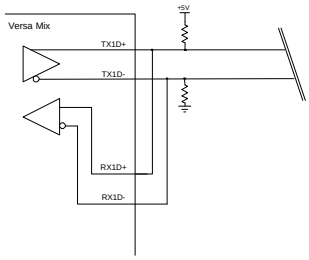
The VMX51C1016 includes a differential transceiver compatible with the standard J1708/RS-485. These are driven by UART1.

The transceiver's signals are differential, providing high electrical noise immunity. The differential interface is capable of transferring/receiving data over hundreds of feet of twisted pair wires.

A number of devices can be connected in parallel to the differential bus in order to implement a multi-drop network. The number of devices that can be networked depends on bus length and configuration.

The admissible common mode voltage range of the differential interface is –2.0 to +7.0 volts. When implementing this type of transmission network over long distances in noisy environments, appropriate protection is recommended to prevent the common mode voltage from causing any damage to the VMX51C1016.

FIGURE 28: DIFFERENTIAL INTERFACE (RS485 CONFIG)



From a software point of view, the differential transceiver is viewed as a differential UART.

The differential transceiver I/Os are connected to UART1 of the VMX51C1016, therefore, communication parameters such as the data length, communication speed, etc. are managed by the UART1 peripheral interface/registers.

Using the UART 1 Differential Transceiver

To use the differential transceiver interface, the following operations must be performed:

- Enable both UART1 and the differential interface by setting bits 1 and 2 of the DIGPWREN
- Configure UART1's operating mode via the S1CON register
- Set the baud rate via the S1RELH and S1RELL registers
- Enable UART1's interrupt (if required)

Use UART1's S1BUF register to transmit and receive data through the differential transceiver. If the P0.2 pin is configured as an output, the signal corresponding to the TX1 signal of UART1 will appear on this pin (note that the P0.3-RX1 pin can be used as a regular digital output).

When the transceiver is connected in half-duplex mode (RX1D+ connected to TX1D+ and RX1D- connected to TX1D-) and the UART1 interrupts are enabled, careful management of the UART1 interrupts will be required as every byte transmitted will generate a local Rx interrupt.

Differential Interface Use Example

The following code provides an example of configuration and use of the VMX51C1016 differential interface.

```
#pragma SMALL
#pragma UNSIGNEDCHAR
#include <vmixreg.h>

// --- function prototypes
void txmit1( unsigned char charact);
void uart1differential(void);

// - global variables

// - Constants definition
sbit UART_TX_EMPTY = USERFLAGS^1;

code char irq0msg[]="Ramtron inc";

//-------------------------------------------------------------------------------------//
//                              MAIN FUNCTION
//-------------------------------------------------------------------------------------//

 at 0x0100 void main (void) {

// Enable and configure the UART1
        uart1differential();        //Config UART1 diff interface

// Warning:  The Clock Control circuit does affect the dedicated baud rate
//           generator S0REL, S1REL and Timer1 operation


//*** Configure the interrupts
        IEN0 |= 0x81;               //Enable interrupts + Ext. 0 interrupt
        IEN2 |= 0x01;               //Enable UART1 Interrupt

        Txmit1('A');                //Transmit one character on UART1

        do
        {

        }while(1);                  //Wait for UART1 Rx interrupt

}// End of main()...


//-------------------------------------------------------------------------------------//
// UART1 Differential interface interrupt
//
// In this example, the source of UART1 interrupt would be caused
// by bytes reception on the differential interface
//-------------------------------------------------------------------------------------//
void int_uart1 (void) interrupt 16 {
        unsigned char charact;

        IEN0 &= 0x7F;

        // -- Put you code here…

        S1CON = S1CON & 0xFC;       //clear both R1I & T1I bits
        IEN0 |= 0x80;               // enable all interrupts

}// end of uart1 INTERRUPT

//-------------------------------------------------------------------------------------//
// EXT INT0 interrupt
//
//
// when the External interrupt 0 is triggered A Message string  is sent over the
// the serial UART1
//-------------------------------------------------------------------------------------//
void int_ext_0 (void) interrupt 0 {

        int x=0;
        idata unsigned char     cptr=0x01;

        IEN0 &= 0x7F;
                                //disable ext0 interrupt

        cptr = cptr-1;
        while( irq0msg[cptr] != '\n')
        //Send a text string over the differential interface
                {
                txmit1( irq0msg[cptr]);
                cptr = cptr +1;
                }

        IEN0 = 0x81;
                                //Enable all interrupts + int_0

//-------------------------------------------------------------------------------------//
//---------------------------- Individual Functions -----------------------------------//
//-------------------------------------------------------------------------------------//

//-------------------------------------------------------------------------------------//
// UART1 DIFFERENTIAL CONFIG
//
// Configure the UART1 differential interface to operate in
// RS232 mode at 115200bps with a crystal of 14.746MHz
//
//-------------------------------------------------------------------------------------//
void uart1differential(void)
        {
        DIGPWREN |= 0x06;       // enable uart1 & differential transceiver
        P0PINCFG |= 0x04;       // pads for uart1
        P0PINCFG = 0x00;

        S1RELL = 0xFC;          // Set com speed = 115200bps
        S1RELH = 0x03;
        S1CON = 0x90;           // Mode B, receive enable
        }//end of uart1differential()  function

//-------------------------------------------------------------------------------------//
// TXMIT1
//
// Transmit one byte on the UART1 Differential interface
//
//-------------------------------------------------------------------------------------//
void txmit1( unsigned char charact){
        S1BUF = charact;
        USERFLAGS = S1CON;

        //Wait TX EMPTY flag to be raised

        while (!UART_TX_EMPTY) {USERFLAGS = S1CON;}

        S1CON = S1CON & 0xFD;           //clear both R1I & T1I bits
        }//end of txmit1() function
```

SPI Interface

The VMX51C1016 SPI peripheral is a highly configurable and powerful interface enabling high speed serial data exchange with external devices such as A/Ds, D/As, EEPROMs, etc.

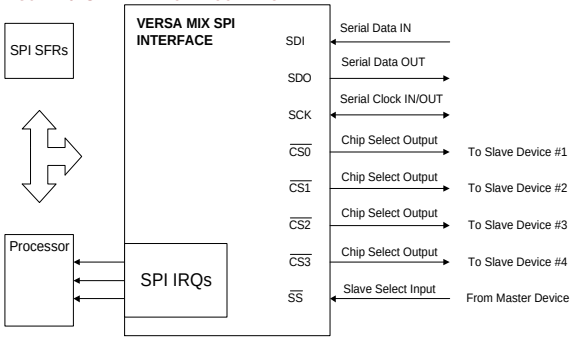
The SPI interface can operate as either a master or a slave device. In master mode, it can control up to four slave devices connected to the SPI bus.

The following lists a number of the VMX51C1016's SPI features:

- Permits synchronous serial data transfers
- Transaction size is configurable from 1 to 32 bits and more
- Full duplex support
- SPI modes 0, 1, 2, 3, 4-supported (full clock polarity and phase control)
- Up to four slave devices can be connected to the SPI bus when configured in master mode
- Slave mode operation
- Data transmission speed is configurable
- Double 32-bit buffers in transmission and reception
- 3 dedicated interrupt flags
  - TX-Empty
  - RX Data Available
  - RX Overrun
- Automatic/Manual control of the chip selects lines
- SPI operation is not affected by the clock control unit

The following provides a block diagram view of the SPI interface.

FIGURE 29: SPI INTERFACE BLOCK DIAGRAM



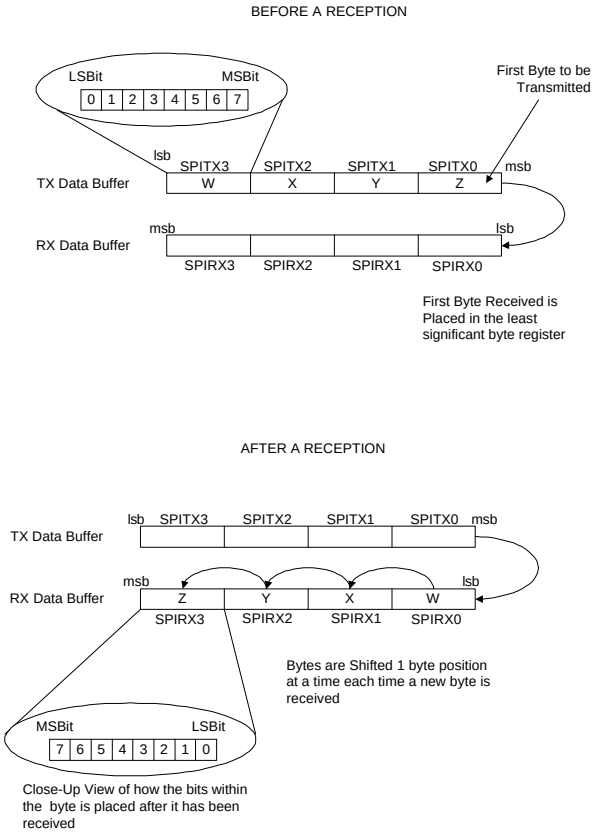
SPI Transmit/Receive Buffer Structure

When receiving data bytes, the first byte received is stored in the SPIRX0 buffer. As bits continue to arrive, the data already present in the buffer is shifted toward the least significant byte end of the receive registers.

For example (see the following figure), assume the SPI is about to receive four consecutive bytes of data: W, X, Y and Z, where the first byte received is byte W. The first received byte (W) will be placed in the SPIRX0 register. Upon reception of the next byte (X), the contents of SPIRX0 will be shifted into SFR register SPIRX1 and byte X will be placed in the SPIRX0 registers. Following this same procedure, bytes W, X, Y and Z will end up in RX data buffer registers SPIRX0, SPIRX1, SPIRX2 and SPIRX3, respectively.

The case where the SDO and SDI pins are shorted together is represented in the following diagram.

FIGURE 30 : SPI INTERFACE RECEIVE TRANSMIT SCHEMATIC



When using the SPI interface, it is important to keep in mind that a transmission is started when the SPIRX3TX0 register is written to.

From an SFR point of view, the transmission and reception buffers of the SPI interface occupy the following addresses:

TABLE 78: (SPIRX3TX0) SPI DATA BUFFER, LOW BYTE - SFR E1H

7	6	5	4	3	2	1	0
SPIRX3TX0 [7:0]							

Bit	Mnemonic	Function
7-0	SPITX0	SPI Transmit Data Bits 7:0
	SPIRX3	SPI Receive Data Bits 31:24

TABLE 79: (SPIRX2TX1) SPI DATA BUFFER, BYTE 1 - SFR E2H

7	6	5	4	3	2	1	0
SPIRX2TX1 [15:8]							

Bit	Mnemonic	Function
15:8	SPITX1	SPI 1 Transmit Data Bits 15:8
	SPIRX2	SPI Receive 1 Data Bits 22:16

TABLE 80: (SPIRX1TX2) SPI DATA BUFFER, BYTE 2 - SFR E3H

7	6	5	4	3	2	1	0
SPIRX1TX2 [23:16]							

Bit	Mnemonic	Function
22:16	SPITX2	SPI Transmit Data Bits 22:16
	SPIRX1	SPI Receive Data Bits 15:8

TABLE 81: (SPIRX0TX3) SPI DATA BUFFER, HIGH BYTE - SFR E4H

7	6	5	4	3	2	1	0
SPIRX0TX3 [31:24]							

Bit	Mnemonic	Function
31:24	SPITX3	SPI Transmit Data Bits 31:24
	SPIRX0	SPI Receive Data Bits 7:0

SPI Control Registers

The SPI control registers are used to define:

- SPI operating speed (master mode)
- Active chip select output (master mode)
- SPI clock phase (master/slave modes)
- SPI clock polarity (master/slave modes)

TABLE 82: (SPICTRL) SPI CONTROL REGISTER - SFR E5H

7	6	5	4
SPICK [2:0]			SPICS_1

3	2	1	0
SPICS_0	SPICKPH	SPICKPOL	SPIMA_SL

Bit	Mnemonic	Function
7:5	SPICK[2:0]	SPI Clock control 000 = OSC Ck Div 2 001 = OSC Ck Div 4 010 = OSC Ck Div 8 011 = OSC Ck Div 16 100 = OSC Ck Div 32 101 = OSC Ck Div 64 110 = OSC Ck Div 128 111 = OSC Ck Div 256
4:3	SPICS[1:0]	Active CS line in Master Mode 00 = CS0- Active 01 = CS1- Active 10 = CS2- Active 11 = CS3- Active
2	SPICKPH	SPI Clock Phase
1	SPICKPOL	SPI Clock Polarity 0 – CK Polarity is Low 1 – CK Polarity is High
0	SPIMA_SL	Master / -Slave 1 = Master 0 = Slave

SPI Operating Speed

Three bits in the SPICTRL register serve to adjust the communication speed of the SPI interface.

SPICK[2:0] Div Ratio	Fosc = 14.74MHz	Fosc = 11.059MHz
Clk Div 2	7.37 MHz	5.53 MHz
Clk Div 4	3.68 MHz	2.76 MHz
Clk Div 8	1.84 MHz	1.38 MHz
Clk Div 16	922 kHz	691 kHz
Clk Div 32	461 kHz	346 kHz
Clk Div 64	230 kHz	173 kHz
Clk Div 128	115 kHz	86 kHz
Clk Div 256	57.6 kHz	43.2 kHz

SPI Master Chip Select Control

When the SPI is configured in master mode, the value of the SPICS[1:0] bits define which chip select pins will be active during the transaction.

The following sections describe how the SPI clock polarity and phase affects the read and write operations of the SPI interface.

SPI Operating Modes

The SPI interface can operate in four distinct modes defined by the value of the SPICKPH and SPICKPOL bits of the SPICTRL register.

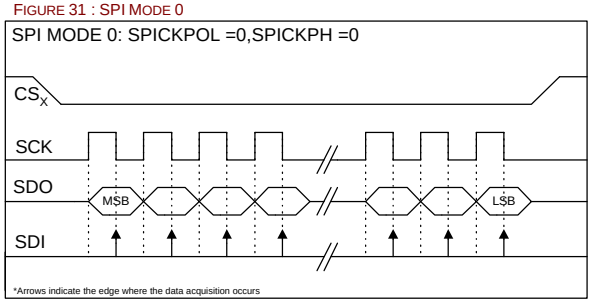
SPICKPH defines the SPI clock phase and SPICKPOL defines the clock polarity for data exchange.

SPICKPOL bit value	SPICKPH bit value	SPI Operating Mode
0	0	SPI Mode 0
0	1	SPI Mode 1
1	0	SPI Mode 2
1	1	SPI Mode 3

SPI Mode 0

- Data is placed on the SDO pin at the rising edge of the clock.
- Data is sampled on the SDI pin at the falling edge of the clock

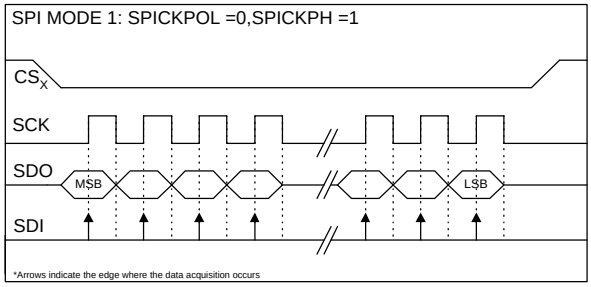
FIGURE 31 : SPI MODE 0



SPI Mode 1

- Data is placed on the SDO pin at the falling edge of the clock
- Data is sampled on the SDI pin at the rising edge of the clock

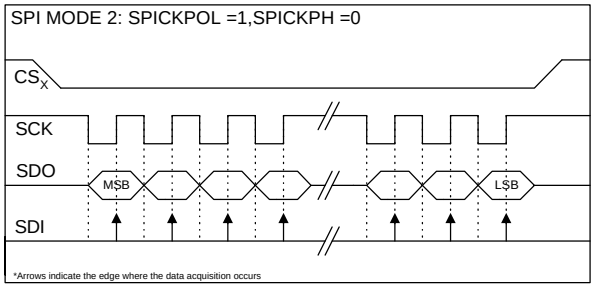
FIGURE 32: SPI MODE 1



SPI Mode 2

- Data is placed on the SDO pin at the falling edge of the clock
- Data is sampled on the SDI pin at the rising edge of the clock

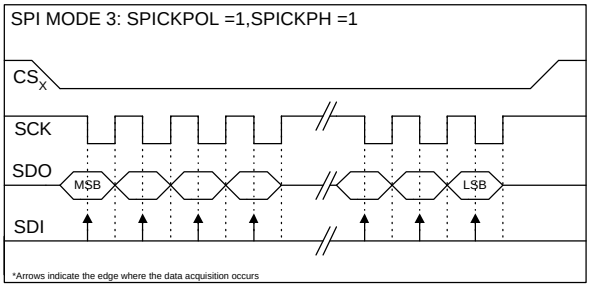
FIGURE 33: SPI MODE 2



SPI Mode 3

- Data is placed on the SDO pin at the rising edge of the clock.
- Data is sampled on the SDI pin at the falling edge of the clock

FIGURE 34: SPI MODE 3



SPI Transaction Size

Many microcontrollers only allow a fixed SPI transaction size of 8 bits. However, most devices requiring SPI control demand transactions of more than 8 bits, giving way to alternate inefficient methods of dealing with SPI transactions.

The VMX51C1016 SPI interface includes a transaction size control register (SPISIZE) that enables different sized transactions to be performed. The SPI interface also automatically controls the chip select line.

The following table describes the SPISIZE register:

TABLE 83: (SPISIZE) SPI SIZE CONTROL REGISTER - SFR E7H

7	6	5	4	3	2	1	0
SPISIZE[7:0]							

Bit	Mnemonic	Function
7:0	SPISIZE[7:0]	Value of the SPI packet size

The following formula is used to calculated the SPI transaction size:

For SPISIZE from 0 to 31:

SPI Transaction Size = [SPISIZE + 1]

For SPISIZE from 32 to 255*:

SPI Transaction Size = [SPISIZE*8 - 216]

An SPI transaction size greater than 32 bits is possible when using the VMX51C1016 SPI interface, however, data packets of this size require careful management of the associated interrupts in order to avoid buffer overwrites.

SPI Interrupts

The SPI interface has three associated interrupts:

- SPI RX Overrun
- SPI RX Data Available
- SPI TX Empty

The SPIRXOVIE, SPIRXAVIE and SPITXEMPIE bits of the SPICONFIG register allow individual enabling of the above interrupt sources at the SPI interface level.

At the processor level, two interrupt vectors are dedicated to the SPI interface:

- SPI RX data available and overrun interrupt
- SPI TX empty interrupt

In order to have the processor jump to the associated interrupt routine, one or both of these interrupts in the IEN1 register must be enabled, and the EA bit of the IEN0 register must be set (see interrupt section).

TABLE 84: (SPICONFIG) SPI CONFIG REGISTER - SFR E6H

7	6	5	4
SPICSLO	-	FSONICS3	SPILOAD

3	2	1	0
-	SPIRXOVIE	SPIRXAVIE	SPITXEMPIE

Bit	Mnemonic	Function
7	SPICSLO	Manual CS up (Master mode) 0 = The CSx goes low when transmission begins and returns to high when it ends. 1 = The CSx stays low after transmission ends. The user must clear this bit for the CSx line to return high.
6	-	-
5	FSONCS3	This bit sends the frame select pulse on CS3.
4	SPILOAD	This bit sends load pulse on CS3.
3	-	-
2	SPIRXOVIE	SPI Receiver overrun interrupt enable.
1	SPIRXAVIE	SPI Receiver available interrupt enable.
0	SPITXEMPIE	SPI Transmitter empty interrupt enable.

The SPIIRQSTAT register contains the interrupts flags associated with the SPI interface.

Monitoring these bits enables polling the control of the SPI interface.

TABLE 85: (SPIIRQSTAT) SPI INTERRUPT STATUS REGISTER - SFR E9H

7	6	5	4
-	-	SPITXEMPTO	SPISLAVESEL

3	2	1	0
SPISEL	SPIOV	SPIRXAV	SPITXEMP

Bit	Mnemonic	Function
7:6	-	-
5	SPITXEMPTO	Flag that indicates that we have not reloaded the transmit buffer fast enough (only used for packets greater than 32 bits.).
4	SPISLAVESEL	Slave Select “NOT” (SSN)
3	SPISEL	This bit is the result of the logical AND operation between CS0, CS1, CS2 and CS3. (Indicates if one chip is selected.)
2	SPIOV	SPI Receiver overrun
1	SPIRXAV	SPI Receiver available
0	SPITXEMP	SPI Transmit buffer is ready to receive mode data. It does not flag that the transmission is completed.

SPI Manual Chip Select Control

In some applications, manual control of the active select line can be useful. Setting the SPICSLO bit of the SPICONFIG register forces the active chip select line to stay low when the SPI transaction is completed in master mode. When the SPICSLO bit is cleared, the chip selected line returns to its active state.

SPI Manual Load Control

The SPI can generate a LOAD pulse on the CS3 pin when the SPILOAD bit is set. This is useful for some D/A converters and avoids having to use a separate I/O pin for this purpose.

SPI Frame Select Control

It is also possible to generate a positive pulse on the CS3 pin of the SPI interface by setting the FSONCS3 bit of the SPICONFIG register. This feature can be used to generate a frame select signal required by some DSP compatible devices without requiring the use of a separate I/O pin.

Note that when both the SPILOAD and FSONCS3 are selected, the internal logic gives priority to the frame select pulse.

SPI Interface to 16-bit D/A Example

The following is a code for executing 16-bit transfers over the SPI interface.

```
//-------------------------------------------------//
// VMIX_SPI_to_dac_interface. c    //
//-------------------------------------------------//
//
//  This demonstration program show the how to interface a 16-bit D/A
//  to the VMX51C1016 SPI interface.
//
#pragma SMALL
#include <vmixreg.h>

// --- function prototypes

//Function Prototype:  Send Data to the 16 bit D/A

void send16bitdac( unsigned char valhigh, unsigned char vallow);

// Bit definition
sbit SPI_TX_EMPTY = USERFLAGS^0;

//-------------------------------------------------------------------------------//
//                          MAIN FUNCTION                                        //
//-------------------------------------------------------------------------------//
 at 0x0100 main (void) {

                unsigned char dacvall=0;                //LSB of current DAC value
                unsigned char dacvalh=0;                //MSB of current DAC value
                DIGPWREN |= 0x08;                       //ENABLE SPI INTERFACE

                //*** Initialise the SPI interface ****
                P2PINCFG |= 0x68;                       // config I/O port to allow the SPI
                                                        //interface to access the pins
// In this application we only need to configure the 5 upper bit of P2PINCFG
                        // P2PINCFG bit 7 - SDIEN  = 0 -> INPUT (NOT USED)
                        // P2PINCFG bit 6 - SDOEN  = 1 -> OUTPUT TO DAC SDI PIN
                        // P2PINCFG bit 5 - SCKEN  = 1 -> OUTPUT TO DAC SCK PIN
                        // P2PINCFG bit 4 - SSEN   = 0 -> INPUT (NOT USED)
                        // P2PINCFG bit 3 - CS0EN  = 1 -> OUTPUT TO DAC CS PIN
                        // P2PINCFG bit 2 - CS1EN  = 0 -> INPUT (NOT USED)
                        // P2PINCFG bit 1 - CS2EN  = 0 -> INPUT (NOT USED)
                        // P2PINCFG bit 0 - CS3EN  = 0 -> INPUT (NOT USED)



                SPICTRL = 0x25;
                        // SPI ctrl: OSC/16, CS0, phase=0, pol=0, master
                        // SPICK    BIT 7:5  = 001 -> SPI CLK SPEED = OSC/2
                        // SPICS    BIT 4:3  = 00  -> CS0 LINE IS ACTIVE
                        // SPICKPH  BIT 2    = 1 SPI CLK PHASE
                        // SPICKPOL BIT 1    = 0 SPI CLOCK POLARITY
                        // SPIMA_SL BIT 0    = 1 -> SET SPI IN MASTER MODE

                SPICONFIG = 0x00;
                        // SPI CONFIG: auto CSLO, no FS, NO Load, clear IRQ flags
                        // SPICSLO    BIT 7 = 0 AUTOMATIC CHIP SELECT CONTROL
                        // UNSUSED    BIT 6 = 0
                        // FSONCS3  BIT 5 = 0 Do not send FrameSelect Signal on CS3
                        // SPILOAD    BIT 4 = 0 do not Sen the Low pulse on CS3
                        // UNUSED     BIT 3 = 0
                        // SPIRXOVIE  BIT 2 = 0 Dont enable SPI RX Overrun IRQ
                        // SPIRXAVIE  BIT 1 = 0 Dont enable SPI RX AVAILLABLE IRQ
                        // SPITXEMPIE BIT 0 = 0 Dont Enable  SPI TX EMPTY IRQ

                SPISIZE = 0x0F;                 // SPI SIZE:  16-bits

                // GENERATE A TRIANGLE WAVE ON THE DAC OUTPUT

                while(1){
                do{
                dacvall = dacvall + 1;
                if( dacvall==0xff)
                                {
                                dacvalh = dacvalh +1;
                                dacvall = 0x00;
                                }

                send16-bitdac( dacvalh, dacvall);
                }while( (dacvall != 0xff) && (dacvalh != 0xff) );

                do{
                dacvall = dacvall - 1;
                if( dacvall==0x00)
                                {
                                dacvalh = dacvalh - 1;
                                dacvall = 0xff;
                                }
                send16-bitdac( dacvalh, dacvall);
                }while( (dacvall != 0x00) && (dacvalh != 0x00) );
                };


}// End of main()...

//-------------------------------------------------------------------------//
//  Send16-bitdac - Send data to 16 bit D/A Converter    //
//-------------------------------------------------------------------------//
void send16-bitdac( unsigned char valhigh, unsigned char vallow){

//              USERFLAGS = 0x00;
//              while(!SPI_TX_EMPTY){USERFLAGS = SPIIRQSTAT;}

                SPIRX2TX1 = vallow;     //Put LSB of value in SPI transmit buffer
                                        //-> trigger transmission
                SPIRX3TX0 = valhigh;    //Put MSB of value in SPI transmit buffer
                                        //-> trigger transmission

                do{             //Wait SPI TX empty flag to be activated
                                USERFLAGS = P2;
                                USERFLAGS &= 0x08;
                                }while( USERFLAGS == 0);
}//end of send16-bitdac
```

SPI Interrupt Example

The following provides an example of basic SPI configuration and interrupt handling.

```
//-------------------------------------------------------------------------//
// Sample C code for SPI RX & TX interrupt set-up
//-------------------------------------------------------------------------//
//
#pragma SMALL
#include <vmixreg.h>

at 0x0100 main (void) {


DIGPWREN = 0x08;                    // Enable SPI
P2PINCFG = 0x4F;                    // Set pads direction
SPICONFIG = 0x03;                   // Enable Rx_avail + TX_empty
SPISIZE = 0x07;                     // SPI SIZE:  8 bits


IEN0 |= 0x80;               // Enable all interrupts
IEN1 |= 0x06;               // Enable SPI Txempty + RXavail interrupt

SPIRX3TX0 = valhigh;        //Put MSB of value in SPI transmit buffer
                            //-> trigger transmission

            Do{
                        }while(1)

}//end of main()


//-------------------------------------------------------------------------//
// SPI TX Empty Interrupt function
//-------------------------------------------------------------------------//

void int_2_spi_tx (void) interrupt 9
{
IEN0 &= 0x7F;                       // Disable all interrupts

            /*-------------------------*/
            /* Interrupt code here*/
            /*-------------------------*/

IRCON &= 0xFD;                      // Clear flag SPITXIF
IEN0 |= 0x80;                       // Enable all interrupts
}

//-------------------------------------------------------------------------//
// SPI RX availlable function
//-------------------------------------------------------------------------//

void int_2_spi_rx (void) interrupt 10
{
IEN0 &= 0x7F;                       // Disable all interrupts

            /*-------------------------*/
            /* Interrupt code here*/
            /*-------------------------*/

IRCON &= 0xFB;                      // Clear flag SPIRXIF
IEN0 |= 0x80;                       // Enable all interrupts
}


//-------------------------------------------------------------------------//
```

Due to the double buffering of the SPI interface, an SPI TX empty interrupt will be activated as soon as the data to be transmitted is written into the SPI interface transmit buffer. If data is subsequently written into the SPI transmit buffer before the original data has been transmitted, the TX empty interrupt will only be activated when the original data has been fully transmitted.

The SPI also includes double buffering for data reception. Once a data reception is completed, the RX interrupt is activated and the data is transferred into the SPI RX buffer. At this point, the SPI interface can receive more data. However, the processor must have retrieved the first data stream before the second data stream reception is complete, otherwise a data overrun will occur and the SPI RX overrun interrupt will be activated if enabled.

I²C Interface

The VMX51C1016 includes an I²C compatible communication interface that can be configured in master or slave mode.

I²C Control Registers

The I2CRXTX SFR register is used to retrieve and transmit data on the I²C interface.

TABLE86: (I2CRXTX) I2C DATA BUFFER - SFR DEH

7	6	5	4	3	2	1	0
I2CRXTX [7:0]							

Bit	Mnemonic	Function
7:0	I2CTX[7:0]	I2C Data Receiver / Transmitter buffer

The I2CCONFIG register serves to configure the operation of the VMX51C1016 I²C interface. The following table describes the I2CCONFIG register bits:

TABLE 87: (I2CCONFIG) I2C CONFIGURATION - SFR DAH

7	6	5	4
I2CMASKID	I2CRXOVIE	I2CRXDAVIE	I2CTXEMPIE

3	2	1	0
I2CMANACK	I2CACKMODE	I2CMSTOP	I2CMASTER

Bit	Mnemonic	Function
7	I2CMASKID	This is used to mask the chip ID when you have only two devices. Therefore in a transaction, rather that receiving the chip ID first, you will receive the first packet of data.
6	I2CRXOVIE	I2C Receiver overrun interrupt enable
5	I2CRXDAVIE	I2C Receiver available interrupt enable
4	I2CTXEMPIE	I2C Transmitter empty interrupt enable
3	I2CMANACK	1= Manual acknowledge line goes to 0 0= Manual acknowledge line goes to 1
2	I2CACKMODE	Used only with Master Rx, Master Tx, and Slave Rx. 1= Manual Acknowledge on 0= Manual Acknowledge off
1	I2CMSTOP	I2C Master receiver stops at next acknowledge phase. (read during data phase)
0	I2CMASTER	I2C Master mode enable 1= I2C interface is Master 0= I2C interface is Slave

The I2CIRQSTAT register provides the status of the I²C interface operation and monitors the I²C bus status.

TABLE 88: (I2CIRQSTAT) I2C INTERRUPT STATUS - SFR DDH

7	6	5	4
I2CGOTSTOP	I2CNOACK	I2CSDA	I2CDATACK

3	2	1	0
I2CIDLE	I2CRXOV	I2CRXAV	I2CTXEMP

Bit	Mnemonic	Function
7	I2CSGOTSTOP	This means that the slave has received a stop (this bit is read only). Reset only when the master begins a new transmission.
6	I2CNOACK	Flag that indicates that no acknowledge has been received. Is reset at the start of the next transaction
5	I2CSDA	Value of SDA line.
4	I2CDATACK	Data acknowledge phase.
3	I2CIDLE	Indicates that I2C is idle
2	I2CRXOV	I2C Receiver overrun
1	I2CRXAV	I2C Receiver available
0	I2CTXEMP	I2C Transmitter empty

The I2CCHIPID register holds the VMX51C1016 I²C interface ID as well as the status bit that indicates whether the last byte monitored on the I²C interface was destined for the VMX51C1016 or not.

The reset value of this register is 0x42, corresponding to an I²C chip ID of 0x21. The chip ID value of the VMX51C1016 can be dynamically changed by writing the desired ID value into the I2CCHIPID register (see the following table).

TABLE 89: (I2CCHIPID) I2C CHIP ID - SFR DCH

7	6	5	4	3	2	1	0
I2CID [6:0]							I2CWID

Bit	Mnemonic	Function
7:1	I2CID[6:0]	The value of this chip's ID
0	I2WID	Read only and is used only in slave mode 0: The ID received corresponds to the I2CID 1: The ID received does not correspond to the I2CID

The I2WID bit is "read only", is used only in slave mode and is an indicator of whether the transaction is targeted to the VMX51C1016 device.

I²C Clock Speed

The VMX51C1016's I²C interface communication speed is fully configurable. This provides the ability to adjust the speed to various I²C bus configurations.

Control of the I²C interface communication speed is enabled via the I2CCLKCTRL register. The following formula is used to calculate the I^2C clock frequency in master mode:

$$I^2C\ Clk = \frac{f_{osc}}{[8 \times (I2CCLKCTRL)]}$$

The table below provides I²C clock (on SCL pin) speeds for various settings of the I2CCLKCTRL register when using a 14.75MHz crystal oscillator to the drive the VMX51C1016.

I2CCLKCTRL Value	I2C Clock (SCL Value)
01h	920KHz
03h	461KHz
07h	230KHz
13h	92.1KHz
27h	46KHz
C7h	9.2KHz

When the I2C interface is configured for slave mode, the I2CCLKCTRL is not used.

TABLE 90: (I2CCLKCTRL) I2C CLOCK CONTROL - SFR DBH

7	6	5	4	3	2	1	0
I2CCLKCTRL [7:0]							

Bit	Mnemonic	Function
7:0	I2CCLKCTRL	I2C Clock speed control

I²C Interface Interrupts

The I²C interface has a dedicated interrupt vector located at address 0x5B. Three flags (see below) share the I²C interrupt vector and can be used to monitor the I²C interface status making it possible to activate the I²C interrupt.

I2CTXEMP: Is set to 1 when the transmit buffer is empty

I2CRXAV: Is set to 1 when data byte reception is completed

I2CRXOV: Is set to 1 if a new byte reception is completed before the previous data in the reception buffer is read, resulting in a data overrun

These flags can all trigger an I²C interrupt if their corresponding bit in the I2CCONFIG register is set to one.1

In the case where more than one of these flags can activate an I²C interrupt, the interrupt service routine is left to determine which condition generated the interrupt.

Note that the I2CRXAV, I2CTXEMP and I2CRXOV flags can still be polled if their corresponding interrupt enable flags are cleared. Therefore, they can still be used to monitor the status.

Master I²C Operation

In master mode, the VMX51C1016 I²C interface controls the I²C bus transfers. In order to configure the I²C interface as a master, the I2CMASTER bit of the I2CCONFIG register must be set to 1.

Once the I²C interface is configured, sending data to a slave device connected to the bus is done by writing the data into the I2CRXTX register.

Before sending data to a slave device, a byte containing the target device's chip ID and a read/write bit must be sent to it.

A master mode data read is triggered by reading the I2CRXAV (bit 1) of the I2CIRQSTAT register. The data is present on the I2CRXTX register when the I2CRXAV bit is set.

Reading the value of the I2CRXTX register resets the I2CRXAV bit. Once started, the I²C byte read process will continue until the master generates a STOP condition.

When the I²C interface is configured as a master, setting the I2CMSTOP bit of the I2CCONFIG register to 1 will result in the interface generating a STOP condition after the reception of the next byte.

In master mode, it is possible to manually control the operation of the acknowledged timing when receiving data. To do this, the I2CMANACK bit of the I2CCONFIG register must be set to 1. Once you have received a byte, you can manually control the acknowledge level by clearing or setting the I2CMANACK bit.

Note: The VMX51C1016 I²C interface is not compatible with I²C the multi-master mode.

Slave I2C Operation

The VMX51C1016 I²C interface can be configured as a slave by clearing the I2CMASTER bit of the I2CCONFIG register.

In slave mode, the VMX51C1016 has no control over the rate or the timing of the data exchange that occurs on the I²C bus. Therefore, in slave mode it is preferable to manage the transactions using the I²C interrupts.

The I2CMASKID bit, when set, will configure to the slave device mask the received ID byte and receive the data directly. This is useful when only two devices are present on the I²C bus.

Note: When the VMX51C1016 starts transmitting data in slave mode, it will continually transmit the value present in the I²C transmit register as long as the master provides the clock signal or until the master device generates a STOP condition

Errata:

The VMX1016 I^2C interface has a critical timing issue when the device is configured as a slave and transmits multiple data bytes. Single byte transmission in slave mode is not affected.

The condition arises if the master device releases the SDA line at the same time it brings the SCL line low for the acknowledge phase.

In order for the VMX1016 I^2C slave transmission to work properly for multiple bytes, the master device MUST release the SDA line AFTER the SCL negative edge.

For this reason it is not possible to have a VMX1016 device configured as an I^2C master and VMX1016 devices configured as I^2C slaves on the same I^2C bus, unless data transmitted from VMX1016 I^2C slaves to the I²C master is done one byte at a time.

I²C EEPROM Interface Example Program

The following provides an example program using the VMX51C1016 interface to perform read and write operations to an externally connected EEPROM device.

```
#pragma SMALL
#include <vmixreg.h>

// --- Function prototypes
unsigned char eeread(idata unsigned char, idata unsigned char);
void eewrite(idata unsigned char, idata unsigned char, unsigned char);

// - Global variables
idata unsigned char          irqcptr=0x00;

sbit  I2C_TX_EMPTY = USERFLAGS^0;
sbit  I2C_RX_AVAIL = USERFLAGS^1;
sbit  I2C_IS_IDLE = USERFLAGS^3;
sbit   I2C_NO_ACK = USERFLAGS^6;


//-------------------------------------------------------------------------//
//                      MAIN FUNCTION                                      //
//-------------------------------------------------------------------------//
void main (void){

                unsigned char x=0;

                DIGPWREN = 0x13;          //Enable the I2C peripheral

                //*** configure I2C Speed.
                I2CCLKCTRL = 0x013;      //…To about 100KHZ...

                //*** Configure the interrupts
                IEN0 |= 0x81;                //Enable Ext INT0 interrupt + main

    //*** infinite loop waiting for ext IRQ
                while(1){
                };

}// End of main()...

//-------------------------------------------------------------------------//
// EXT INT0 interrupt
//
// When the External interrupt 0 is triggered read and write
// operations are performed on the EEPROM
//-------------------------------------------------------------------------//
void int_ext_0 (void) interrupt 0 {

// Local variables declaration
                idata unsigned char eedata;
                idata unsigned char adrsh =0;
                idata unsigned char adrsl =0;
                idata int adrs =0;

//              IEN0 &= 0x7F;                //disable ext0 interrupt
                                             //(Masked for debugger compatibility)

                //Write irqcptr into the EEPROM at adrs 0x0100
                eewrite( 0x01,0x00,irqcptr);

                irqcptr = irqcptr + 1;       //Increment the Interrupt counter

                //Perform an EEPROM read at address 0x100
                eedata = eeread(0x01, 0x00);

                delay1ms(100);                //Debo delay for the switch on INT0
//              IEN0 = 0x81;                 // enable all interrupts + int_0 (Removed
                                             //for debugger compatibility)

}// end of EXT INT 0

//-------------------------------------------------------------------------//
//        INDIVIDUALS FUNCTIONS                                            //
//-------------------------------------------------------------------------//
//-----------------------------------------------------------------//
//   EEREAD - EEPROM Random Read  //
//-----------------------------------------------------------------//
 unsigned char eeread(idata unsigned char adrsh, idata unsigned char adrsl)
                {
                idata unsigned char x=0;
                idata unsigned char readvalue=0;

                I2CCONFIG = 0x03;
                                //I2C MASTER MODE NO INTERRUPT

                I2CRXTX = 0xA8;
                                //SEND 24LC64 ADRS + write COMMAND
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}

                I2CRXTX = adrsh;
                                //SEND 24LC64 ADRSH
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}

                I2CRXTX = adrsl;
                                //SEND 24LC64 ADRSL
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}
                USERFLAGS = 0x00;

                //wait for I2C interface to be idle
                while(!I2C_IS_IDLE){USERFLAGS = I2CIRQSTAT;}

                I2CCONFIG &= 0xFD;       //set Master Rx Stop, only 1 byte to receive

                I2CCONFIG |= 0x02;

                I2CRXTX = 0xA9;              // Chip ID read

                USERFLAGS = 0x00;
                while(!I2C_RX_AVAIL){USERFLAGS = I2CIRQSTAT;}

                readvalue = I2CRXTX;

                USERFLAGS = 0x00;
                while(!I2C_IS_IDLE){USERFLAGS = I2CIRQSTAT;}
                //Wait for I2C IDLE
                return          readvalue;

                }//End of EEREAD

//-----------------------------------------------------------------//
//  EEWRITE - EEPROM Random WRITE //
//-----------------------------------------------------------------//
void eewrite(idata unsigned char adrsh, idata unsigned char adrsl, unsigned char
eedata)
                {
                idata unsigned char x;
                I2CCONFIG = 0x01;        //I2C MASTER MODE NO INTERRUPT
                I2CRXTX = 0xA8;          //SEND EEPROM ADRS + READ
                                         //COMMAND
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}

                I2CRXTX = adrsh;         //SEND ADRSH
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}

                I2CRXTX = adrsl;         //SEND ADRSL
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}

                I2CRXTX = eedata;        //SEND 24LC64 DATA and wait
                                         //for I2C bus IDLE
                USERFLAGS = 0x00;
                while(!I2C_IS_IDLE){USERFLAGS = I2CIRQSTAT;}
                ///--Wait Write operation to end

                I2CCONFIG = 0x01;        //I2C Master Mode no Interrupt

                do{
                I2CRXTX = 0xA8;          //Send 24LC64 Adrs +read Command
                USERFLAGS = 0x00;
                while(!I2C_TX_EMPTY){USERFLAGS = I2CIRQSTAT;}
                USERFLAGS = I2CIRQSTAT;
                                }while(I2C_NO_ACK);
                delay1ms(5);             //5ms delay for EEPROM write
                }// End of EEPROM Write
```

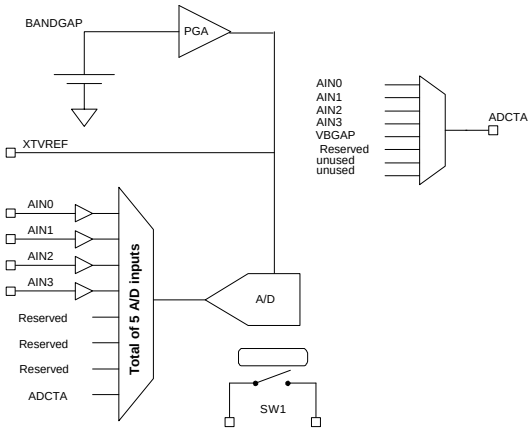
Analog Signal Path

The VMX51C1016 implements a complete single chip acquisition system by integrating the following analog peripherals:

- 12-bit, A/D converter with 5 external inputs. The ADC conversion rate is programmable up to 10KHz
- Internal bandgap reference and PGA
- Digital switch

The following figure provides a block diagram of the VMX51C1016 analog peripherals and their connections.

FIGURE 35: ANALOG SIGNAL PATH OF THE VMX51C1016



The on-chip calibrated bandgap or the external reference provides the reference for the ADC.

Analog Peripherals Power Control

Selection of the internal/external reference, the ADC control and their respective power downs are controlled via the ANALOGPWREN SFR registers.

Internal Reference and PGA

The VMX51C1016 provides a temperature calibrated internal bandgap reference coupled with a programmable gain amplifier.

The programmable gain amplifier's role is to amplify the bandgap output and bring it to 2.7 volts and to provide the drive required for the ADC reference input.

Both the bandgap and the PGA are calibrated during production and their associated calibration registers are automatically loaded with the appropriate calibration vectors when the device is reset.

The bandgap and PGA calibration vectors are stored into the BGAPCAL and PGACAL SFR registers when a reset occurs. It is possible for the user program to overwrite the contents of these registers. .

TABLE 91: (BGAPCAL) BAND-GAP CALIBRATION VECTOR REGISTER - SFR B3H

7	6	5	4	3	2	1	0
BGAPCAL [7:0]							

Bit	Mnemonic	Function
7:0	BGAPCAL	Band-gap data calibration

TABLE 92: (PGACAL) PGA CALIBRATION VECTOR REGISTER - SFR B4H

7	6	5	4	3	2	1	0
PGACAL [7:0]							

Bit	Mnemonic	Function
7:0	PGACAL	8 MSBs of PGA Calibration Vector (LSBit is on PGACAL0)

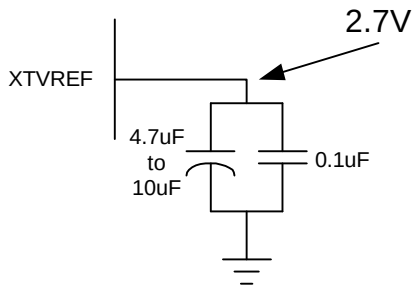
Using the VMX51C1016 Internal Reference

The configuration and setup up of the VMX51C1016 internal reference is achieved by setting bits 0 and 1 of the ANALOGPWREN register to 1. This powers-on the bandgap and the PGA, respectively.

Use of the internal reference requires the addition of two external tank capacitors on the XTVREF pin. These capacitors consist of one 4.7uF to 10uF tantalum capacitor in parallel with one 0.1uF ceramic capacitor.

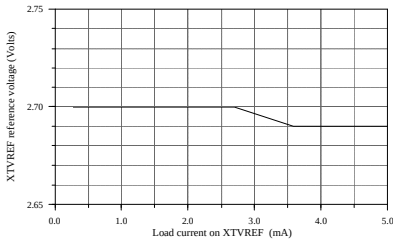
The next figure shows the connection of the tank capacitors to the XTVREF pin.

FIGURE 36: TANK CAPACITORS CONNECTION TO THE XTVREF PIN



The VMX51C1016 internal reference can also be used as an external reference,,provided that the load on the XTVREF pin is kept to a minimum. The following table shows the typical affect of loading on the XTVREF voltage.

FIGURE 37: TANK CAPACITORS CONNECTION TO THE XTVREF PIN



It is recommended that the external load on the XTVREF pin to be less than 1mA.

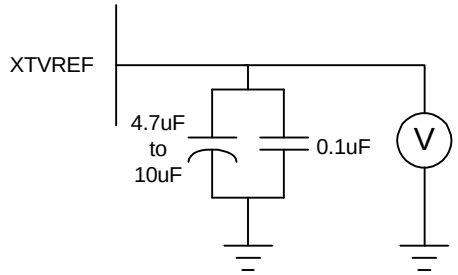
Note: A stabilization delay of more than 1ms should be provided between the activation of the bandgap, the PGA and the first A/D conversion.

Using an External Reference

An external reference can be used to drive the VMX51C1016 ADC instead of the internal reference.

The external reference voltage source can be set from 0.5 to 3.5 volts and must provide sufficient drive to operate the ADC load.

FIGURE 38: EXTERNAL REFERENCE CONNECTION TO THE XTVREF PIN



Warning:

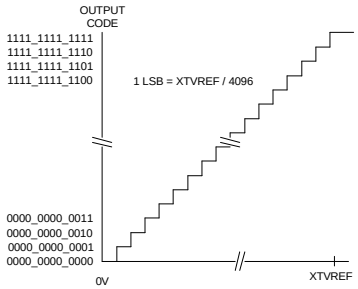
When an external reference source is applied to the XTVREF pin, it is mandatory not to power-on the PGA. The internal bandgap reference should also remain deactivated.

A/D Converter

The VMX51C1016 includes a feature rich and highly configurable on-chip 12-bit A/D converter.

The A/D conversion data is output as an unsigned 12-bit binary with 1 LSB = Full Scale/4096. The following figure describes the ideal transfer function for the ADC.

FIGURE 39: IDEAL A/D CONVERTER TRANSFER FUNCTION



The A/D converter includes a system that provides the ability to trigger automatic periodic conversions of up to 10kHz without processor intervention.

Once the conversion is complete, the A/D system can activate an interrupt that can wake-up the processor (assuming it has been put into idle mode) or automatically throttle the processor clock to full speed.

The VMX51C1016 ADC can also be configured to perform conversion on one specific channel or on four consecutive channels (in round-robin fashion).

These features make the A/D converter adaptable for many applications.

The following paragraphs describe the A/D converter register features.

ADC Data Registers

The ADC data registers hold the ADC conversion results. The ADCDxLO register(s) hold the eight least significant bits (LSBs) of the conversion results, while the ADCDxHI register(s) hold the four most significant bits (MSB) of the conversion results.

TABLE 93: (ADCD0LO) ADC CHANNEL 0 DATA REGISTER, LOW BYTE - SFR A6H

Bit	Mnemonic	Function
7:0	ADCD0LO	ADC channel 0 low

TABLE 94: (ADCD0HI) ADC CHANNEL 0 DATA REGISTER, HIGH BYTE - SFR A7H

Bit	Mnemonic	Function
3:0	ADCD0HI	ADC channel 0 high

TABLE 95: (ADCD1LO) ADC CHANNEL 1 DATA REGISTER, LOW BYTE - SFR A9H

7	6	5	4	3	2	1	0
ADCD1LO [7:0]							

Bit	Mnemonic	Function
7:0	ADCD1LO	ADC channel 1 low

TABLE 96: (ADCD1HI) ADC CHANNEL 1 DATA REGISTER, HIGH BYTE - SFR AAH

7	6	5	4	3	2	1	0
-	-	-	-	ADCD1HI [3:0]			

Bit	Mnemonic	Function
3:0	ADCD1HI	ADC channel 1 high

TABLE 97: (ADC2LO) ADC CHANNEL 2 DATA REGISTER, LOW BYTE - SFR ABH

7	6	5	4	3	2	1	0
ADCD2LO [7:0]							

Bit	Mnemonic	Function
7:0	ADCD2LO	ADC channel 2 low

TABLE 98: (ADCD2HI) ADC CHANNEL 2 DATA REGISTER, HIGH BYTE - SFR ACH

7	6	5	4	3	2	1	0
-	-	-	-	ADCD2HI [3:0]			

Bit	Mnemonic	Function
7:4	-	-
3:0	ADCD2HI	ADC channel 2 high

TABLE 99: (ADCD3LO) ADC CHANNEL 3 DATA REGISTER, LOW BYTE - SFR ADH

7	6	5	4	3	2	1	0
ADCD3LO [7:0]							

Bit	Mnemonic	Function
7:0	ADCD3LO	ADC channel 3 low

TABLE 100: (ADCD3HI) ADC CHANNEL 3 DATA REGISTER, HIGH BYTE - SFR AEH

7	6	5	4	3	2	1	0
-	-	-	-	ADCD3HI [3:0]			

Bit	Mnemonic	Function
7:4	-	-
3:0	ADCD3HI	ADC channel 3 high

ADC Input Selection

A/D conversions can be performed on a single channel, sequentially on the four lower channels or sequentially on the four upper channels of the ADC input multiplexer.

An input buffer is present on each of the four external ADC inputs (ADIN0 to AIN3).

These buffers must be enabled before a conversion can take place on the ADC AIN0-AIN3 inputs. These buffers are enabled by setting the corresponding bits of the lower quartet (AIEN [3:0]) of the INMUXCTRL register to 1.

TABLE 101: (INMUXCTRL) ANALOG INPUT MULTIPLEXER CONTROL REGISTER - SFR B5H

7	6	5	4	3	2	1	0
-	ADCINSEL [2:0]			AINEN [3:0]			

Bit	Mnemonic	Function
7	-	-
6:4	ADCINSEL[2:0]	ADC Input Select 000 - AIN0 001 - AIN1 010 - AIN2 011 - AIN3 111 - ADCTA
3:0	AINEN[3:0]	Analog Input Enable

The upper four bits of the INMUXCTRL register are used to define the channel on which the conversion will take place when the ADC is set to perform the conversion on one specific channel.

ADC Control Register

The ADCCTRL register is the main register used for control and operation of the ADC operating mode.

TABLE 102: (ADCCTRL) ADC CONTROL REGISTER - SFR A2H

7	6	5	4
ADCIRQCLR	XVREFCAP	1	ADCIRQ

3	2	1	0
ADCIE	ONECHAN	CONT	ONESHOT

Bit	Mnemonic	Function
7	ADCIRQCLR	ADC interrupt clear Writing 1 Clears interrupt
6	XVREFCAP	Always keep this bit at 1
5	Reserved = 1	Keep this bit = 1
4	ADCIRQ	Read ADC Interrupt Flag Write 1 generate ADC IRQ
3	ADCIE	ADC interrupt enable
2	ONECHAN	1 = Conversion is performed on one channel Specified ADCINSEL 0 = Conversion is performed on 4 ADC channels
1	CONT	1 = Enable ADC continuous conversion
0	ONESHOT	1 = Force a single conversion on 1 or 4 channels

ADC Continuous / One Shot Conversion

The CONT bit sets the ADC conversion mode. When the CONT bit is set to 1, the ADC will implement continuous conversions at a rate defined by the conversion rate register.

When the CONT bit is set to 0, the A/D operates in “one shot” mode, initiating a conversion when the ONESHOT bit of the ADCCONTRL register is set.

ADC One Channel/Four Channel Conversion

The VMX51C1016’s ADC includes a feature that renders it possible to perform a conversion on one specific channel or on four consecutive channels.

This feature minimizes the load on the processor when reading more than one ADC input is required.

The ONECHAN bit of the ADCCTRL register controls this feature. When the ONECHAN is set to 1, the conversion will take place on the channel selected by the INMUXCTRL register. Once the conversion is complete, the result will be placed into the ADCD0LO and ADCD0HI registers

When the ONECHAN bit is set to 0, the conversion, once triggered, will be done sequentially on four channels and the conversion results will be placed into the ADCDxLO and ADCDxHI registers.

Bit 6 of the INMUXCTRL register controls whether the conversion will take place on the four upper channels of the input multiplexer or on the four lower channels.

ADC Clock Source Configuration

The A/D converter derives its clock source from the main VMX51C1016 clock. .The frequency of the ADC clock should be set between 250kHz and 1.25MHz.

Configuration of the ADC clock source frequency is done by adjusting the value of the ADCCLKDIV register. The following equation is used to calculate the ADC reference clock.

ADC Clock Reference Equation:

$$\text{ADC Clk ref} = \frac{f_{OSC}}{4x\ (\text{ADCCDIV} + 1)}$$

The ADC conversion requires 111 ADC clock cycles to perform the conversion on one channel.

The following table provides recommended ADCCLKDIV register values versus conversion rate. The numbers given are conservative figures and derived from a 14.74MHz clock.

ADCCLKDIV	Maximum Conv. Rate*
0x02	10500 Hz
0x03	8000 Hz
0x05	5000 Hz
0x07	4000 Hz
0x08	3500 Hz
0x09	3200 Hz
0x0B	2500 Hz
0x0D, 0x0E, 0x0F	2200 Hz

* The maximum conversion rate is for the single channel condition. If the conversion is performed on four channels, divide the maximum conversion rate by 4. For example, performing the conversion at 25KHz on four channels, the ADCCLKDIV register should be set to 0x02 (4 x 2500Hz = 10KHz).

TABLE 103: (ADCCLKDIV) ADC CLOCK DIVISION CONTROL REGISTER - SFR 95H

7	6	5	4	3	2	1	0
ADCCLKDIV [7:0]							

Bit	Mnemonic	Function
7:0	ADCCLKDIV[7:0]	ADC clock divider

ADC Conversion Rate Configuration

The VMX51C1016's ADC conversion rate, when configured in continuous mode, is defined by the value present in a 24-bit A/D conversion rate register that serves as the time base for triggering the ADC conversion process.

The following equation is used to calculate the value of the conversion rate.

Conversion Rate Equation:

$$\text{Conversion rate registers value (24-bit)} = \frac{f_{OSC}}{\text{Conv\_Rate}}$$

The conversion rate register is accessible using three SFR registers as follows:

TABLE 104: (ADCCONVRLOW) ADC CONVERSION RATE REGISTER LOW BYTE - SFR A3H

Bit	Mnemonic	Function
7:0	ADCCONVRLOW	Conversion rate low byte

TABLE 105: (ADCCONVERMED) ADC CONVERSION RATE REGISTER MED BYTE - SFR A4H

Bit	Mnemonic	Function
7:0	ADCCONVRMED	Conversion rate medium byte

TABLE 106: (ADCCONVRHIGH) ADC CONVERSION RATE REGISTER HIGH BYTE - SFR A5H

Bit	Mnemonic	Function
7:0	ADCCONVRHIGH	Conversion rate high byte

The following table provides examples of typical values versus conversion rate.

Conversion Rate	**Fosc= 14.74MHz**
1Hz	E10000h
10Hz	168000h
100Hz	024000h
1kHz	003999h
2.5kHz	00170Ah
5kHz	000B85h
8kHz	000733h
10kHz	0005C2h

ADC Status Register

The ADC shares interrupt vector 0x6B with the interrupt on Port 1 change and Compare and Capture Unit 3. To enable the ADC interrupt, the ADCIE bit of the ADCCTRL register must be set. Before or at the same time this bit is set, the ADCIRQCLR and ADCIRQ bits must be cleared. The ADCPCIE bit of the IEN1 register must also be set, as well as the EA bit of the IEN0 register.

Once the ADC interrupt occurs, the ADC interrupt must be cleared by writing a '1' into the ADCINTCLR bit of the ADCCTRL register. The ADCIF flag in the IRCON register must also be cleared.

A/D Converter Use Example

The following provides example code for the A/D converter. The first section of the code is the interrupt setup/module configuration, whereas the second section is the interrupt function itself.

Sample C code to setup the A/D converter:

```
//-------------------------------------------------------------------------//
//                          MAIN FUNCTION
//-------------------------------------------------------------------------//

(…)
at 0x0100 void main (void) {

//*** Initialize the Analog Peripherals ***

ANALOGPWREN = 0x07;             //Enable the following analog
                                //peripherals:  ADC, PGA,
                                // BGAP.  TA = OFF (mandatory)

//Configure the ADC and Start it
ADCCLKDIV=0x0F;                 //SET ADC CLOCK SOURCE
ADCCONVRLOW=0x00;               //CONFIGURE CONVERSION RATE
ADCCONVRMED=0x40;               //= 100Hz @ 14.74 MHz
ADCCONVRHIGH =0x02;

INMUXCTRL=0x0F;                 //Enable All ADC External inputs
                                //buffers and select ADCI0
ADCCTRL=0xEA;                   //Configure the ADC as follow:
                                //bit 7: =1 ADCIRQ Clear
                                //Bit 6: =1 XVREFCAP (always)
                                //Bit 5: =1 (always)
                                //Bit 4: =0 = ADCIRQ (don't care)
                                //Bit 3: =1 = ADC IRQ enable
                                //Bit 2: =0 conversion on 4
                                //channels
                                //Bit 1: =1 Continuous conversion
                                //Bit 0: =0 No single shot mode

//*** Configure the interrupts
IEN0 |= 0x80;                   //enable main interrupt
IEN1 |= 0x020;                  //Enable ADC Interrupt
while(1);                       //Infinite loop waiting ADC interrupts
}// End of main()...

//-------------------------------------------------------------------------//
//                      ADC INTERRUPT ROUTINE
//-------------------------------------------------------------------------//
void int_adc (void) interrupt 13 {
idata int value = 0;

IEN0 &= 0x7F;                   //disable ext0 interrupts
ADCCTRL |=0x80;                 //Clear ADC interrupt

// Read ADC channel 0
value = ADCD0HI;
value = valeur*256;
value = valeur + ADCD0LO;
(…)
// Read ADC channel 3
value = ADCD3HI;
value = valeur*256;
value = valeur + ADCD3LO;
(…)
IRCON &= 0xDF;                  //Clear adc irq flag
ADCCTRL |=0xFA;                 //prepare adc for next acquisition
IEN0 |= 0x80;                   // enable all interrupts

}// End of ADC IRQ

(…)
```

Warning:

When using the ADC, make sure the output multiplexer controlled by the TAEN bit of the ANALOGPWREN register (92h) is powered-down at all times, otherwise, the signal present on the ISRCOUT can be routed back to the selected ADC input, causing conversion errors.

TABLE 107: (PGACAL0) PGA CALIBRATION BIT 0 VALUE - SFR BCH

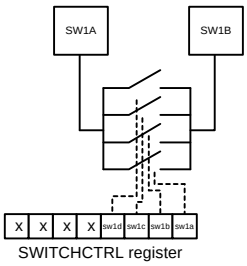
7	6	5	4	3	2	1	0
PGACAL0	Reserved						

Bit	Mnemonic	Function
7	PGACAL0	Bit 0 of PGACAL
6:0	--	Reserved

Digitally Controlled Switches

On the VMX51C1016 includes a digital switch composed of four sub-switches connected in parallel. These sub-switches can be individually controlled by writing to the SFR register at B7h.

FIGURE 42: SWITCH FUNCTIONAL DIAGRAM



SWITCHCTRL register

The “ON” switch resistance is between 50 and 100 Ohms, depending on the number of sub-switches being used. If, for example, one sub-switch is closed, the switch resistance will be about 100 Ohms, and if all four switches are closed, the switch resistance will go down to about 50 Ohms.

TABLE 108: (SWITCHCTRL) USER SWITCHES CONTROL REGISTERS - SFR B7H

7	6	5	4	3	2	1	0
Not Used but implemented				SWTCH1 [3:0]			

Bit	Mnemonic	Function
7:4	User Flags	Not used but implemented bits Can be used as general purpose storage
3:0	SWITCH1[3:0]	Switch 1 control (composed of 4 individual switches each bit controlled)

The upper 4 bits of the SWITCHCTRL register are not used but they are implemented. They can be used as general purpose flags.

Analog Output Multiplexer

The VMX51C1020’s analog output multiplexer is used for production test purposes and provides access to internal test points of the analog signal path. It can however, be used in applications, but due to its high intrinsic impedance, care must be taken with respect to loading.

We recommend not accessing this SFR register.

TABLE 109: (OUTMUXCTRL) ANALOG OUTPUT MULTIPLEXER CONTROL REGISTER - SFR B6H

7	6	5	4	3	2	1	0
-	-	-	-	-	TAOUTSEL [2:0]		

Bit	Mnemonic	Function
7:3	Unused	Unused
2:0	TAOUTSEL[2:0]	Signal output on TA 000 – AIN0 001 – AIN1 010 – AIN2 011 – AIN3 100 – VBGAP 101 – reserved 110 – unused 111 – unused

VMX51C1016 Interrupts

The VMX51C1016 is a highly integrated device incorporating a vast number of peripherals for which a comprehensive set of 24 interrupt sources sharing 11 interrupt vectors is available to ease system program development. Most of the VMX51C1016 peripherals can generate an interrupt, providing feedback to the MCU core that an event has occurred or a task has been completed.

The following are key VMX51C1016 interrupt features:

- Each digital peripheral on the VMX51C1016 has an interrupt channel
- The SPI, UARTs and I²C all have event specific flag bits
- When the processor is in IDLE mode, an interrupt may be used to wake it up
- The processor can run at full speed during interrupt routines

The following table summarizes the interrupt sources, natural priority and associated interrupt vector addresses on the VMX51C1016.

TABLE 110: INTERRUPT SOURCES AND NATURAL PRIORITY

Interrupt	Interrupt Vector
Reserved	0E43h
INT0	0003h
UART1	0083h
TIMER 0	000Bh
SPI TX Empty	004Bh
Reserved	0013h
SPI RX & SPI RX OVERRUN / COMPINT0	0053h
TIMER 1	001Bh
I2C (Tx, Rx, Rx Overrun) / COMPINT1	005Bh
UART0	0023h
MULT/ACCU 32bit Overflow / COMPINT2	0063h
TIMER 2: T2 Overflow, T2EX	002Bh
ADC and interrupt on Port 1 change (8 int.) / COMPINT3	006Bh

It is also possible to program the interrupts to wake-up the processor from an IDLE condition or force its clock to throttle up to full speed when an interrupt condition occurs.

Interrupt Enable Registers

The following tables describe the interrupt enable registers and their associated bit functions:

TABLE 111: (IEN0) INTERRUPT ENABLE REGISTER 0 - SFR A8H

7	6	5	4
EA	WDT	T2IE	S0IE

3	2	1	0
T1IE	0	T0IE	INT0IE

Bit	Mnemonic	Function
7	EA	General Interrupt control 0 = Disable all Enabled interrupts 1 = Authorize all Enabled interrupts
6	WDT	Watch Dog timer refresh flag. This bit is used to initiate a refresh of the watchdog timer. In order to prevent an unintentional reset, the watchdog timer the user must set this bit directly before SWDT.
5	T2IE	Timer 2 Overflow / external Reload interrupt 0 = Disable 1 = Enable
4	S0IE	Uart0 interrupt. 0 = Disable 1 = Enable
3	T1IE	Timer 1 overflow interrupt 0 = Disable 1 = Enable
2	Reserved	Always keep this bit to 0
1	T0IE	Timer 0 overflow interrupt 0 = Disable 1 = Enable
0	INT0IE	External Interrupt 0 0 = Disable 1 = Enable

TABLE 112: (IEN1) INTERRUPT ENABLE 1 REGISTER -SFR E8H

7	6	5	4
T2EXIE	SWDT	ADCPCIE	MACOVIE

3	2	1	0
I2CIE	SPIRXOVIE	SPITEIE	reserved

Bit	Mnemonic	Function
7	T2EXIE	T2EX interrupt Enable 0 = Disable 1 = Enable
6	SWDT	Watch Dog timer start/refresh flag. Set to activate/refresh the watchdog timer. When directly set after setting WDT, a watchdog timer refresh is performed. Bit SWDT is reset.
5	ADCPCIE	ADC and Port change interrupt 0 = Disable 1 = Enable
4	MACOVIE	MULT/ACCU Overflow 32 bits interrupt 0 = Disable 1 = Enable
3	I2CIE	I2C Interrupt 0 = Disable 1 = Enable
2	SPIRXOVIE	SPI Rx avail + Overrun 0 = Disable 1 = Enable
1	SPITEIE	SPI Tx Empty interrupt 0 = Disable 1 = Enable
0	reserved	

TABLE 113: (IEN2) INTERRUPT ENABLE 2 REGISTER - SFR 9AH

7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	S1IE

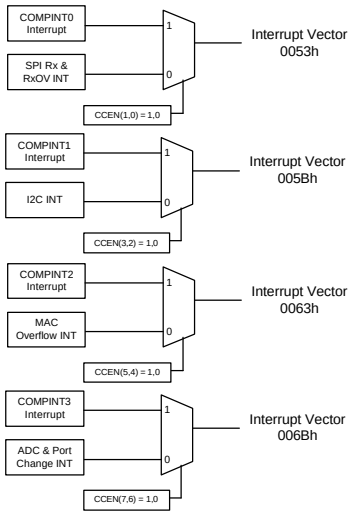
Bit	Mnemonic	Function
7-1	-	-
0	S1IE	UART 1 Interrupt 0 = Disable UART 1 Interrupt 1 = Enable UART 1 Interrupt

Timer 2 Compare Mode Impact on Interrupts

The SPI RX (and RXOV), I²C, MULT/ACCU and ADC interrupts are shared with the four Timer 2 compare and capture unit interrupts.

When the compare and capture units of Timer 2 are configured in compare mode via the CCEN register, the compare and capture unit takes control of one interrupt vector as shown in the next figure.

FIGURE 43: COMPARE CAPTURE INTERRUPT STRUCUTRE



The impact of this is that the corresponding peripheral interrupt, if enabled, will be blocked. The output signal from the comparison module will be routed to the interrupt system and the control lines will be dedicated to the compare and capture unit.

This interrupt control “take over” is specific to each individual compare and capture unit. For example if Compare and Capture Unit 2 is configured to generate a PWM signal on P1.2, the MULT/ACCU overflow interrupt, if enabled, will be dedicated to Compare and Capture Unit 2 and the SPI, I²C and ADC interrupts won’t be affected.

Interrupt Status Flags

The IRCON register is used to identify the source of an interrupt. Before exiting the interrupt service routine, the IRCON register bit that corresponds with the serviced interrupt should be cleared.

TABLE 114: (IRCON) INTERRUPT REQUEST CONTROL REGISTER - SFR 91H

7	6	5	4
T2EXIF	TF2IF	ADCIF	MACIF

3	2	1	0
I2CIF	SPIRXIF	SPITXIF	Reserved

Bit	Mnemonic	Function
7	T2EXIF	Timer 2 external reload flag This bit informs the user whether an interrupt has been generated from T2EX, if the T2EXIE is enabled.
6	TF2IF	Timer 2 overflow flag
5	ADCIF / COMPINT3	A/D converter interrupt request flag/ port 0 change. / COMPINT3
4	MACIF / COMPINT2	MULT/ACCU unit interrupt request flag / COMPINT2
3	I2CIF / COMPINT1	I^2C interrupt request flag / COMPINT1
2	SPIRXIF / COMPINT0	RX available flag SPI + RX Overrun / / COMPINT0
1	SPITXIF	TX empty flag SPI
0	Reserved	Reserved

Interrupt Priority Register

All of the VMX51C1016's interrupt sources are combined into groups with four levels of priority.

These groups can be programmed individually to one of the four priority levels: from Level 0 to Level 3, with Level 3 being the highest priority.

The IP0 and IP1 registers serve to define the specific priority of each of the interrupt groups. By default, when the IP0 and IP1 registers are at reset state 00h, the natural priority order of the interrupts described above is in force.

TABLE 115: (IP0) INTERRUPT PRIORITY REGISTER 0 - SFR B8H

7	6	5	4	3	2	1	0
UF8	WDTSTAT	IP0 [5:0]					

Bit	Mnemonic	Function		
7	UF8	User Flag bit		
6	WDTSTAT	Watch Dog Timer status flag. Set to 1 by hardware when the Watch Dog Timer overflows. Must be cleared manually		
5	IP0.5	Timer 2	Port1 Change	ADC
4	IP0.4	UART0	-	MULT/ACCU
3	IP0.3	Timer 1	-	I2C
2	IP0.2	-	-	SPI RX available
1	IP0.1	Timer 0 Interrupt	-	SPI TX Empty
0	IP0.0	External INT0	UART1	External INT 0

Table 116: (IP1) Interrupt Priority Register 1 - SFR B9h

7	6	5	4	3	2	1	0
-	-	IP1 [5:0]					

Bit	Mnemonic	Function		
7	-	-		
6	-	-		
5	IP1.5	Timer 2	Port1 Change	ADC
4	IP1.4	UART0	-	MULT/ACCU
3	IP1.3	Timer 1	-	I2C
2	IP1.2	-	-	SPI RX available
1	IP1.1	Timer 0 Interrupt	-	SPI TX Empty
0	IP1.0	External INT0	UART1	External INT 0

Configuring the IP0 and IP1 registers makes it possible to change the priority order of the peripheral interrupts in order to give higher priority to a specified interrupt that belongs to a specified group.

TABLE 117: INTERRUPT GROUPS

Bit	**Interrupt Group**		
IP1.5, IP0.5	Timer 2	Port1 Change	ADC
IP1.4, IP0.4	UART0	-	MULT/ACCU
IP1.3, IP0.3	Timer 1	-	I2C
IP1.2, IP0.2	-	-	SPI RX available
IP1.1, IP0.1	Timer 0 Interrupt	-	SPI TX Empty
IP1.0, IP0.0	External INT0	UART1	External INT 0

The respective values of the IP1.x and IP0.x bits define the priority level of the interrupt group vs. the other interrupt groups as follows:

TABLE 118: INTERRUPT PRIORITY LEVEL

IP1.x	IP0.x	Priority Level
0	0	Level 0 (Low)
0	1	Level 1
1	0	Level 2
1	1	Level 3 (High)

The WDTSTAT bit of the IP0 register is the watchdog status flag, which is set to 1 by the hardware whenever a watchdog timer overflow occurs. This bit must be cleared manually.

Finally, bit 7 of the IP0 register can be used as a general purpose user flag.

Setting up INT0 Interrupts

The IT0 bit of the TCON register defines whether external interrupt 0 will be edge or level triggered.

When an interrupt condition occurs on INT0, the associated interrupt flag IE0 will be set. The interrupt flag is automatically cleared when the interrupt is serviced.

TABLE 119: (TCON) TIMER 0, TIMER 1 TIMER/COUNTER CONTROL - SFR 88H

7	6	5	4
TF1	TR1	TF0	TR0

3	2	1	0
--	--	IE0	IT0

Bit	Mnemonic	Function
7	TF1	Timer 1 overflow flag set by hardware when Timer 1 overflows. This flag can be cleared by software and is automatically cleared when interrupt is processed.
6	TR1	Timer 1 Run control bit. If cleared Timer 1 stops.
5	TF0	Timer 0 overflows flag set by hardware when Timer 0 overflows. This flag can be cleared by software and is automatically cleared when interrupt is processed.
4	TR0	Timer 0 Run control bit. If cleared timer 0 stops.
3	--	Reserved
2	--	Reserved
1	IE0	Interrupt 0 edge flag. Set by hardware when falling edge on external pin INT0 is observed. Cleared when interrupt is processed.
0	IT0	Interrupt 0 type control bit. Selects falling edge or low level on input pin to cause interrupt.

INT0 example

The following provides example code for interrupt setup and module configuration:

```
//-------------------------------------------------------------------------
// Sample C code to setup INT0
//-------------------------------------------------------------------------
#pragma TINY
#include <vmixreg.h>

at 0x0100 void main (void) {

// INT0 Config

        TCON |= 0x01;   //Interrupt on INT0 will be caused by a High->Low
                        //edge on the pin
// Enable INT0 interrupts
        IEN0 |= 0x80;           // Enable all interrupts
        IEN0 |= 0x01;           // Enable interrupt INT0

// Wait for INT0…
        do
        {
        }while(1);      //Wait for INT0 interrupts

}//end of main function

//-------------------------------------------------------------------------
// Interrupt Function

void int_ext_0 (void) interrupt 0
{
IEN0 &= 0x7F;                   // Disable all interrupts

/* Put the Interrupt code here*/

IEN0 |= 0x80;                   // Enable all interrupts
}
//-------------------------------------------------------------------------
```

UART0 and UART1 Interrupt Example

The following program example demonstrates how to initialize the UART0 and UART1 interrupts.

```
//-------------------------------------------------------------------------
// Sample C code for UART0 and UART1 interrupt example
//-------------------------------------------------------------------------
#pragma TINY
#include <vmixreg.h>

// --- function prototypes
void txmit0( unsigned char charact);
void txmit1( unsigned char charact);
void uart1Config(void);
void uart0ws0relcfg(void);

// - Constants definition
sbit UART_TX_EMPTY = USERFLAGS^1;

//-------------------------------------------------------------------------
//                        MAIN FUNCTION
//-------------------------------------------------------------------------
 at 0x0100 void main (void) {

        // Enable and configure the UART0  & UART1
        uart0ws0relcfg();       //Configure UART0
        uart1Config();          //Configure UART1


//*** Configure the interrupts
        IEN0 |= 0x91;           //Enable UART0 Int + enable all int
        IEN2 |= 0x01;           //Enable UART1 Interrupt
        do
                {
                }while(1);      //Wait for UARTs interrupts
        // End of main()...

//-------------------------------------------------------------------------
//                INTERRUPT ROUTINES
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// UART0 interrupt
//
// Retrieve character received in S0BUF and transmit it
// back on UART0
// //-------------------------------------------------------------------------
void int_uart0 (void) interrupt 4 {

        IEN0 &= 0x7F;                   //disable All interrupts

//--- The only UART0 interrupt source is Rx...
        txmit0(S0BUF);                  // Return the character
                                        //received on UART0

        S0CON = S0CON & 0xFC;           //clear R0I & T0I bits
        IEN0 |= 0x80;                   // enable all interrupts
        }// end of UART0 interrupt

//-------------------------------------------------------------------------
// UART1 interrupt
//
// Retrieve character received in S1BUF and transmit it
// back on UART1
// //-------------------------------------------------------------------------
void int_uart1 (void) interrupt 16 {

        IEN0 &= 0x7F;                   //disable All interrupts

//--- The only UART1 interrupt source is Rx...
        txmit1(S1BUF);                  // Return the character
                                        // received on UART1
        S1CON = S1CON & 0xFC;           // clear both R1I & T1I bits
        IEN0 |= 0x80;                   // enable all interrupts

}// end of UART1 interrupt
```

Note: See UART0 / UART1 section for configuration examples and TXMITx functions

Interrupt on P1 change

The VMX51C1016 includes an interrupt on the port change feature, which is available on the Port 1 pins of the VMX51C1016.

This feature is like having eight extra external interrupt inputs sharing the ADC interrupt vector at address 006Bhc and can be very useful for applications such as switches, keypads, etc.

To activate this interrupt, the bits corresponding to the pins being monitored must be set in the PORTIRQEN register. The ADCPCIE bit in the IEN1 register must be set as well as the EA bit of the IEN0 register.

TABLE 120: (PORTIRQEN) PORT CHANGE IRQ CONFIGURATION - SFR 9FH

7	6	5	4
--	--	--	--

3	2	1	0
P13IEN	P12IEN	P11IEN	P10IEN

Bit	Mnemonic	Function
7	--	Reserved, Keep at 0
6	--	Reserved, Keep at 0
5	--	Reserved, Keep at 0
4	--	Reserved, Keep at 0
3	P13IEN	Port 1.3 IRQ on change enable 0 = Disable 1 = Enable
2	P12IEN	Port 1.2 IRQ on change enable 0 = Disable 1 = Enable
1	P11IEN	Port 1.1 IRQ on change enable 0 = Disable 1 = Enable
0	P10IEN	Port 1.0 IRQ on change enable 0 = Disable 1 = Enable

The PORTIRQSTAT register monitors the occurrence of the interrupt on port change. This register serves to define which P1 pin has changed when an interrupt occurs.

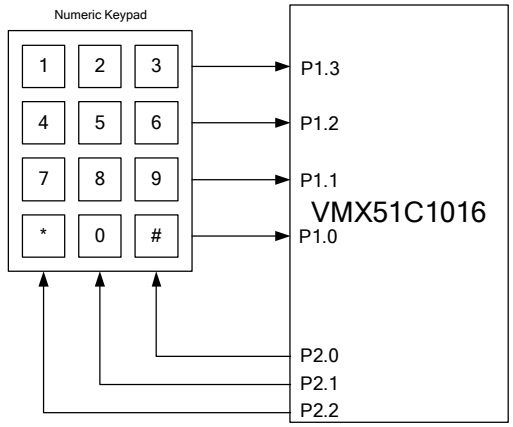
TABLE 121: (PORTIRQSTAT) PORT CHANGE IRQ STATUS - SFR A1H

7	6	5	4
P17ISTAT	P16ISTAT	P15ISTAT	P14ISTAT

3	2	1	0
P13ISTAT	P12ISTAT	P11ISTAT	P10ISTAT

Bit	Mnemonic	Function
7	--	Unused
6	--	Unused
5	--	Unused
4	--	Unused
3	P13ISTAT	Port 1.3 changed 0 = No 1 = Yes
2	P12ISTAT	Port 1.2 changed 0 = No 1 = Yes
1	P11ISTAT	Port 1.1 changed 0 = No 1 = Yes
0	P10ISTAT	Port 1.0 changed 0 = No 1 = Yes

FIGURE 44: APPLICATION EXAMPLE OF PORT CHANGE INTERRUPT



The following provides an assembler example for configuration of the interrupt on Port 1 pin change and how it is shared with the ADC interrupt.

```
include VMIXreg.INC
;*** INTERRUPT VECTORS JUMP TABLE  *
ORG 0000H               ;BOOT ORIGIN VECTOR
            LJMP        START
ORG 006BH               ;INT ADC and P1 change interrupt
            LJMP        INT_ADC_P1

;*** MAIN PROGRAM
ORG    0100h

START:      MOV         DIGPWREN,#01H       ;ENABLE TIMER 2
            MOV         P2PINCFG,#0FFH

;*** Initialise Port change interrupt on P1.0 - P1.7
            MOV         PORTIRQSTAT,#00H
            MOV         PORTIRQEN,#11111111B

;*** Initialise the ADC, BGAP, PGA Operation
            MOV         ANALOGPWREN,#07h

;Select CH0 as ADC input + Enable input buffer + Adc clk
            MOV         INMUXCTRL,#0Fh
            MOV         ADCCLKDIV,#0Fh
            MOV         ADCCONVRLOW,#000h

;*** configure ADC Conversion Rate
            MOV         ADCCONVRMED,#080h
            MOV         ADCCONVRHIGH,#016h
            MOV         ADCCTRL,#11111010b

;***Activate All interrupts + (serial port for debugger support)
            MOV         IEN0,#090H
;*** Enable ADC interrupt
            MOV         IEN1,#020H

;***Wait IRQ…
WAITIRQ:    LJMP        WAITIRQ

ORG    0200h
;*******************************************************************************
;* IRQ ROUTINE:        IRQADC + P1Change
;*******************************************************************************
INT_ADC_P1:
            ;MOV         IEN0,#00h    ;DISABLE ALL INTERRUPT

;***Check if IRQ was caused by Port Change
;***If PORTIRQSTAT = 00h -> IRQ comes from ADC
            MOV         A,PORTIRQSTAT
            JZ          CASE_ADC

;*** If interrupt was caused by Port 1, change
CASE_P0CHG:
            MOV         PORTIRQSTAT,#00H
;*** Perform other instructions related to Port1 change IRQ
            ;(...)

;*** Jump to Interrupt end
            AJMP        ENDADCP1INT

;*** If interrupt was caused by ADC
CASE_ADC:
            ANL         ADCCTRL,#11110011b
;***Reset ADC interrupt flags & Reset ADC for next acquisition
            ORL         ADCCTRL,#080h
            ORL         ADCCTRL,#11111010b
;*** Perform other instructions related to Port1 change IRQ
;(...)

;** End of ADC and Port 1 Change interrupt
ENDADCP0INT:
            ANL         IRCON,#11011111b

;***Enable All interrupts before exiting
            ; MOV       IEN0,#080H
            RETI
END
```

The Clock Control Circuit

The VMX51C1016's clock control circuit allows dynamic adjustment of the clock from which the processor and the peripherals derive their clock source. This enables reduction of overall power consumption by modulating the operating frequency according to processing requirements or peripheral use.

A typical application for this is a portable acquisition system, in which significant power savings can be achieved by lowering the operating frequency between A/D conversions, and automatically throttling it to full speed when an A/D converter interrupt is generated. Note that the ADC operation is not affected by the clock control unit.

The clock control circuit allows adjusting the system clock from [Fosc/1] (full speed) down to [Fosc/512]. The clock division control is done via the CLKDIVCTRL register located at address 94h of the SFR register area.

TABLE 122: (CLKDIVCTRL) CLOCK DIVISION CONTROL REGISTER -SFR 94H

7	6	5	4
SOFTRST	-	-	IRQNORMSPD

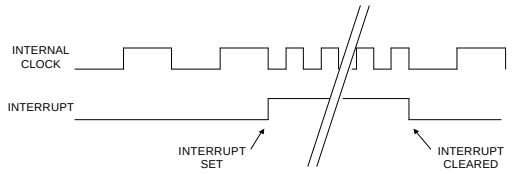
3	2	1	0
MCKDIV [3:0]			

Bit	Mnemonic	Function
7	SOFTRST	Writing 1 into this bit location provokes a reset. Read as a 0
6:5	-	-
4	IRQNORMSPD	0 = Full Speed in IRQ 1 = Selected speed during IRQs
3:0	MCKDIV [3:0]	Master Clock Divisor 0000 – Sys CLK 0001 = SYS /2 0010 = SYS /4 0011 = SYS /8 0100 = SYS /16 0101 = SYS /32 0110 = SYS /64 0111 = SYS /128 1000 = SYS /256 1001 = SYS /512 (…) 1111 = SYS /512

The value written into the lower nibble of the CLKDIVCTRL register, MCKDIV [3:0], defines the clock division ratio.

When the IRQNORMSPD bit is cleared, the VMX51C1016 will run at the maximum operating speed when an interrupt occurs (see the following figure).

FIGURE 45: CLOCK TIMING WHEN AN INTERRUPT OCCURS



Once the interrupt is cleared, the VMX51C1016 returns to the selected operating speed as defined by the MCKDIV [3:0] bits of the CLKDIVCTRL register.

When the IRQNORMSPD bit is set the VMX51C1016 will continue to operate at the selected speed as defined by the MCKDIV [3:0] bits of the CLKDIVCTRL register.

Note With the exception of the A/D converter, all the peripheral operating speeds are affected by the clock control circuit.

Software Reset

Software reset can be generated by setting the SOFTRST bit of the CLKDIVCTRL register to 1.

Power-on/Brown-Out Reset

The VMX51C1016 includes a power-on-reset/brown-out detector circuit that ensures the VMX51C1016 enters and stays in the reset state as long as the supply voltage is below the reset threshold voltage (in the order of 3.7 – 4.0 volts).

In most applications, the VMX51C1016 requires no external components to perform a power-on reset when the device is powered-on.

The VMX51C1016 also has a reset pin for applications in which external reset control is required. The reset pin includes an internal pull-up resistor. When a power-on reset occurs, all SFR locations return to their default values and peripherals are disabled.

Errata:

The VMX51C1016 may fail to exit the reset state if the supply voltage drops below the reset threshold, but not below 3 volts. For applications where this condition can occur, use an external supply monitoring circuit to reset the device.

Processor Power Control

The processor power management unit has two modes of operation: IDLE mode and STOP mode.

IDLE Mode

When the VMX51C1016 is in IDLE mode, the processor clock is halted. However, the internal clock and peripherals continue to run. The power consumption drops because the CPU is not active. As soon as an interrupt or reset occurs, the CPU exits IDLE mode.

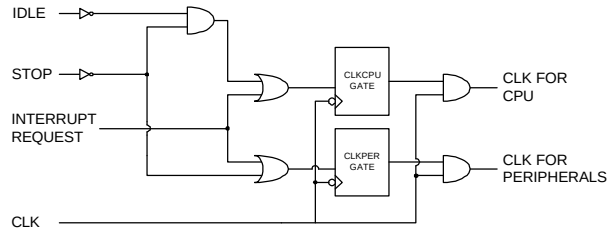
In order to enter IDLE mode, the user must set the IDLE bit of the PCON register. Any enabled interrupts will force the processor to exit IDLE mode.

STOP Mode

In order to enter STOP mode, the user must set the STOP bit of the PCON register. In this mode, in contrast to IDLE mode, all internal clocking shuts down. The CPU will exit this state only when a non-clocked external interrupt or reset occurs (internal interrupts are not possible because they require clocking activity).

The following interrupts can restart the processor from STOP mode: Reset, INT0, SPI Rx/Rx Overrun, and the I²C interface.

FIGURE 46: POWER MANAGEMENT ON THE VMX51C1016



The following table describes the power control register of the VMX51C1016.

TABLE 123: (PCON) POWER CONTROL (CPU) - SFR 87H

7	6	5	4	3	2	1	0
SMOD	-	-	-	GF1	GF0	STOP	IDLE

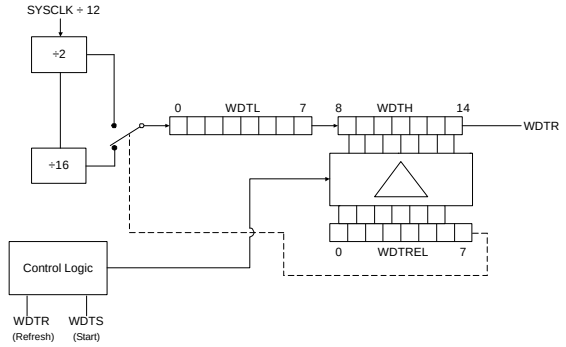
Bit	Mnemonic	Function
7	SMOD	The speed in Mode 2 of Serial Port 0 is controlled by this bit. When SMOD= 1, f_{clk} /32. This bit is also significant in Mode 1 and 3, as it adds a factor of 2 to the baud rate.
6	-	-
5	-	-
4	-	-
3	GF1	Not used for power management
2	GF0	Not used for power management
1	STOP	Stop mode control bit. Setting this bit turns on the STOP Mode. STOP bit is always read as 0.
0	IDLE	IDLE mode control bit. Setting this bit turns on the IDLE mode. IDLE bit is always read as 0.

Watchdog Timer

The VMX51C1016's watchdog timer is used to monitor program operation and reset the processor in cases where the code could not refresh the watchdog before it's timeout period has lapsed. This can come about from an event that results in the program counter executing faulty or incorrect code and inhibiting the device from doing its intended job.

The watchdog timer consists of a 15-bit counter composed of two registers (WDTL and WDTH) and a reload register (WDTREL). See the following figure.

FIGURE 47: WATCH DOG TIMER



The WDTL and WDTH registers are not accessible from the SFR register. However, the WDTREL register makes it possible to load the upper 6 bits of the WDTH register.

The PRES bit of the WDTREL register selects the prescaler clock that is fed into the watchdog timer.

When PRES = 0, the clock prescaler = 24
When PRES = 1, the clock prescaler = 384

TABLE 124: (WDTREL) WATCHDOG TIMER RELOAD REGISTER - SFR D9H

7	6	5	4	3	2	1	0
PRES	WDTREL [6:0]						

Bit	Mnemonic	Function
7	PRES	Prescaler select bit. When set, the watchdog is clocked through an additional divide-by-16 prescaler.
6-0	WDTREL	7-bit reload value for the high-byte of the watchdog timer. This value is loaded into the WDT when a refresh is triggered by a consecutive setting of the WDT and SWDT bits.

TABLE 125: (IP0) INTERRUPT PRIORITY REGISTER 0 - SFR B8H

7	6	5	4	3	2	1	0
UF8	WDTSTAT	IP0 [5:0]					

Bit	Mnemonic	Function		
7	UF8	User Flag bit		
6	WDTSTAT	Watchdog timer status flag. Set to 1 by hardware when the Watch Dog timer overflows. Must be cleared manually		
5	IP0.5	Timer 2	Port1 Change	ADC
4	IP0.4	UART0	-	MULT/ACCU
3	IP0.3	Timer 1	-	I2C
2	IP0.2	-	-	SPI RX availlable
1	IP0.1	Timer 0 Interrupt	-	SPI TX Empty
0	IP0.0	External INT0	UART1	External INT 0

The WDTSTAT bit of the IP0 register is the watchdog status flag. This bit is set to 1 by the hardware whenever a watchdog timer overflow occurs. This bit must be cleared manually.

Setting-up the Watchdog Timer

Control of the watchdog timer is enabled by the following bits;

Bit	Location	Role
WDOGEN	DIGPWREN.6	Watchdog timer enable
WDTR	IEN0.6	Watchdog timer refresh flag
WDTS	IEN1.6	Watchdog timer Start bit

In order for the watchdog to begin counting, the user must set the WDOGEN bit (bit 6) of the DIGPWREN register as follows:

```
MOV     DIGPWREN,#x1xxxxxxB     ;x=0 or 1 depending
                                ;of other peripherals
                                ;to enable
```

The value written into the WDTREL register defines the delay time of the watchdog timer as follows:

WDT delay when the WDTREL bit 7 is cleared
WDT Delay = $\frac{24*[32768-(WDTREL(6:0) \times 256)]}{Fosc}$

WDT delay when the WDTREL bit 7 is set
WDT Delay = $\frac{384*[32768-(WDTREL(6:0) \times 256)]}{Fosc}$

The following table demonstrates WDT reload values and their corresponding delay times:

Fosc	WDTREL	WDT Delay
14.74MHz	00h	53.3ms
14.74MHz	4Fh	20.4ms
14.74MHz	CCh	347ms

Note: The value present in the CLKDIVCTRL register affects the watchdog timer delay time. The above equations and examples assume that the CLKDIVCTRL register content is 00h.

Starting the Watchdog Timer

To start the watchdog timer using the hardware automatic start procedure, the WDTS (IEN1) and WDTR (IEN0) bits must be set. The watchdog will begin to run with default settings i.e. all registers will be set to zero.

```
;*** Do a Watchdog Timer Refresh / Start sequence

SETB    IEN0.6          ;Set the WDTR bit first
SETB    IEN1.6          ;Then without delay set the
;WDTS bit
```

When the WDT registers enter the state 7FFFh, the asynchronous signal, WDTS will become active. This signal will set bit 6 in the IP0 register and trigger a reset.

To prevent the watchdog timer from resetting the VMX51C1016, reset it periodically by clearing the WDTR and clear the WDTS bit immediately afterward,.

As a security feature to prevent an inadvertent clearing of the watchdog timer, no delay (instruction) is allowed between the clearing of the WDTR and WDTS bits.
a) Watchdog timer refresh example 1:

```
*** The Simple way ***

MOV     IEN0,#x1xxxxxxB     ;DIRECT WRITE THAT SET BIT
                            ;WDTR (x = 0 or 1)
MOV     IEN1,#x1xxxxxxB     ;DIRECT WRITE THAT SET BIT
                            ;WDTS (x = 0 or 1)
```

In the case where the program makes use of the interrupts, it is recommended to deactivate the interrupts before the watchdog refresh is performed and reactivate them afterward.

b) Watchdog timer refresh example 2:

```
*** If Interrupts are used: ***

CLR     IEN0.7          ;Deactivate the interrupt
MOV     A,IEN0          ;Retrieve IEN0 content
ORL     A,#01000000B    ;set the bit 6 (WDTR)
XCH     A,R1            ;Store IENO New Value
MOV     A,IEN1          ;Retrieve IEN1 content
ORL     A,#01000000B    ;Set bit 6, (WDTS)
MOV     IEN0,R1         ; Set WDTR BIT
MOV     IEN1,A          ;Set WDTS BIT
SETB    IEN0.7          ;Reactivate the Interrupts
```

Watchdog Timer Reset

To determine whether the reset condition was caused by the watchdog timer, the state of the WDTSTAT bit of the IP0 register should be monitored. On a standard power-on reset condition, this bit is cleared.

WDT Initialization and Use Example Program

```
ORG 0000H                ;RESET & WD IRQ VECTOR
LJMP        START

;***************************************
;* MAIN PROGRAM BEGINNING  *
;***************************************

ORG     0100h
;*** Initialize WDT and other peripherals***
MOV         DIGPWREN,#40H          ;ENABLE WDT OPERATION

;*** INITIALIZE WATCHDOG TIMER RELOAD VALUE
MOV         WDTREL,#04FH           ;The WDTREL register is used to
                                   ;define the Delay Time WDT.
                                   ;Bit 7 of WDTREL define clock
                                   ;prescalng value
                                   ;Bit 6:0 of WDTREL defines the
                                   ;upper 7 bits reload value of the
                                   ;watchdog Timer 15-bit timer

;*** PERFORM A WDT REFRESH/START SEQUENCE
SETB        IEN0.6                 ;Set the WDTR bit first
SETB        IEN1.6                 ;Then without delay (instruction)
                                   ;set the WDTS bit right after.
                                   ;No Delays are permitted between
                                   ;setting of the WDTR bit and
                                   ;setting of the WDTS bit.
                                   ;This is a security feature to
                                   ;prevent inadvertent reset/start of
                                   ;the WDT

                                   ;IF other interrupt are enabled,
                                   ;It is recommended to disable
                                   ;interrupts before refreshing the
                                   ;WDT and reactivate them after

;*** Wait WDT Interrupt

WAITWDT:                NOP

;*** If the two following code lines below are  put "in-comment", the ;***WDT will
trigger a reset, and the program will restart.

;*** PERFORM A WATCHDOG TIMER REFRESH/START SEQUENCE
;SETB        IEN0.6                 ;Set the WDTR bit first
;SETB        IEN1.6                 ;Then without delay (instruction)
LJMP        WAITWDT                ;set the WDTS bit right after.
                                   ;No Delays are permitted between
                                   ;setting of the WDTR bit and
                                   ;setting of WDTS bit.
                                   ;This is a security feature to
                                   ;prevent inadvertent reset/start of
                                   ;the WDT
                                   ;It is recommended to disable
                                   ;interrupts before refreshing the ;WDT
                                   and reactivate them after
```

VMX51C1016 Programming

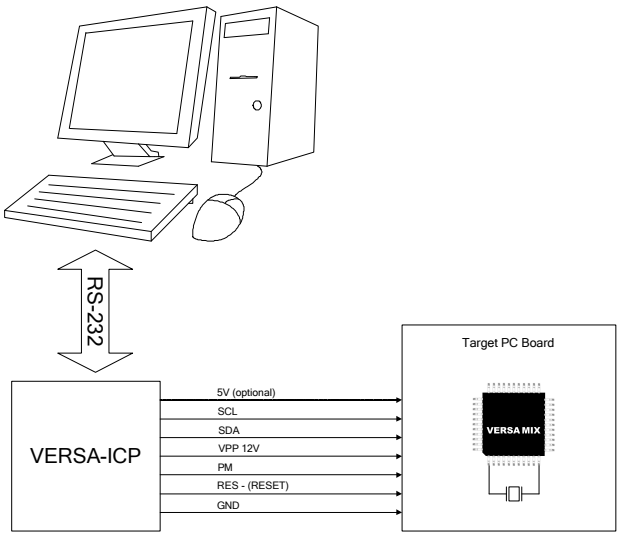
When the PM pin is set to 1, the I²C interface becomes the programming interface for the VMX51C1016's Flash memory.

An in-circuit programming interface is easy to implement at the board level. See VMIX APP-Note001.

Erasing and programming the VMX51C1016's Flash memory requires an external programming voltage of 12 volts. The voltage is supplied/controlled by the programming hardware/tools.

The VMX51C1016 can be programmed using Ramtron in-circuit programmers.

FIGURE 48: VMX51C1016 PROGRAMMING



VMX51C1016 Debugger

The VMX51C1016 includes hardware debugging features that can help speed up embedded software development time.

Debugger Features

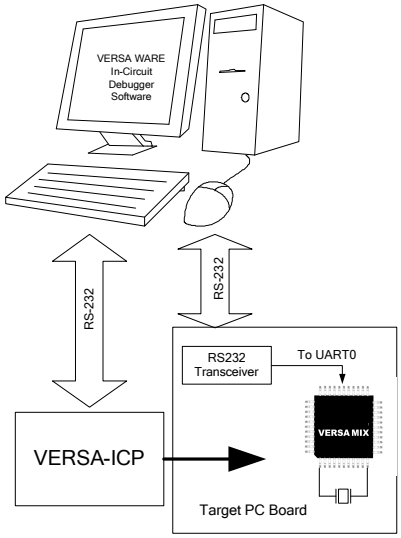
The VMX51C1016 debugger supports breakpoints and single-stepping of the user program. It supports retrieval and editing of the SFR register and RAM memory contents when a breakpoint is reached or when the device operates in single-step mode. Unlike ROM monitor programs that execute user program instructions at a much lower speed, the VMX51C1016 debugger does not affect program operating speed when in "Run Mode" before encountering a breakpoint.

Debugger Hardware Interface

The VMX51C1016's development system provides the ideal platform for running the VMX51C1016 debugger. Interfacing the VMX51C1016 debugger is done via the UART0 serial interface.

It is possible to run the VMX51C1016 debugger on the end user's PCB providing that access to the VMX51C1016 UART0 is available. However, a connection to a stand alone in-circuit programmer (ICP) will be required to perform Flash programming, control the reset line and activate the debugger on the target VMX51C1016 device.

FIGURE 49: VMX51C1016 DEBUGGER HARDWARE INTERFACE



Debugger Software Interface

The Versa Ware VMX51C1016 software running on Windows™ provides an easy-to-use user interface for in-circuit debugging.

For more details on the VMX51C1016 debugger, consult the *"Versa Ware VMX51C1016 – V1 Software User Guide.pdf"*

All documents are also accessible on the Ramtron web site at http://www.ramtron.com/

VMX51C1016 - 44 pin Quad Flat Package

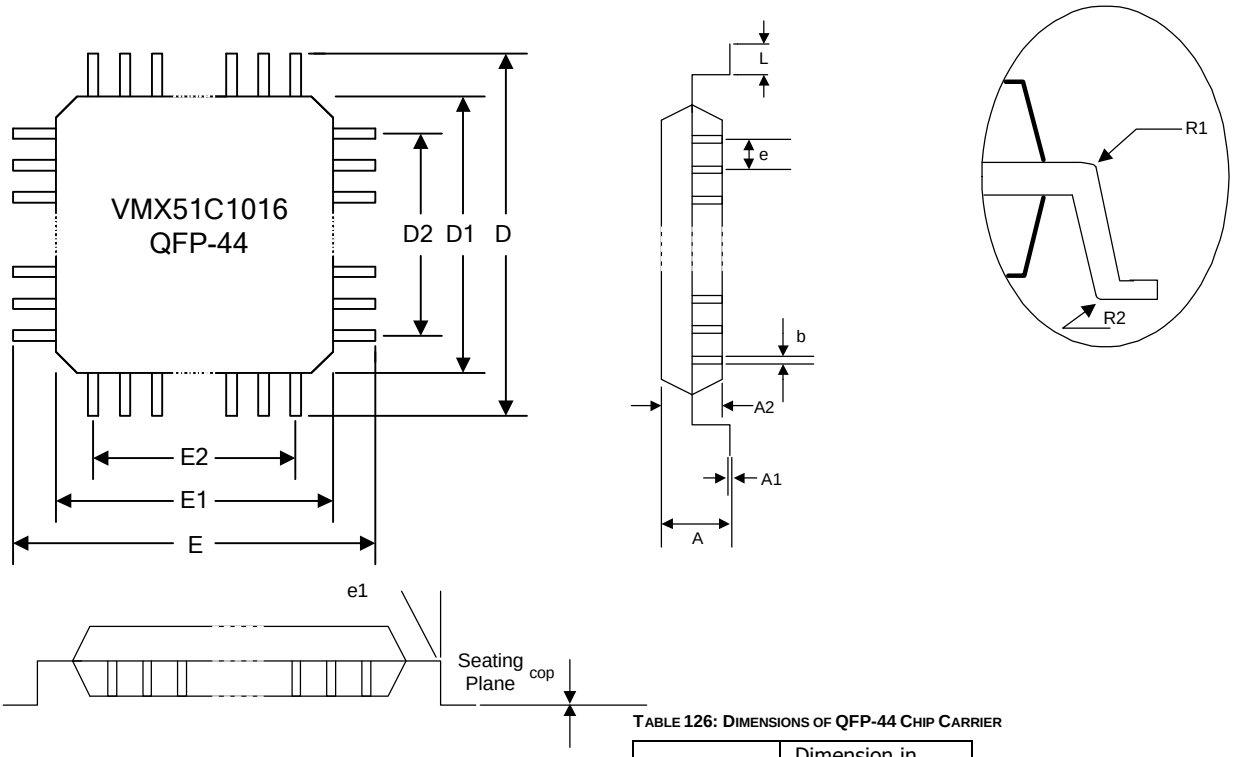


TABLE 126: DIMENSIONS OF QFP-44 CHIP CARRIER

Symbol	Dimension in mm
	Minimal/Maximal
A	-/2.45
Al	0.25/0.50
A2	1.95/2.10
b	0.30/0.40
e1	10º typ
D	13.20 BSC
D1	10.00 BSC
E	13.20 BSC
E1	10.00 BSC
e	0.80
L	0.78/1.03
cop	-/0.10
R1	0.2 RAD typ.
R2	0.3 RAD typ.