VRS51L3xxx

Datasheet *Rev 2.01*

High-Performance 8051 MCU + F-RAM

Overview

The VRS51L3xxx constitute a family of high performance, 8051-based microcontrollers coupled with a fully integrated array of peripherals for addressing a broad range of embedded design applications.

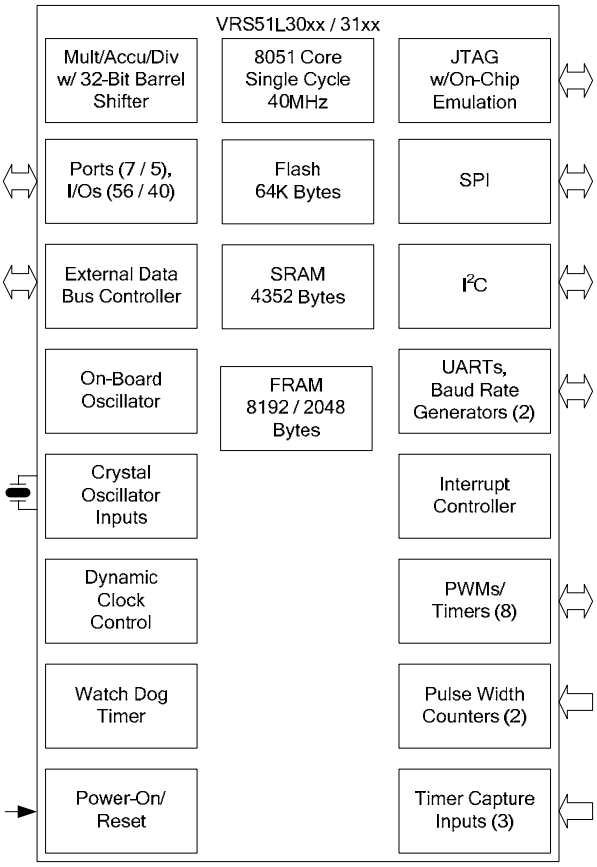
Based on a powerful 40-MIPS, single-cycle, 8051 microprocessor, these MCU's memory sub-system features 64KB of Flash and 4352 bytes of SRAM and 8192 or 2048 Bytes of nonvolatile FRAM (ferroelectric random access memory) memory.

Support peripherals include a hardware based arithmetic unit capable of performing complex mathematical operations, a JTAG interface for Flash programming and non-intrusive in-circuit debugging/emulation, an internal oscillator, and a watchdog timer.

Communication and control of external devices is facilitated via an assortment of digital peripherals such as an enhanced, fully configurable SPI bus, an I²C interface, dual UARTs with dedicated baud rate generators, three 16-bit timers, 8 PWM controllers each with a 16-bit timer, and 2 pulse width counter modules.

The VRS51L3xxx devices operate from 3.0 to 3.6 volts over the industrial temperature range and are available in QFP-64 (30xx) and QFP-44 (31xx) packages.

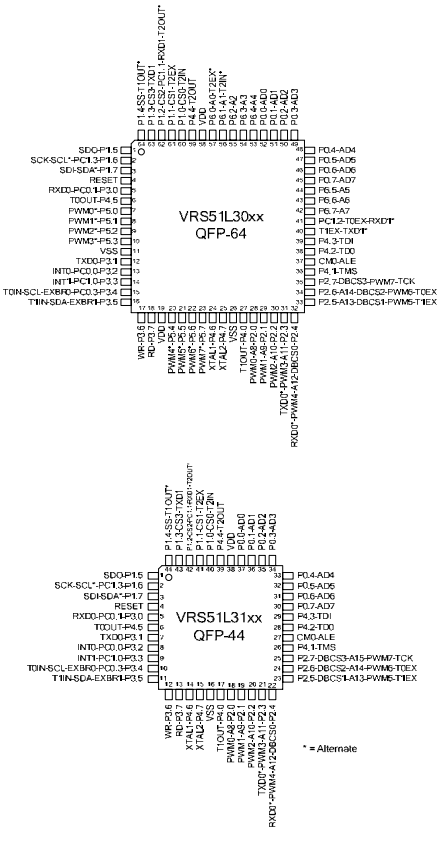
FIGURE 1: VRS51L30XX / 31XX FUNCTIONAL DIAGRAM



Feature Set

- 8051 High Performance Single Cycle Processor (Operation up to 40 MIPS)
- 64KB Flash Program Memory (In-System/In-Application Programmable)
- 4352 Bytes of SRAM (4KB + 256) (Ext. 4KB can be used for program or data memory)
- 8192 / 2048 Bytes of on-chip F-RAM memory
- JTAG Interface for Flash Programming and Non-Intrusive Debugging/In-Circuit Emulation
- MULT/DIV/ACCU Unit including Barrel Shifter
- 56 / 40 General Purpose I/Os (64/44-pin version)
- 2 Serial UARTs/2 Baud Rate Generators (20-bit)
- Enhanced SPI Interface (fully configurable word size)
- Fully Configurable I²C Interface (Master/Slave)
- 16 External Interrupt Pins/Interrupt On Port Pin Change
- 16-bit General Purpose Timer/Counters
- 2 Pulse Width Counter Modules
- 8 PWM Controller Outputs with Individual Timers
- PWMs can be used as General Purpose Timers
- Internal Oscillator
- Dynamic System Clock Frequency Adjustment
- Power Saving Features
- Power-On Reset/Brown-Out Detect
- Watchdog Timer
- Operating voltage: 3.0V to 3.6V
- Operating Temperature -40°C to +85°C

FIGURE 2: VRS51L30XX (QFP-64) / VRS51L31XX (QFP-44)

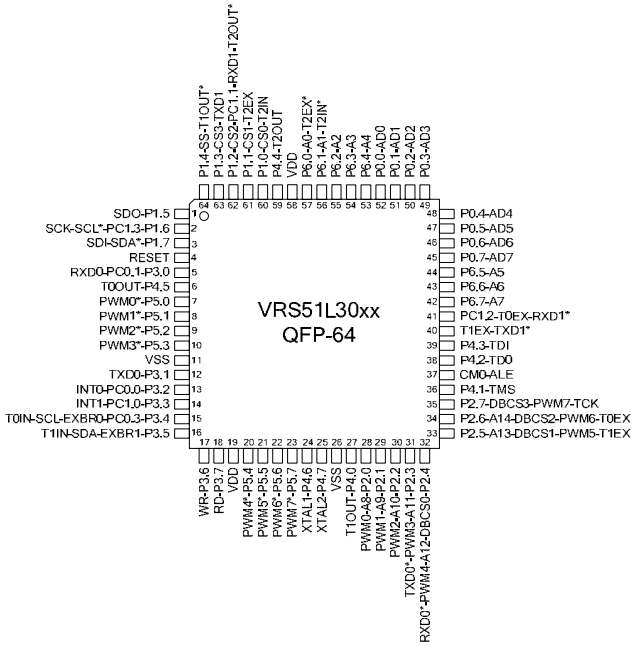


VRS51L30xx (QFP-64) Pin Description

TABLE 1: VRS51L30XX PIN DESCRIPTIONS FOR QFP-64 PACKAGE

QFP - 64	Name	I/O	Function
1	P1.5	I/O	Port 1.5
	SDO	O	SPI Data output
2	P1.6	I/O	Port 1.6
	SCK	O	SPI Clock
	SCL*	I/O	I²C Clock (Alternate Pin)
	PC1.3	I	Pulse Counter PC1 input 3
3	P1.7	I/O	Port P1.7
	SDI	I	SPI Data Input
	SDA*	I/O	I²C Data (Alternate Pin)
4	RESET	I	Reset
5	P3.0	I/O	Port 3.0
	RXD0	I	UART0 RX pin
	PC0.1	I	Pulse Counter PC0 input 1
6	P4.5	I/O	Port 4.5
	T0OUT	O	Timer 0 output
7	P5.0	I/O	Port 5.0
	PWM0*	O	PWM0 Output (Alternate Pin)
8	P5.1	I/O	Port 5.1
	PWM1*	O	PWM1 Output (Alternate Pin)
9	P5.2	I/O	Port 5.2
	PWM2*	O	PWM2 Output (Alternate Pin)
10	P5.3	I/O	Port 5.3
	PWM3*	O	PWM3 Output (Alternate Pin)
11	VSS	GND	Device ground
12	P3.1	I/O	Port 3.1
	TXD0	O	UART0 TX pin
13	P3.2	I/O	Port 3.2
	INT0	I	Interrupt 0 input
	PC0.0		Pulse Counter PC0 input 0
14	P3.3	I/O	Port 3.3
	INT1	I	Interrupt 1 input
	PC1.0		Pulse Counter PC1 input 0
15	P3.4	I/O	Port 3.4
	SCL	I/O	I²C clock
	T0IN	I	Timer 0 Input
	PC0.3	I	Pulse Counter PC0 input 3
	EXBR0	I	UART0 External Baud Rate Input
16	P3.5	I/O	Port 3.5
	SDA	I/O	I²C Data
	T1IN	I	Timer 1 Input
	EXBR1	I	UART1 External Baud Rate input
17	P3.6	I/O	Port 3.6
	WR	O	Ext Data memory access write signal (active low)
18	P3.7	I/O	Port 3.7
	RD	O	Ext Data memory access read signal (active low)
19	VDD	VDD	Positive supply
20	P5.4		Port 5.4
	PWM4*	O	PWM4 Output (Alternate Pin)
21	P5.5		Port 5.5
	PWM5*	O	PWM5 Output (Alternate Pin)
22	P5.6		Port 5.6
	PWM6*	O	PWM6 Output (Alternate Pin)
23	P5.7		Port 5.7
	PWM7*	O	PWM7 Output (Alternate Pin)
24	XTAL1	O	Crystal Oscillator (Output)
	P4.6	I/O	Port 4.6
25	XTAL2	I	Crystal Oscillator (Input)
	P4.7	I/O	Port 4.7
26	VSS	GND	Device ground
27	P4.0	I/O	Port 4.0
	T1OUT	O	Timer 1 Output
28	P2.0	I/O	Port 2.0
	PWM0	O	PWM0 Output
	A8	O	Ext. Address Bus A8
29	P2.1	I/O	Port 2.1
	PWM1	O	PWM1 Output
	A9	O	Ext. Address Bus A9
30	P2.2	I/O	Port 2.2
	PWM2	O	PWM2 Output
	A10	O	Ext. Address Bus A10
31	P2.3	I/O	Port 2.3
	PWM3	O	PWM3 Output
	TXD0*	O	UART0 TX pin (Alternate Pin)
	A11	O	Ext. Address Bus A11
32	P2.4	I/O	Port 2.4
	PWM4	O	PWM4 Output
	RXD0*	I	UART0 RX pin (Alternate Pin)
	PC0.2	I	Pulse Counter PC0 input 2
	A12	O	Ext. Address Bus A12
	DBCS0	O	Ext. Data bus DBCS0
33	P2.5	I/O	Port 2.5
	PWM5	O	PWM5 output
	T1EX	I	Timer 1 EX input
	A13	O	Ext. Address Bus A13
	DBCS1	O	Ext. Data bus DBCS1
34	P2.6	I/O	Port 2.6
	PWM6	O	PWM6 output
	T0EX	I	Timer 0 EX input
	A14	O	Ext. Address Bus A114
	DBCS2	O	Ext. Data bus DBCS2
35	P2.7	I/O	Port 2.7
	PWM7	O	PWM7 output
	TCK	I	JTAG TCK input
	DBCS3	O	Ext. Data bus DBCS3
36	P4.1	I/O	Port 4.1
	TMS	I	JTAG TMS Input
37	CM0	I	JTAG Program mode
	ALE	O	Ext Address Latch Enable
38	P4.2	I/O	Port 4.2
	TDO	O	JTAG TDO Line
39	P4.3	I/O	Port 4.3
	TDI	I	JTAG TDI line
40	TXD1*	O	UART1 TX pin (Alternate Pin)
	T1EX	I	Timer 1 EX input
41	RXD1*	I	UART1 RX pin (Alternate Pin)
	T0EX	I	Timer 0 EX input
	PC1.2	I	Pulse Counter PC1 input 2
42	P6.7	I/O	Port 6.7
	A7	O	Ext. Address 7 (Non-Multiplexed mode)
43	P6.6	I/O	Port 6.6
	A6	O	Ext. Address 6 (Non-Multiplexed mode)
44	P6.5	I/O	Port 6.5
	A5	O	Ext. Address 5 (Non-Multiplexed mode)

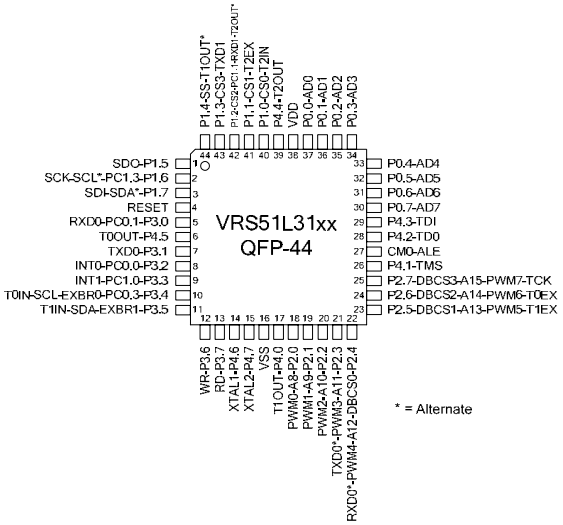
QFP - 64	Name	I/O	Function
45	P0.7	I/O	Port 0.7
	AD7	I/O	Ext. Address/Data Bus AD7
46	P0.6	I/O	Port 0.6
	AD6	I/O	Ext. Address/Data Bus AD6
47	P0.5	I/O	Port 0.5
	AD5	I/O	Ext. Address/Data Bus AD5
48	P0.4	I/O	Port 0.4
	AD4	I/O	Ext. Address/Data Bus AD4
49	P0.3	I/O	Port 0.3
	AD3	I/O	Ext. Address/Data Bus AD3
50	P0.2	I/O	Port 0.2
	AD2	I/O	Ext. Address/Data Bus AD2
51	P0.1	I/O	Port 0.1
	AD1	I/O	Ext. Address/Data Bus AD1
52	P0.0	I/O	Port 0.0
	AD0	I/O	Ext. Address/Data Bus AD0
53	P6.4	I/O	Port 6.4
	A4	O	Ext. Address 4 (Non-Multiplexed mode)
54	P6.3	I/O	Port 6.3
	A3	O	Ext. Address 3 (Non-Multiplexed mode)
55	P6.2	I/O	Port 6.2
	A2	O	Ext. Address 2 (Non-Multiplexed mode)
56	P6.1	I/O	Port 6.1
	A1	O	Ext. Address 1 (Non-Multiplexed mode)
	T2IN*	I	Timer 2 input (Alternate)
57	P6.0	I/O	Port 6.0
	A0	O	Ext. Address 0 (Non-Multiplexed mode)
	T2EX*	I	Timer 2 EX Input (Alternate)
58	VDD		Positive supply
59	P4.4	I/O	Port 4.4
	T2OUT	O	Timer 2 Output
60	P1.0	I/O	Port 1.0
	CS0	O	SPI Chip Select 0
	T2IN	I	Timer 2 input
61	P1.1	I/O	Port 1.1
	CS1	O	SPI Chip Select 1
	T2EX	I	Timer 2 EX input
62	P1.2	I/O	Port 1.2
	CS2	O	SPI Chip Select 2
	RXD1	I	UART1 RX line
	PC1.1	I	Pulse Counter PC1 input 1
	T2OUT	O	Timer 2 Output Pin (Alternate Pin)
63	P1.3	I/O	Port 1.3
	CS3	O	SPI Chip Select 3
	TXD1	O	UART1 TX line
64	P1.4	I/O	Port 1.4
	SS	I	SPI Slave Select input
	T1OUT*	O	Timer 1 Output (Alternate pin)



VRS51L31xx (QFP-44) Pin Description

TABLE 2: VRS51L31XX PIN DESCRIPTIONS FOR QFP-44 PACKAGE

QFP - 44	Name	I/O	Function
1	P1.5	I/O	Port 1.5
	SDO	O	SPI Data output
2	P1.6	I/O	Port 1.6
	SCK	O	SPI Clock
	SCL*	I/O	I²C Clock (Alternate Pin)
	PC1.3	I	Pulse Counter PC1 input 3
3	P1.7	I/O	Port P1.7
	SDI	I	SPI Data Input
	SDA*	I/O	I²C Data (Alternate Pin)
4	RESET	I/O	Reset
5	P3.0	I/O	Port 3.0
	RXD0	I	UART0 RX pin
	PC0.1	I	Pulse Counter PC0 input 1
6	P4.5	I/O	Port 4.5
	T0OUT	O	Timer 0 output
7	P3.1	I/O	Port 3.1
	TXD0	O	UART0 TX pin
8	P3.2	I/O	Port 3.2
	INT0	I	Interrupt 0 input
	PC0.0		
9	P3.3	I/O	Port 3.3
	INT1	I	Interrupt 1 input
	PC1.0		Pulse Counter PC1 input 0
10	P3.4	I/O	Port 3.4
	SCL	I/O	I²C clock
	T0IN	I	Timer 0 Input
	PC0.3	I	Pulse Counter PC0 input 3
	EXBR0	I	UART0 External Baud Rate Input
11	P3.5	I/O	Port 3.5
	SDA	I/O	I²C Data
	T1IN	I	Timer 1 Input
	EXBR1	I	UART1 External Baud Rate input
12	P3.6	I/O	Port 3.6
	WR	O	Ext Data memory access write signal (active low)
13	P3.7	I/O	Port 3.7
	RD	O	Ext Data memory access read signal (active low)
14	XTAL1	O	Crystal Oscillator (Output)
	P4.6	I/O	Port 4.6
15	XTAL2	I	Crystal Oscillator (Input)
	P4.7	I/O	Port 4.7
16	VSS	GND	Device ground
17	P4.0	I/O	Port 4.0
	T1OUT	O	Timer 1 Output
18	P2.0	I/O	Port 2.0
	PWM0	O	PWM0 Output
	A8	O	Ext. Address Bus A8
19	P2.1	I/O	Port 2.1
	PWM1	O	PWM1 Output
	A9	O	Ext. Address Bus A9
20	P2.2	I/O	Port 2.2
	PWM2	O	PWM2 Output
	A10	O	Ext. Address Bus A10
21	P2.3	I/O	Port 2.3
	PWM3	O	PWM3 Output
	TXD0*	O	UART0 TX pin (Alternate Pin)
	A11	O	Ext. Address Bus A11
22	P2.4	I/O	Port 2.4
	PWM4	O	PWM4 Output
	RXD0*	I	UART0 RX pin (Alternate Pin)
	PC0.2	I	Pulse Counter PC0 input 2
	A12	O	Ext. Address Bus A12
	DBCS0	O	Ext. Data Bus DSCS0
23	P2.5	I/O	Port 2.5
	PWM5	O	PWM5 output
	T1EX	I	Timer 1 EX input
	A13	O	Ext. Address Bus A13
	DBCS1	O	Ext. Data Bus DSCS1
24	P2.6	I/O	Port 2.6
	PWM6	O	PWM6 output
	T0EX	I	Timer 0 EX input
	A14	O	Ext. Address Bus A114
	DBCS2	O	Ext. Data Bus DSCS2
25	P2.7	I/O	Port 2.7
	PWM7	O	PWM7 output
	TCK	I	JTAG TCK input
	A15	O	Ext. Address Bus A15
	DBCS3	O	Ext. Data Bus DSCS3
26	P4.1	I/O	Port 4.1
	TMS	I	JTAG TMS Input
27	CM0	I	JTAG Program mode
	ALE	O	Ext Address Latch Enable
28	P4.2	I/O	Port 4.2
	TDO	O	JTAG TDO Line
29	P4.3	I/O	Port 4.3
	TDI	I	JTAG TDI line
30	P0.7	I/O	Port 0.7
	AD7	I/O	Ext. Address/Data Bus AD7
31	P0.6	I/O	Port 0.6
	AD6	I/O	Ext. Address/Data Bus AD6
32	P0.5	I/O	Port 0.5
	AD5	I/O	Ext. Address/Data Bus AD5
33	P0.4	I/O	Port 0.4
	AD4	I/O	Ext. Address/Data Bus AD4
34	P0.3	I/O	Port 0.3
	AD3	I/O	Ext. Address/Data Bus AD3
35	P0.2	I/O	Port 0.2
	AD2	I/O	Ext. Address/Data Bus AD2
36	P0.1	I/O	Port 0.1
	AD1	I/O	Ext. Address/Data Bus AD1
37	P0.0	I/O	Port 0.0
	AD0	I/O	Ext. Address/Data Bus AD0
38	VDD		Positive supply
39	P4.4	I/O	Port 4.4
	T2OUT	O	Timer 2 Output
40	P1.0	I/O	Port 1.0
	CS0	O	SPI Chip Select 0
	T2IN	I	Timer 2 input
41	P1.1	I/O	Port 1.1
	CS1	O	SPI Chip Select 1
	T2EX	I	Timer 2 EX input
42	P1.2	I/O	Port 1.2
	CS2	O	SPI Chip Select 2
	RXD1	I	UART1 RX line
	PC1.1	I	Pulse Counter PC1 input 1
	T2OUT	O	Timer 2 Output Pin (Alternate pin)
43	P1.3	I/O	Port 1.3
	CS3	O	SPI Chip Select 3
	TXD1	O	UART1 TX line
44	P1.4	I/O	Port 1.4
	SS	I	SPI Slave Select input
	T1OUT*		Timer 1 Output (Alternate pin)



1 Instruction Set

The following table describes the instruction set of the VRS51L3xxx. The instructions are binary code-compatible and perform the same functions as industry standard 8051s.

TABLE 3: 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 4: VRS51L3XXX INSTRUCTION SET

Mnemonic	Description	Size (bytes)	Instr. Cycles	Hex Code
Arithmetic instructions				
ADD A, Rn	Add register to A	1	2	28h-2Fh
ADD A, direct	Add direct byte to A	2	3	25h
ADD A, @Ri	Add data memory to A	1	3	26h-27h
ADD A, #data	Add immediate to A	2	2	24h
ADDC A, Rn	Add register to A with carry	1	2	38h-3Fh
ADDC A, direct	Add direct byte to A with carry	2	3	35h
ADDC A, @Ri	Add data memory to A with carry	1	3	36h-37h
ADDC A, #data	Add immediate to A with carry	2	2	34h
SUBB A, Rn	Subtract register from A with borrow	1	2	98h-9Fh
SUBB A, direct	Subtract direct byte from A with borrow	2	3	95h
SUBB A, @Ri	Subtract data mem from A with borrow	1	3	96h-97h
SUBB A, #data	Subtract immediate from A with borrow	2	2	94h
INC A	Increment A	1	2	04h
INC Rn	Increment register	1	2	08h-0Fh
INC direct	Increment direct byte	2	3	05h
INC @Ri	Increment data memory	1	3	06h-07h
DEC A	Decrement A	1	2	14h
DEC Rn	Decrement register	1	2	18h-1Fh
DEC direct	Decrement direct byte	2	3	15h
DEC @Ri	Decrement data memory	1	3	16h-17h
INC DPTR	Increment data pointer	1	2	A3h
MUL AB	Multiply A by B	1	2	A4h
DIV AB	Divide A by B	1	2	84h
DA A	Decimal adjust A	1	4	D4h
Logical Instructions				
ANL A, Rn	AND register to A	1	2	58h-5Fh
ANL A, direct	AND direct byte to A	2	3	55h
ANL A, @Ri	AND data memory to A	1	3	56h-57h
ANL A, #data	AND immediate to A	2	2	54h
ANL direct, A	AND A to direct byte	2	3	52h
ANL direct, #data	AND immediate data to direct byte	3	3	53h
ORL A, Rn	OR register to A	1	2	48h-4Fh
ORL A, direct	OR direct byte to A	2	3	45
ORL A, @Ri	OR data memory to A	1	3	46h-47h
ORL A, #data	OR immediate to A	2	2	44h
ORL direct, A	OR A to direct byte	2	3	42h
ORL direct, #data	OR immediate data to direct byte	3	3	43h
XRL A, Rn	Exclusive-OR register to A	1	2	68h-6Fh
XRL A, direct	Exclusive-OR direct byte to A	2	3	65h
XRL A, @Ri	Exclusive-OR data memory to A	1	3	66h-67h
XRL A, #data	Exclusive-OR immediate to A	2	2	64h
XRL direct, A	Exclusive-OR A to direct byte	2	3	62h
XRL direct, #data	Exclusive-OR immediate to direct byte	3	3	63h
CLR A	Clear A	1	1	E4h
CPL A	Compliment A	1	1	F4h
SWAP A	Swap nibbles of A	1	1	C4h
RL A	Rotate A left	1	1	23h
RLC A	Rotate A left through carry	1	1	33h
RR A	Rotate A right	1	1	03h
RRC A	Rotate A right through carry	1	1	13h
Boolean Instruction				
CLR C	Clear Carry bit	1	1	C3h
CLR bit	Clear bit	2	4	C2h
SETB C	Set Carry bit to 1	1	1	D3h
SETB bit	Set bit to 1	2	4	D2h
CPL C	Complement Carry bit	1	1	B3h
CPL bit	Complement bit	2	4	B2h
ANL C,bit	Logical AND between Carry and bit	2	4	82h
ANL C,#bit	Logical AND between Carry and not bit	2	4	B0h
ORL C,bit	Logical ORL between Carry and bit	2	4	72h
ORL C,#bit	Logical ORL between Carry and not bit	2	4	A0h
MOV C,bit	Copy bit value into Carry	2	4	A2h
MOV bit,C	Copy Carry value into Bit	2	3	92h
Data Transfer Instructions				
MOV A, Rn	Move register to A	1	2	E8h-EFh
MOV A, direct	Move direct byte to A	2	3	E5h
MOV A, @Ri	Move data memory to A	1	3	E6h-E7h
MOV A, #data	Move immediate to A	2	2	74h
MOV Rn, A	Move A to register	1	1	F8h-FFh
MOV Rn, direct	Move direct byte to register	2	3	A8h-AFh
MOV Rn, #data	Move immediate to register	2	2	78h-7Fh
MOV direct, A	Move A to direct byte	2	3	F5h
MOV direct, Rn	Move register to direct byte	2	3	88h-8Fh
MOV direct, direct	Move direct byte to direct byte	3	3	85h
MOV direct, @Ri	Move data memory to direct byte	2	3	86h-87h
MOV direct, #data	Move immediate to direct byte	3	3	75h
MOV @Ri, A	Move A to data memory	1	2	F6h-F7h
MOV @Ri, direct	Move direct byte to data memory	2	3	A6h-A7h
MOV @Ri, #data	Move immediate to data memory	2	2	76h-77h
MOV DPTR, #data	Move immediate to data pointer	3	3	90h
MOVC A, @A+DPTR	Move code byte relative DPTR to A	1	3+1	93h
MOVC A, @A+PC	Move code byte relative PC to A	1	3+1	83h
MOVX A,{MPAGE, @Ri}	Move external data (A8) to A	1	3*	E2h-E3h
MOVX A, @DPTR	Move external data (A16) to A	1	2*	E0h
MOVX {MPAGE, @Ri},A	Move A to external data (A8)	1	2*	F2h-F3h
MOVX @DPTR, A	Move A to external data (A16)	1	1*	F0h
PUSH direct	Push direct byte onto stack	2	3	C0h
POP direct	Pop direct byte from stack	2	2	D0h
XCH A, Rn	Exchange A and register	1	3	C8h-CFh
XCH A, direct	Exchange A and direct byte	2	4	C5h
XCH A, @Ri	Exchange A and data memory	1	4	C6h-C7h
XCHD A, @Ri	Exchange A and data memory lower nibble	1	4	D6h-D7h
Branching Instructions				
ACALL addr 11	Absolute call to subroutine	2	4+1	11h-F1h
LCALL addr 16	Long call to subroutine	3	5+1	12h
RET	Return from subroutine	1	3+1	22h
RETI	Return from interrupt	1	3+1	32h
AJMP addr 11	Absolute jump unconditional	2	2+1	01h-E1h
LJMP addr 16	Long jump unconditional	3	3+1	02h
SJMP rel	Short jump (relative address)	2	3+1	80h
JC rel	Jump on carry = 1	2	3+1	40h
JNC rel	Jump on carry = 0	2	3+1	50h
JB bit, rel	Jump on direct bit = 1	3	3 / 4 +1	20h
JNB bit, rel	Jump on direct bit = 0	3	3 / 4 +1	30h
JBC bit, rel	Jump on direct bit = 1 and clear	3	3 / 4 + 1	10h
JMP @A+DPTR	Jump indirect relative DPTR	1	2+1	73h
JZ rel	Jump on accumulator = 0	2	3+1	60h
JNZ rel	Jump on accumulator 1= 0	2	3+1	70h
CJNE A, direct, rel	Compare A, direct JNE relative	3	4 / 5 +1	B5h
CJNE A, #d, rel	Compare A, immediate JNE relative	3	3 / 4 +1	B4h
CJNE Rn, #d, rel	Compare reg, immediate JNE relative	3	3 / 4 +1	B8h-BFh
CJNE @Ri, #d, rel	Compare ind, immediate JNE relative	3	4 / 5 + 1	B6h-B7h
DJNZ Rn, rel	Decrement register, JNZ relative	2	3 / 4 +1	D8h-DFh
DJNZ direct, rel	Decrement direct byte, JNZ relative	3	3 / 4 +1	D5
Miscellaneous Instruction				
NOP	No operation	1	1	00h
NOP	If PCON.4 is 0 (reset Value): NOP	1	1	A5h
MOV @RamPtr,A	If MSB (@RamPtr) == 0 Accumulator value is written in SFR{1,@RamPtr[6:0]}	2	3	A5h
MOV A,@RamPtr	If MSB (@RamPtr) == 1 SFR{1,@RamPtr[6:0]} is written in Accumulator	3	4	A5h

Definitions

Rn: Any of the register R0 to R7
@Ri: Indirect addressing using Register R0 or R1
#data: immediate Data provided with Instruction
#data16: Immediate data included with instruction
bit: address at the bit level
rel: relative address to Program counter from +127 to –128
Addr11: 11-bit address range
Addr16: 16-bit address range
#d: Immediate Data supplied with instruction

Notes on number of Cycles:

"X / Y" cycles denotes number of cycle Without / With Jump
"+1" indicates extra Cycle that may be required because of Flash access

2 Special Function Registers (SFR)

Addresses 80h to FFh of the SFR address space can be accessed in direct addressing mode only. The following table lists the VRS51L3xxx special function registers. Due to the VRS51L3xxx's high level of integration, the SFRs have been mapped onto two pages.

The following tables summarize the SFR assignment. Complete functional descriptions of each register will be provided throughout the datasheet.

2.1 SFR Map Page 0

TABLE 5: SPECIAL FUNCTION REGISTERS (SFR) PAGE 0

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								DPSEL	0000 0000b
PCON	87h	OSCSTOP	INTMODEN	DEVCFGEN	SFRINDADR	GF1	GF0	PDOWN	IDLE	0110 0000b
INTEN1	88h	T1IEN	U1IEN	U0IEN	PCHGIEN0	T0IEN	SPIRXOVIEN	SPITXEIEN	-	0000 0000b
T0T1CFG	89h	-	T1GATE	T0GATE	T1CLKSRC	T1OUTEN	T1MODE8	T0OUTEN	T0MODE8	0000 0000b
TL0	8Ah									0000 0000b
TH0	8Bh									0000 0000b
TL1	8Ch									0000 0000b
TH1	8Dh									0000 0000b
TL2	8Eh									0000 0000b
TH2	8Fh									0000 0000b
P1	90h	-	-	-	-	-	-	-	-	1111 1111b
WDTCFG	91h	WDTPERIOD3	WDTPERIOD2	WDTPERIOD1	WDTPERIOD0	WTIMERF	ASTIMER	WDTF	WDTRESET	0000 0000b
RCAP0L	92h									0000 0000b
RCAP0H	93h									0000 0000b
RCAP1L	94h									0000 0000b
RCAP1H	95h									0000 0000b
RCAP2L	96h									0000 0000b
RCAP2H	97h									0000 0000b
P5*	98h									1111 1111b
T0T1CLKCFG	99h	T1CLKCFG3	T1CLKCFG2	T1CLKCFG1	T1CLKCFG0	T0CLKCFG3	T0CLKCFG2	T0CLKCFG1	T0CLKCFG0	0000 0000b
T0CON	9Ah	T0OVF	T0EXF	T0DOWNEN	T0TOGOUT	T0EXTEN	TR0	T0COUNTEN	T0RLCAP	0000 0000b
T1CON	9Bh	T1OVF	T1EXF	T1DOWNEN	T1TOGOUT	T1EXTEN	TR1	T1COUNTEN	T1RLCAP	0000 0000b
T2CON	9Ch	T2OVF	T2EXF	T2DOWNEN	T2TOGOUT	T2EXTEN	TR2	T2COUNTEN	T2RLCAP	0000 0000b
T2CLKCFG	9Dh	-	-	T2CLKSRC	T2OUTEN	T2CLKCFG3	T2CLKCFG2	T2CLKCFG1	T2CLKCFG0	0000 0000b
PWC0CFG	9Eh	PWC0IF	PWC0RST	PWC0END	PWC0START	PWC0ENDSRC1	PWC0ENDSRC0	PWC0STSRC1	PWC0STSRC0	0000 0000b
PWC1CFG	9Fh	PWC1IF	PWC1RST	PWC1END	PWC1START	PWC1ENDSRC1	PWC1ENDSRC0	PWC1STSRC1	PWC1STSRC0	0000 0000b
P2	A0h	-	-	-	-	-	-	-	-	1111 1111b
UART0INT	A1h	COLEN	RXOVEN	RXAVAILEN	TXEMPTYEN	COLENF	RXOVF	RXAVENF	TXEMPTYF	0000 0001b
UART0CFG	A2h	BRADJ3	BRADJ2	BRADJ1	BRADJ0	BRCLKSRC	B9RXTX	B9EN	STOP2EN	1110 0000b
UART0BUF	A3h									0000 0000b
UART0BRL	A4h									0000 0000b
UART0BRH	A5h									0000 0000b
UART0EXT	A6h	U0TIMERF	U0TIMEREN	U0RXSTATE	MULTIPROC	Reserved	Reserved	Reserved	Reserved	0010 0000b
Reserved	A7h									
INTEN2	A8h	PCHGIEN1	AUWDTIEN	PWMT47IEN	PWMT03IEN	PWCIEN	I2CUARTCI	I2CIEN	T2IEN	0000 0000b
PWMCFG	A9h	-	PWMWAIT	PWMCLRALL	PWMLSBMSB	PWMMIDEND	PWMCH2	PWMCH1	PWMCH0	0000 0000b
PWMEN	AAh	PWM7EN	PWM6EN	PWM5EN	PWM4EN	PWM3EN	PWM2EN	PWM1EN	PWM0EN	0000 0000b

PWMLDPOL	ABh	PWM7LDPOL	PWM6LDPOL	PWM5LDPOL	PWM4LDPOL	PWM3LDPOL	PWM2LDPOL	PWM1LDPOL	PWM0LDPOL	0000 0000b
PWMDATA	ACh									0000 0000b
PWMTMREN	ADh	PWM7TMREN	PWM6TMREN	PWM5TMREN	PWM4TMREN	PWM3TMREN	PWM2TMREN	PWM1TMREN	PWM0TMREN	0000 0000b
PWMTMRF	AEh	PWM7TMRF	PWM6TMRF	PWM5TMRF	PWM4TMRF	PWM3TMRF	PWM2TMRF	PWM1TMRF	PWM0TMRF	0000 0000b
PWMCLKCFG	AFh	U4PWMCLK3	U4PWMCLK2	U4PWMCLK1	U4PWMCLK0	L4PWMCLK3	L4PWMDCLK2	L4PWMCLK1	L4PWMCLK0	0000 0000b
P3	B0h	-	-	-	-	-	-	-	-	1111 1011b
UART1INT	B1h	COLEN	RXOVEN	RXAVAILEN	TXEMPTYEN	COLENF	RXOVF	RXAVENF	TXEMPTYF	0000 0001b
UART1CFG	B2h	BRADJ3	BRADJ2	BRADJ1	BRADJ0	BRCLKSRC	B9RXTX	B9EN	STOP2EN	1110 0000b
UART1BUF	B3h									0000 0000b
UART1BRL	B4h									0000 0000b
UART1BRH	B5h									0000 0000b
UART1EXT	B6h	U1TIMERF	U1TIMEREN	U1RXSTATE	MULTIPROC	0	0	0	0	0010 0000b
Not used	B7h									
IPINFLAG1	B8h	P37IF	P36IF	P35IF	P34IF	P31IF	P30IF	INT1IF	INT0IF	0000 0000b
PORTCHG	B9h	PMONFLAG1	PCHGMSK1	PCHGSEL1	PCHGSEL0	PMONFLAG0	PCHGMSK0	PCHGSEL1	PCHGSEL0	0000 0000b
P4	C0h									1111 1111b
SPICTRL	C1h	SPICLK2	SPICLK1	SPICLK0	SPICS1	SPICS0	SPICLKPH	SPICLKPOL	SPIMASTER	0000 0001b
SPICONFIG	C2h	SPIMANCS	SPIUNDERC	FSONCS3	SPILOADCS3	SPISLOW	SPIRXOVEN	SPIRXAVEN	SPITXEEN	0000 0000b
SPISIZE	C3h									0000 0111b
SPIRXTX0	C4h									0000 0000b
SPIRXTX1	C5h									0000 0000b
SPIRXTX2	C6h									0000 0000b
SPIRXTX3	C7h									0000 0000b
P6*	C8h									1111 1111b
SPISTATUS	C9h	SPIREVERSE	-	SPIUNDERF	SSPINVAL	SPINOCS	SPIRXOVF	SPIRXAVF	SPITXEMPF	0011 1001b
PSW	D0h	CY	AC	F0	RS1	RS0	OV	F1	P	0000 0000b
I2CCONFIG	D1h	MASTRARB	I2CRXOVEN	I2CRXAVEN	I2CTXEEN	I2CMASTART	I2CSCLLOW	I2CRXSTOP	I2CMODE	0000 0100b
I2CTIMING	D2h									0000 1100b
I2CIDCFG	D3h	I2CID6	I2CID5	I2CID4	I2CID3	I2CID2	I2CID1	I2CID0	I2CADVCFG	0000 0000b
I2CSTATUS	D4h	I2CERROR	I2CNOACK	I2CSDASYNC	I2CACKPH	I2CIDLEF	I2CRXOVF	I2CRXAVF	I2CTXEMPF	0010 1001b
I2CRXTX	D5h									0000 0000b
IPININV1	D6h	P37IINV	P36IINV	P35IINV	P34IINV	P33IINV	P32IINV	INT1IINV	INT0IINV	0000 0000b
IPININV2	D7h	P07IINV	P06IINV	P05IINV	P04IINV	P03IINV	P02IINV	P01IINV	P00IINV	0000 0000b
IPINFLAG2	D8h	P07IF	P06IF	P05IF	P04IF	P03IF	P02IF	P01IF	P00IF	0000 0000b
XMEMCTRL	D9h	EXTBUSCFG	EXTBUSCS	-	-	STRECH3	STRECH2	STRECH1	STRECH0	0000 0000b
Reserved	DAh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DBh	-	-	-	-	-	-	-	-	0000 0000b
FRAMCFG1	DCh	FREADIDLE	0	FRAMCLK1	FRAMCLK0	BURSTEN	FRAMOP1	FRAMOP0	RUNFRAMOP	1000 0000b
FRAMCFG2	DDH	0	0	0	0	FRAMBP1	FRAMBP0	FRAMWEL	0	0000 0000b
Reserved	DEh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DFH	-	-	-	-	-	-	-	-	0000 0000b
ACC	E0h	-	-	-	-	-	-	-	-	0000 0000b
DEVIOMAP	E1h	Reserved	PWMALTMAP	I2CALTMAP	U1ALTMAP	U0ALTMAP	T2ALTMAP	T1ALTMAP	T0ALTMAP	0000 0000b
INTPRI1	E2h	T1P37PRI	U1P36PRI	U0P35PRI	PC0P34PRI	T0P31PRI	SRP30PRI	STP33PRI	INT0P32PRI	0000 0000b
INTPRI2	E3h	PC1P00PRI	AUP06PRI	PTHP05PRI	PTLP04PRI	PWCP23PRI	I10P02PRI	I2CP01PRI	T2P00PRI	0000 0000b
INTSRC1	E4h	INTSRC1.7	INTSRC1.6	INTSRC1.5	INTSRC1.4	INTSRC1.3	INTSRC1.2	INTSRC1.1	INTSRC1.0	0000 0000b
INTSRC2	E5h	INTSRC2.7	INTSRC2.6	INTSRC2.5	INTSRC2.4	INTSRC2.3	INTSRC2.2	INTSRC2.1	INTSRC2.0	0000 0000b
IPINSENS1	E6h	P37ISENS	P36ISENS	P35ISENS	P34ISENS	P33ISENS	P32ISENS	INT1ISENS	INT0ISENS	0000 0000b
IPINSENS2	E7h	P07ISENS	P06ISENS	P05ISENS	P04ISENS	P03ISENS	P02ISENS	P01ISENS	P00ISENS	0000 0000b
GENINTEN	E8h	-	-	-	-	-	-	CLRPININT	GENINTEN	0000 0000b
FPICONFIG	E9h	FPILOCK1	FPILOCK0	FPIIDLE	FPIRDY	0	FPI8BIT	FPITASK1	FPITASK0	0000 0100b
FPIADDRL	EAh									0000 0000b
FPIADDRH	EBh									0000 0000b
FPIDATAL	ECh									0000 0000b

FPIDATAH	EDh									0000 0000b
FPICLKSPD	EEh					FPICLKSPD3	FPICLKSPD2	FPICLKSPD1	FPICLKSPD0	0000 0000b
Reserved	EFh	-	-	-	-	-	-	-	-	0000 0000b
B	F0h									0000 0000b
MPAGE	F1h									0000 0000b
DEVCLKCFG1	F2h	SOFTRESET	OSCSELECT	CLKDIVEN	FULLSPDINT	CLKDIV3	CLKDIV2	CLKDIV1	CLKDIV0	0110 0000b
DEVCLKCGF2	F3h	CYOSCEN	INTOSCEN	-	-	CYRANGE1	CYRANGE0	0	-	0100 1001b
PERIPHEN1	F4h	SPICSEN	SPIEN	I2CEN	U1EN	U0EN	T2EN	T1EN	T0EN	0000 0000b
PERIPHEN2	F5h	PWC1EN	PWC0EN	AUEN	XRAM2CODE	IOPORTEN	WDTEN	PWMSFREN	FPIEN	0000 1000b
DEVMEMCFG	F6h	EXTBUSEN	FRAMEN	-	-	-	-	-	SFRPAGE	0000 0000b
PORTINEN	F7h	Reserved (0)	P6INPUTEN	P5INPUTEN	P4INPUTEN	P3INPUTEN	P2INPUTEN	P1INPUTEN	P0INPUTEN	0111 1111b
USERFLAGS	F8h									0000 0000b
P0PINCFG	F9h	P07IN1OUT0	P06IN1OUT0	P05IN1OUT0	P04IN1OUT0	P03IN1OUT0	P02IN1OUT0	P01IN1OUT0	P00IN1OUT0	1111 1111b
P1PINCFG	FAh	P17IN1OUT0	P16IN1OUT0	P15IN1OUT0	P14IN1OUT0	P13IN1OUT0	P12IN1OUT0	P11IN1OUT0	P10IN1OUT0	1111 1111b
P2PINCFG	FBh	P27IN1OUT0	P26IN1OUT0	P25IN1OUT0	P24IN1OUT0	P23IN1OUT0	P22IN1OUT0	P21IN1OUT0	P20IN1OUT0	1111 1111b
P3PINCFG	FCh	P37IN1OUT0	P36IN1OUT0	P35IN1OUT0	P34IN1OUT0	P33IN1OUT0	P32IN1OUT0	P31IN1OUT0	P30IN1OUT0	1111 1111b
P4PINCFG	FDh	P47IN1OUT0	P46IN1OUT0	P45IN1OUT0	P44IN1OUT0	P43IN1OUT0	P42IN1OUT0	P41IN1OUT0	P40IN1OUT0	1111 1111b
P5PINCFG*	FEh	P57IN1OUT0	P56IN1OUT0	P55IN1OUT0	P54IN1OUT0	P53IN1OUT0	P52IN1OUT0	P51IN1OUT0	P50IN1OUT0	1111 1111b
P6PINCFG*	FFh	P67IN1OUT0	P66IN1OUT0	P65IN1OUT0	P64IN1OUT0	P63IN1OUT0	P62IN1OUT0	P61IN1OUT0	P60IN1OUT0	1111 1111b

*VRS51L30xx (QFP-64) only

2.2 SFR Map Page 1

TABLE 6: SPECIAL FUNCTION REGISTERS (SFR) PAGE 1

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								DPSEL	0000 0000b
PCON	87h	OSCSTOP	INTMODEN	DEVCFGEN	SFRINDADR	GF1	GF0	PDOWN	IDLE	0110 0000b
INTEN1	88h	T1IEN	U1IEN	U0IEN	PCHGIEN0	T0IEN	SPIRXOVIEN	SPITXEIEN	-	0000 0000b
T0T1CFG	89h	-	T1GATE	T0GATE	T1CLKSRC	T1OUTEN	T1MODE8	T0OUTEN	T0MODE8	0000 0000b
TL0	8Ah									0000 0000b
TH0	8Bh									0000 0000b
TL1	8Ch									0000 0000b
TH1	8Dh									0000 0000b
TL2	8Eh									0000 0000b
TH2	8Fh									0000 0000b
P1	90h	-	-	-	-	-	-	-	-	1111 1111b
WDTCFG	91h	WDTPERIOD3	WDTPERIOD2	WDTPERIOD1	WDTPERIOD0	WTIMERF	ASTIMER	WDTF	WDTRESET	0000 0000b
RCAP0L	92h									0000 0000b
RCAP0H	93h									0000 0000b
RCAP1L	94h									0000 0000b
RCAP1H	95h									0000 0000b
RCAP2L	96h									0000 0000b
RCAP2H	97h									0000 0000b
P5*	98h									1111 1111b
T0T1CLKCFG	99h	T1CLKCFG3	T1CLKCFG2	T1CLKCFG1	T1CLKCFG0	T0CLKCFG3	T0CLKCFG2	T0CLKCFG1	T0CLKCFG0	0000 0000b
T0CON	9Ah	T0OVF	T0EXF	T0DOWNEN	T0TOGOUT	T0EXTEN	TR0	T0COUNTEN	T0RLCAP	0000 0000b
T1CON	9Bh	T1OVF	T1EXF	T1DOWNEN	T1TOGOUT	T1EXTEN	TR1	T1COUNTEN	T1RLCAP	0000 0000b
T2CON	9Ch	T2OVF	T2EXF	T2DOWNEN	T2TOGOUT	T2EXTEN	TR2	T2COUNTEN	T2RLCAP	0000 0000b
T2CLKCFG	9Dh	-	-	T2CLKSRC	T2OUTEN	T2CLKCFG3	T2CLKCFG2	T2CLKCFG1	T2CLKCFG0	0000 0000b
Reserved	9Eh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	9Fh	-	-	-	-	-	-	-	-	0000 0000b
P2	A0h	-	-	-	-	-	-	-	-	1111 1111b
Reserved	A1h	-	-	-	-	-	-	-	-	0000 0000b
AUA0	A2h*									0010 0000b
AUA1	A3h*									0010 0000b
AUC0	A4h*									0010 0000b
AUC1	A5h*									0010 0000b
AUC2	A6h*									0010 0000b
AUC3	A7h*									0010 0000b
INTEN2	A8h	PCHGIEN1	AUWDTIEN	PWMT47IEN	PWMT03IEN	PWCIEN	I2CUARTCI	I2CIEN	T2IEN	0000 0000b
P3	B0h	-	-	-	-	-	-	-	-	1111 1011b
AUB0DIV	B1h*									0010 0000b
AUB0	B2h*									0010 0000b
AUB1	B3h*									0010 0000b
AURES0	B4h*									0010 0000b
AURES1	B5h*									0010 0000b
AURES2	B6h*									0010 0000b
AURES3	B7h*									0010 0000b
IPINFLAG1	B8h	P37IF	P36IF	P35IF	P34IF	P31IF	P30IF	INT1IF	INT0IF	0000 0000b

PORTCHG	B9h	PMONFLAG1	PCHGMSK1	PCHGSEL1	PCHGSEL0	PMONFLAG0	PCHGMSK0	PCHGSEL1	PCHGSEL0	0000 0000b
Reserved	BAh	-	-	-	-	-	-	-	-	*0001 0000b*
Reserved	BBh	-	-	-	-	-	-	-	-	*0000 0000b*
Reserved	BCh	-	-	-	-	-	-	-	-	*0000 0000b*
Reserved	BDh	-	-	-	-	-	-	-	-	*0000 0000b*
Reserved	BEh	-	-	-	-	-	-	-	-	
Reserved	BFh	-	-	-	-	-	-	-	-	
P4	C0h									1111 1111b
AUSHIFTCFG	C1h*	SHIFTMODE	ARITHSHIFT	SHIFT5	SHIFT4	SHIFT3	SHIFT2	SHIFT1	SHIFT0	0010 0000b
AUCONFIG1	C2h*	CAPPREV	CAPMODE	OVCAPEN	READCAP	ADDSRC1	ADDSRC0	MULCMD1	MULCMD0	0000 0000b
AUCONFIG2	C3h*	AUREGCLR2	AUREGCLR1	AUREGCLR0	AUINTEN	-	DIVOUTRG	AUOV16	AUOV32	0000 0000b
AUPREV0	C4h*									0000 0000b
AUPREV1	C5h*									0000 0000b
AUPREV2	C6h*									0000 0000b
AUPREV3	C7h*									0000 0000b
P6*	C8h									0000 0000b
Reserved	C9h	-	-	-	-	-	-	-	-	0000 0000b
Reserved	CAh	-	-	-	-	-	-	-	-	0000 0001b
Reserved	CBh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	CCh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	CDh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	CEh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	CFh	-	-	-	-	-	-	-	-	
PSW	D0h	CY	AC	F0	RS1	RS0	OV	-	P	0000 0000b
Reserved	D1h	-	-	-	-	-	-	-	-	
Reserved	D2h	-	-	-	-	-	-	-	-	
Reserved	D3h	-	-	-	-	-	-	-	-	
Reserved	D4h	-	-	-	-	-	-	-	-	
Reserved	D5h	-	-	-	-	-	-	-	-	
IPININV1	D6h	P37IINV	P36IINV	P35IINV	P34IINV	P33IINV	P32IINV	INT1IINV	INT0IINV	0000 0000b
IPININV2	D7h	P07IINV	P06IINV	P05IINV	P04IINV	P03IINV	P02IINV	P01IINV	P00IINV	0000 0000b
IPINFLAG2	D8h	P07IF	P06IF	P05IF	P04IF	P03IF	P02IF	P01IF	P00IF	0000 0000b
XMEMCTRL	D9h	EXTBUSCFG	EXTBUSCS	-	-	STRECH3	STRECH2	STRECH1	STRECH0	0000 0000b
Reserved	DAh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DBh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DCh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DDH	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DEh	-	-	-	-	-	-	-	-	0000 0000b
Reserved	DFh	-	-	-	-	-	-	-	-	0000 0000b
ACC	E0h	-	-	-	-	-	-	-	-	0000 0000b
DEVIOMAP	E1h	Reserved	PWMALTMAP	I2CALTMAP	U1ALTMAP	U0ALTMAP	T2ALTMAP	T1ALTMAP	T0ALTMAP	0000 0000b
INTPRI1	E2h	T1P37PRI	U1P36PRI	U0P35PRI	PC0P34PRI	T0P31PRI	SRP30PRI	STP33PRI	INT0P32PRI	0000 0000b
INTPRI2	E3h	PC1P00PRI	AUP06PRI	PTHP05PRI	PTLP04PRI	PWCP23PRI	I10P02PRI	I2CP01PRI	T2P00PRI	0000 0000b
INTSRC1	E4h	INTSRC1.7	INTSRC1.6	INTSRC1.5	INTSRC1.4	INTSRC1.3	INTSRC1.2	INTSRC1.1	INTSRC1.0	0000 0000b
INTSRC2	E5h	INTSRC2.7	INTSRC2.6	INTSRC2.5	INTSRC2.4	INTSRC2.3	INTSRC2.2	INTSRC2.1	INTSRC2.0	0000 0000b
IPINSENS1	E6h	P37ISENS	P36ISENS	P35ISENS	P34ISENS	P33ISENS	P32ISENS	INT1ISENS	INT0ISENS	0000 0000b
IPINSENS2	E7h	P07ISENS	P06ISENS	P05ISENS	P04ISENS	P03ISENS	P02ISENS	P01ISENS	P00ISENS	0000 0000b
GENINTEN	E8h	-	-	-	-	-	-	CLRPININT	GENINTEN	0000 0000b
FPICONFIG	E9h	FPILOCK1	FPILOCK0	FPIIDLE	FPIRDY	0	FPI8BIT	FPITASK1	FPITASK0	0000 0100b
FPIADDRL	EAh									0000 0000b
FPIADDRH	EBh									0000 0000b
FPIDATAL	ECh									0000 0000b
FPIDATAH	EDh									0000 0000b
FPICLKSPD	EEh					FPICLKSPD3	FPICLKSPD2	FPICLKSPD1	FPICLKSPD0	0000 0000b

Reserved	EFh	-	-	-	-	-	-	-	-	0000 0000b
B	F0h									0000 0000b
MPAGE	F1h									0000 0000b
DEVCLKCFG1	F2h	SOFTRESET	OSCSELECT	CLKDIVEN	FULLSPDINT	CLKDIV3	CLKDIV2	CLKDIV1	CLKDIV0	0110 0000b
DEVCLKCGF2	F3h	CYOSCEN	INTOSCEN	-	-	CYRANGE1	CYRANGE0	0	-	0100 1001b
PERIPHEN1	F4h	SPICSEN	SPIEN	I2CEN	U1EN	U0EN	T2EN	T1EN	T0EN	0000 0000b
PERIPHEN2	F5h	PWC1EN	PWC0EN	AUEN	XRAM2CODE	IOPORTEN	WDTEN	PWMSFREN	FPIEN	0000 1000b
DEVMEMCFG	F6h	EXTBUSEN	FRAMEN	-	-	-	-	-	SFRPAGE	0000 0000b
PORTINEN	F7h	Reserved (0)	P6INPUTEN	P5INPUTEN	P4INPUTEN	P3INPUTEN	P2INPUTEN	P1INPUTEN	P0INPUTEN	0111 1111b
USERFLAGS	F8h									0000 0000b
P0PINCFG	F9h	P07IN1OUT0	P06IN1OUT0	P05IN1OUT0	P04IN1OUT0	P03IN1OUT0	P02IN1OUT0	P01IN1OUT0	P00IN1OUT0	1111 1111b
P1PINCFG	FAh	P17IN1OUT0	P16IN1OUT0	P15IN1OUT0	P14IN1OUT0	P13IN1OUT0	P12IN1OUT0	P11IN1OUT0	P10IN1OUT0	1111 1111b
P2PINCFG	FBh	P27IN1OUT0	P26IN1OUT0	P25IN1OUT0	P24IN1OUT0	P23IN1OUT0	P22IN1OUT0	P21IN1OUT0	P20IN1OUT0	1111 1111b
P3PINCFG	FCh	P37IN1OUT0	P36IN1OUT0	P35IN1OUT0	P34IN1OUT0	P33IN1OUT0	P32IN1OUT0	P31IN1OUT0	P30IN1OUT0	1111 1111b
P4PINCFG	FDh	P47IN1OUT0	P46IN1OUT0	P45IN1OUT0	P44IN1OUT0	P43IN1OUT0	P42IN1OUT0	P41IN1OUT0	P40IN1OUT0	1111 1111b
P5PINCFG*	FEh	P57IN1OUT0	P56IN1OUT0	P55IN1OUT0	P54IN1OUT0	P53IN1OUT0	P52IN1OUT0	P51IN1OUT0	P50IN1OUT0	1111 1111b
P6PINCFG*	FFh	P67IN1OUT0	P66IN1OUT0	P65IN1OUT0	P64IN1OUT0	P63IN1OUT0	P62IN1OUT0	P61IN1OUT0	P60IN1OUT0	1111 1111b

*VRS51L30xx (QFP-64) only

2.3 Bit Accessible Registers

As is the case with standard 8051s, all SFR registers in which the lower nibble of address is x0 or x8, are bit-addressable. The bit-addressable registers allow bit-oriented instructions to alter individual register bit values.

TABLE 7:BIT ADDRESSABLE SFR 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
INTEN1	88h	T1IEN	U1IEN	U0IEN	PCHGIEN0	T0IEN	SPIRXOVIEN	SPITXEIEN	-	0000 0000b
P1	90h	-	-	-	-	-	-	-	-	1111 1111b
P5	98h									1111 1111b
P2	A0h	-	-	-	-	-	-	-	-	1111 1111b
INTEN2	A8h	PCHGIEN1	AUWDTIEN	PWMT47IEN	PWMT03IEN	PWCIEN	I2CUARTCI	I2CIEN	T2IEN	0000 0000b
P3	B0h	-	-	-	-	-	-	-	-	1111 1011b
IPINFLAG1	B8h	P37IF	P36IF	P35IF	P34IF	P31IF	P30IF	INT1IF	INT0IF	0000 0000b
P6	C8h									1111 1111b
PSW	D0h	CY	AC	F0	RS1	RS0	OV	-	P	0000 0000b
IPINFLAG2	D8h	P07IF	P06IF	P05IF	P04IF	P03IF	P02IF	P01IF	P00IF	0000 0000b
ACC	E0h	-	-	-	-	-	-	-	-	0000 0000b
GENINTEN	E8h	-	-	-	-	-	-		GENINTEN	0000 0000b
B	F0h									0000 0000b
USERFLAGS	F8h									0000 0000b

3 SFR Registers

3.1 Accumulator, B and User Flags Register

The VRS51L3xxx accumulator is located at address E0h on SFR pages 0 and 1. The accumulator is the source and destination for many 8051 instructions.

TABLE 8: THE ACCUMULATOR - ACC OR A SFR E0H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
ACC[7:0]							

The B register is mainly used for MUL and DIV instructions, holding the MSB of the MUL instruction and the remainder of the DIV instruction. It can also be used as a general purpose register that is bit-addressable. It is accessible on both SFR pages 0 and 1 at address F0h.

TABLE 9: B REGISTER - SFR F0H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
B[7:0]							

3.2 PSW Register

The PSW register is a bit-addressable register that contains the status flags (CY, AC, OV, P), user flag (F0) and register bank select bits (RS1, RS0) of the 8051 processor.

TABLE 10:THE PSW SFR REGISTER - PSW SFR D0H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	CY	Carry Bit Flag. Indicates that the last addition/subtraction resulted in a carry or borrow. The CY bit is cleared by other arithmetic instructions, the JBC and CLR C instructions.
6	AC	Auxiliary Carry Bit Flag. Indicates that the last addition/subtraction resulted in a carry or borrow from the higher nibble. The AC bit is cleared by other arithmetic instructions and by the JBC instruction.
5	F0	User General Purpose Flag
4:3	RS1:RS0	Register Select Address Bank for R0 – R7 00 R0 to R7 From 00h to 07h 01 R0 to R7 From 08h to 0Fh 10 R0 to R7 From 10h to 17h 11 R0 to R7 From 17h to 1Fh
2	OV	Overflow Flag Indicates that the last addition/subtraction resulted in a carry/borrow/overflow. The OV bit is cleared by other arithmetic instructions and the JBC instruction.
1	F1	User General Purpose Flag
0	P	Parity Flag

3.3 Data Pointers

The VRS51L3xxx includes two 16-bit data pointers that are described in the following tables. The active data pointer is controlled via a DPS register located at SFR address 86h (see below).

TABLE 11: DATA POINTER 0 HIGH - DPH0 SFR 83H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
DPTR0[15:8]							

TABLE 12: DATA POINTER 0 LOW - DPL0 SFR 82H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
DPTR0[7:0]							

TABLE 13: DATA POINTER 1 HIGH - DPH1 SFR 85H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
DPTR1[15:8]							

TABLE 14: DATA POINTER 1 LOW - DPL1 SFR 84H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
DPTR1[7:0]							

TABLE 15: DATA POINTER SELECT REGISTER - DPS SFR 86H

7	6	5	4	3	2	1	0
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:1	unused	
0	DPSEL	DPS value 0 : Selects DPTR 0 1 : Selects DPTR 1

3.4 Stack Pointer

The stack pointer is a register located at address 81h of the SFR register area whose value corresponds to the address of the last item that was put on the processor stack. Each time new data is put on the processor stack, the value of the stack pointer is incremented.

TABLE 16: STACK POINTER - SP SFR 81H

7	6	5	4	3	2	1	0
R/W, Reset = 0x07							
SP[7:0]							

By default, the stack pointer value is 07h. The stack can be set anywhere in the internal SRAM from address 00h to FFh.

Each time a function call is performed or an interrupt is serviced, the 16-bit return address (2 bytes) is stored on the stack. Data can be manually placed on the stack by using the PUSH and POP functions.

3.5 SFR Structure

The VRS51L3xxx peripheral registers are accessible through two SFR pages mapped directly into the 80h to FFh address range in the 256 bytes of memory. Most peripherals are accessible via both SFR pages. The following peripherals are only accessible via SFR Page 0:

- I²C Interface
- SPI Interface
- PWC Interface
- F-RAM Memory configuration

The enhanced arithmetic unit is only mapped onto SFR Page 1. To access SFR Page 1 registers, set the SFRPAGE bit of DEVMEMCFG register to 1, as shown below:

```
ORL     DEVMEMCFG,0x01  ;   SELECT SFR PAGE 1
```

Returning to SFR Page 0 is done by clearing SFRPAGE bit of the DEVMEMCFG register.

```
ANL     DEVMEMCFG,0xFEH     ;SELECT SFR PAGE 0
```

3.6 Indirect Addressing of the SFR

It is possible to access the SFR register in indirect addressing mode. Unique to the VRS51L3xxx, this feature enables efficient SFR content data transfers.

When the SFRINDADR bit 4 of the PCON register is set to 1, the A5h (NOP) instruction functions as an indirect SFR access.

Indirect SFR addressing uses the accumulator as well as the four bank Rn registers of SRAM memory area 00h to 1Fh to indirectly transfer the data to and from the SFR memory space.

Indirect SFR Register Write

For an indirect SFR write operation, perform the following steps after the SFRINDADR bit of the PCON register is set to 1:

- Write the data value into the accumulator.
- Hold the SFR address where the write operation is performed in the internal SRAM memory from address 00h to 1Fh.

The same SRAM memory area [00h to 1Fh] holds four sets of 8x Rn registers that are used for indirect addressing. Only one set of Rn registers is active at any given time and is defined by the value of the bits RS1 and RS0 of the PSW register.

For an indirect SFR write operation, bit 7 of the SFR address written into Rn must be cleared. For example, to write to the SPITX0 register located at address C4h, 44h should be written into the Rn register.

Example using the Bank 1, R0 register:

```
MOV     R0,#44    ;Target is SFR C4h (with Bit 7 stripped)
```

Example using the Bank 1, R3 register:

```
MOV     R3,#44    ;Target is SFR C4h (with Bit 7 stripped)
```

The next step involves calling the SFR indirect addressing function. This is a two-step process composed of the A5h instruction itself followed by the physical address of the Rn register, where the SFR address is stored. If the R0 register of Bank 1 has been used, the next instructions should be:

```
db. 0xA5
db. 0x00
```

If the R3 register of Bank 0 has been used, the next instructions should be:

```
db. 0xA5
db. 0x03
```

This would also work for the Rn registers located in Bank 4. For example, if the R0 register of Bank 4 contains the target SFR address, the instruction should be:

```
db. 0xA5
db. 0x18
```

Once the A5h instruction is executed, the processor will take the value stored in the accumulator and put it into the SFR address identified by the Rn register address.

```
;// Perform Indirect Write of Value 0xAA
;// into USERFLAGS SFR address (0xF8) using indirect SFR WRITE
    ORL 0x87, #0x10;        ;SET A5 for indirect SFR addressing
    MOV  0xF8,#00           ;Clear USERFLAGS
    MOV A, #0xAA            ;Acc = AAh
    MOV R0, #0x78           ;R0 (bank1) = address USERFLAGS (F8h)
                            ;with Bit 7 cleared
    .db 0xA5                ;Perform the indirect SFR write
    .db 0x00                ;After the second .db instruction,
                            ;P2 contains the value 0xAA
    ANL 0x87, #0xEF;        ;Set A5 for NOP operation
```

Indirect SFR Read

Indirect SFR address read functions similarly to indirect SFR write, the main differences being that the SFR target address stored in the Rn register is the actual SFR address (bit 7 = 1) with the accumulator containing the current SFR data.

```
;// Perform Indirect Read of Value in USERFLAGS SFR Address (0xF8)
;// into ACC using indirect SFR READ function
    ORL 0x87, #0x10;        ;SET A5 for indirect SFR addressing
    MOV A,#0x00             ;Acc = 00h
    MOV R0, #0xF8           ;R0 (bank1) = address P2 with Bit 7 cleared
   .db 0xA5                 ;Perform the indirect SFR Write
   .db 0x00                 ;After the second .db instruction,
                            ;Acc contain the value 0xAA
     ANL 0x87, #0xEF;       ;Set A5 for NOP operation
```

3.7 User Flags Register

The user flags register is a bit-addressable register used for condition testing or as a general purpose storage register.

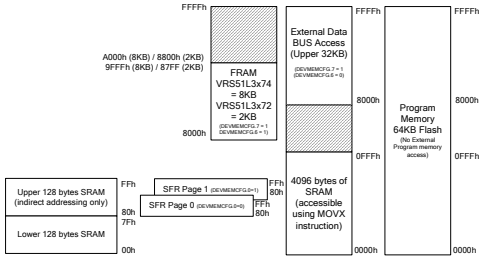
TABLE 17: USERFLAGS REGISTER - USERFLAGS SFR F8H

7	6	5	4	3	2	1	0
USERFLAGS, RESET = 0x00							
USERFLAGS[7:0]							

4 Memory Architecture

The following is an overview of the VRS51L3xxx's memory structure:

FIGURE 3: VRS51L3XXX DATA AND PROGRAM MEMORY STRUCTURE



The VRS51L3xxx devices include 64KB of on-chip Flash memory that can be used as program memory or as nonvolatile data storage.

The Flash memory is programmed via the JTAG or the FPI interface. The VRS51L3xxx members cannot be programmed in parallel mode (check FPI interface section for details).

4.1 Internal Scratch Pad SRAM (256 B)

As in standard 8051s, all VRS51L3xxx family members include 256 bytes of internal scratch pad SRAM: the lower 128 bytes are accessed by either direct or indirect addressing; the upper 128 bytes are accessed by indirect addressing only. Using direct addressing for the upper 128 bytes of scratch pad SRAM will access the SFR register area.

4.2 Integrated 4KB SRAM Block

The VRS51L3xxx devices include a 4KB block of SRAM that is mapped from address 0000h to 0FFFh on the external memory bus. This SRAM can be used for general purpose data memory or program memory.

The 4KB SRAM memory is always active and it is accessed using MOVX instructions.

Although the SRAM is mapped on the external memory, accessing it will have no impact on the I/O pins used for the external data memory bus.

4.3 Running Programs from the External 4KB SRAM Block

The VRS51L3xxx processors can execute code directly from the external 4KB of SRAM. Running the program from the SRAM memory can significantly save power, especially at lower operating frequencies. This is because SRAM power consumption is directly proportional to the access frequency, while power consumption of the Flash memory is less dependant of the device operating frequency.

To execute code from the 4KB SRAM block:

1. Copy the code from the Flash to the SRAM and apply the appropriate address shifting, if required.
2. Before switching to an XRAM operation, the program must execute from a Flash address higher than 0FFFh.
3. Set the XRAM2CODE bit (bit 4) of the PERIPHEN2 register.
4. Jump to the code copied into XRAM.

Code Example:

Copies code from the Flash memory to the XRAM memory and switches the program execution to the XRAM.

```
;--------VRS51L3xxx - Running program into XRAM  ---------------
;- DESCRIPTION: This program gives an examples on how
;-              to switch code execution from Flash to XRAM
;-------------------------------------------------------------------------------
include VRS51L3074_RIDE.inc

;---------- Variable definition -------------
CPTR          EQU 030h

   org      00000H
   LJMP       INIT

;-------------------------------------------------------------------
;---------------------- MAIN PROGRAM BEGINS ------------
;-------------------------------------------------------------------
INIT:       MOV PERIPHEN2,#08H       ;ENABLE IO
            MOV P1PINCFG,#00H        ;CONFIGURE P1 AS OUTPUT

      MOV PERIPHEN1,#00000000B;
   MOV    PERIPHEN2,#00001000B
                                ;BIT4 - XRAM2CODE = 0

;--COPY CODE FROM FLASH INTO XRAM MEMORY
   CLR DPS
   MOV    DPTR,#01000H       ;SET DPTR0 (POINT TO CODE)
   MOV    DPS,#01H           ;SWITCH TO DPTR1
   MOV    DPTR,#0000H        ;SET DPTR1 (POINT TO XRAM)

COPYLOOP:
   MOV    DPS,#00            ;POINT TO DPTR0 (FLASH)
   CLR    A
   MOVC  A,@A+DPTR           ;
   INC    DPTR               ;INC DPTR0 (FLASH)

   MOV    DPS,#01H           ;SWITCH TO DPTR1 (XRAM)
   MOVX  @DPTR,A             ;WRITE VALUE INTO XRAM
   INC    DPTR               ;INC dptr1 (XRAM)

   MOV    A,DPH1              ;CHECK IF DPTR1 (XRAM) REACH ADDRESS 0300H
   CJNE   A,#03,COPYLOOP
   LJMP   OUTSIDEXRAM         ;JUMP TO FLASH LOCATION OUTSIDE XRAM AREA
```

```
;-------------------------------------------------------------
;- SECTION OF CODE OUTSIDE THE XRAM
;-------------------------------------------------------------
ORG     2000H
OUTSIDEXRAM:
   MOV PERIPHEN2,#18H   ;ACTIVATE XRAM2CODE BIT AND IOPORTS
                        ;ANY JUMP TO THE 0000H - 0FFFH AREA SHOULD
                        EXECUTE FROM XRAM

   LJMP   0100H         ;JUMP TO THE P1 TOGGLE LOOP COPIED INTO XRAM
   MOV    P1,#00        ;FORCE P1 = 0X00H IF STUCK INTO THE FLASH
LOOP:   LJMP   LOOP     ;INFINITE LOOP

;-------------------------------------------------------------------------------
;-  Code to be moved into XRAM from address 0000h to 02FFH
;   ASSUMED CODE CONTAINED FROM 1000H TO 12FFH...
;   WILL BE COPIED FROM 0000H TO 02FFH INTO XRAM
;-------------------------------------------------------------------------------
;-------------------------------------
;- XRAM_Port_Toggle:
;-------------------------------------
org 1100h

TOGGLE:
   MOV    P1,#00H       ;SET PORT 1 = 00H
   LCALL  0200H         ;CALL DELAY FUNCTION
   MOV    P1,#0FFH      ;SET PORT 1 = FFH
   LCALL  0200H         ;CALL DELAY FUNCTION
   LJMP   0100H

org 1200h
;-------------------------------------------------------------
;- DELAY1MSTO : 1MS DELAY USING TIMER0
;-------------------------------------------------------------
DELAY1MS: MOV CPTR,#1

   MOV    A,PERIPHEN1       ;LOAD PERIPHEN1 REG
   ORL    A,#00000001B      ;ENABLE TIMER 0
   MOV    PERIPHEN1,A

DELAY1MSLP:
   MOV    TH0,#063H       ; TIMER0 RELOAD VALUE FOR 1MS AT 40MHZ
   MOV    TL0,#0C0H

   MOV    T0T1CLKCFG,#00H   ;NO PRESCALER FOR TIMER 0 CLOCK
   MOV    T0CON,#00000100B  ;START TIMER 0,  COUNT UP

DWAITOVT0:
   MOV    A,T0CON         ;READ TIMER 0 CONTROL, WAIT FOR OVERFLOW
   ANL    A,#080H           ;ISOLATE TIMER OVERFLOW FLAG
   JZ     DWAITOVT0         ;LOOP AS LONG AS TIMER 0 DONT OVERFLOW

   MOV    T0CON,#00H        ;STOP TIMER 0
   DJNZ   CPTR,DELAY1MSLP   ;

   MOV    A,PERIPHEN1          ;LOAD PERIPHEN1 REG
   ANL    A,#11111110B      ;DISABLE TIMER 0
   MOV    PERIPHEN1,A
   RET
```

4.4 Enabling F-RAM & External Data Memory Bus

The VRS51L3xxx devices provide access to the external data bus memory, enabling direct interfacing of the chip to external devices such as SRAM, data converters, etc. Activation of the external data memory bus, the F-RAM (ferroelectric random access memory) memory and the active SFR page is controlled via the DEVMEMCFG register.

TABLE 18: DEVICE MEMORY CONFIGURATION REGISTER - DEVMEMCFG SFR F6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	EXTBUSEN	When set this bit activates the external data bus access through Port 0, Port 2, P3.6 and P3.7
6	FRAMEN	To activate the F-RAM memory module both EXTBUSEN and the FRAMEN bit must be set to 1.
5:2	Not used	
0	SFRPAGE	When set, SFR Page 1 is selected

The EXTBUSEN bit of the DEVMEMCFG register controls the access devices connected to the external data memory bus shared with ports P0, P2, P3.6 and P3.7. When the EXTBUSEN bit is set to 1 and the FRAMEN bit is set to 0, any MOVX instructions with an address >= 0x8000 will activate the external data memory bus pins.

To activate the F-RAM memory module, set both the FRAMEN and the EXTBUSEN bits to 1. Any MOVX instructions with a target address from 0x8000 to 0x9FFF will then target the F-RAM memory and there will be no activity on ports P0, P2, P3.6 and P3.7. A roll-over to 0x8000 will occur if read or write operations to F-RAM are performed with a target address larger than 0x9FFF.

Bit 0 of the DEVMEMCFG defines the active SFR page.

Accessing the F-RAM memory has no impact on the I/O pins associated with the external data memory bus.

4.5 Integrated F-RAM Memory

The VRS51L3x74 devices include 8192 bytes of F-RAM mapped into 8000h to 9FFFh of the external memory address space.

The VRS51L3x72 devices include 2048 bytes of F-RAM mapped into 8000h to 87FFh of the external memory address space.

F-RAM memory is ideal for applications that require nonvolatile data storage.

Two SFR registers (FRAMCFG1 and FRAMCFG2) located at address DCh and DDh on SFR Page 0 (respectively) configure the F-RAM module's operation.

TABLE 19:F-RAM CONFIGURATION REGISTER 1 – FRAMCFG1 SFR DCH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	W	W	W	W
1	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	FREADIDLE	W: 1: Further accelerate F-RAM read when burst mode is activated (BURSTEN = 1) 0: Normal read mode
		R: 0: F-RAM module is busy 1: F-RAM module is IDLE
6	Reserved	This bit must bit must be kept to 0
5:4	FRAMCLK[1:0]	F-RAM module operation speed 00: Sysclk/2 01: Sysclk/3 10: Sysclk/4 11: Sysclk/8
3	BURSTEN	W: 1: Activate burst mode operation. The MOVX operations to F-RAM space are frozen until data has been read/written 0: Normal mode operation
		R: Read as 0
2:1	FRAMOP[1:0]	W: F-RAM Operation 00: Enable transfer of FRAMCFG2 defined parameters to F-RAM module. 01: Disable write operations from FRAMCFG2 to the F-RAM module 10: Prepare for read FRAMCFG2 register 11: Write FRAMCFG2 register to F-RAM module
		R: Read as 00
0	RUNFRAMOP	W : When this bit is set to 1, the selected F-RAM operation is executed
		R: Read as 0

Bit 7 of the FRAMCFG1 register (FREADIDLE) when read, indicates the status of the F-RAM module. Reading a 1 indicates that the F-RAM module is IDLE and ready to receive commands.

Writing a 1 into the FREADIDLE bit will activate fast read burst mode, as long as the BURSTEN bit is also set to 1.

FRAMCLK[1:0] bit 5,4 of the FRAMCFG1 register controls the F-RAM module operating clock frequency. By default, the F-RAM module operates at system clock/2. However, in certain cases it is preferable to lower the F-RAM memory module operating frequency to reduce burst speed operations allowing more time for data processing between F-RAM burst mode access.

TABLE 20:FRAMCLK[1:0] SETTING VS. F-RAM MODULE CLOCK

FRAMCLK[1:0] setting	F-RAM module Clock speed	F-RAM module frequency (Fosc =40MHz)
00	Sysclk/2	20MHz
01	Sysclk/3	13.3MHz
10	Sysclk/4	10MHz
11	Sysclk/8	5MHz

Bit 6 of the FRAMCFG1 register is reserved and must be written as 0 when write operations are performed.

When set to 1, bit 3 of the FRAMCFG1 register (BURSTEN) will activate burst mode, enabling faster data transfers to/from the F-RAM for both read/write operations to/from consecutive addresses (see the section on burst mode operations for more detail).

When burst mode is activated, writing a 1 to the FREADIDLE bit will activate fast burst read mode, which further accelerates F-RAM memory read operations.

The FRAMOP[1:0] and RUNFRAMOP bits of the FRAMCFG1 register are used to:

1. Initiate F-RAM module operations related to the activation/deactivation of the write protection feature on the F-RAM memory.
2. Read the status of the F-RAM's Write Enable Latch.

There are four operations controlled by the FRAMOP[1:0] bits, as described in the following table:

TABLE 21:F-RAM OPERATION ACCORDING TO FRAMOP[1:0] SETTING

FRAMOP[1:0]	F-RAM Operation
00	Enables transfer of FRAMCFG2 defined parameters to F-RAM module
01	Disables write operations from FRAMCFG2 to the F-RAM module
10	Updates the FRAMCFG2 register contents (read)
11	Transfers the contents of the FRAMCFG2 register to the F-RAM module

The FRAMOP[1:0] and the RUNFRAMOP bits work in conjunction with the FRAMCFG2 register.

The FRAMOP[1:0] bits define which operation will be performed. When set to 1, the RUNFRAMOP bit will initiate the operation selected by the FRAMOP[1:0] bit. The FRAMOP[1:0] and RUNFRAMOP bits can be written simultaneously or sequentially.

The F-RAM module requires a number of cycles to execute each operation. During that time, the processor continues to operate. In cases where read or write access to the F-RAM is initiated soon after a FRAMOP is executed, we recommend waiting until the FRAMIDLE bit is set to 1 before performing the operation.

The FRAMCFG2 register is used to enable the write protect option and monitor the current state of the block protect configuration and F-RAM Write Enable Latch flag (FRAMWEL bit). FRAMWEL is a read-only flag.

TABLE 22:F-RAM CONFIGURATION REGISTER 2 – FRAMCFG2 SFR DDH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	0	
6	0	
5	0	
4	0	
3:2	FRAMBP[1:0]	F-RAM Memory Block Protect 00: None 01: 8800h – 8FFFh (upper ¼ on VRS51L3x74) 10: 9000h – 9FFFh (upper ½ on VRS51L3x74) 11: 8000h – 9FFFh (all)
1	FRAMWEL	R: Indicates the state of the WEL flag
0	0	

FRAMCFG2 SFR mirrors the protection configuration register in the F-RAM memory. As such, the following steps are required to access the FRAMCFG2 register:

1) Write the configuration value into the FRAMCFG2 register.
2) Activate the write to FRAMCFG2 register operation by writing 07h into the FRAMCFG1 register.

Example:

```
MOV FRAMCFG2,#value to be written;
MOV FRAMCFG1,#0x07;
```

Similarly, before reading the FRAMCFG2 register, its contents should be refreshed by performing the following operations:

1) Activate the read from FRAMCFG2 register operation by writing 05h into the FRAMCFG1 register.
2) Read the contents of the FRAMCFG2 register.

Example:

```
MOV FRAMCFG1,#0x05;
MOV destination,FRAMCFG2;
```

The FREADIDLE bit of the FRAMCFG1 register should be monitored to ensure that the F-RAM module is in IDLE mode before initiating a read or write operation.

Example:

```
WAITILDE:   MOV A,FRAMCFG1;
            ANL   A,#0x80;
            JZ    WAITIDLE;
```

4.6 F-RAM Normal Mode Access

Access to the F-RAM memory requires using the MOVX instruction in the F-RAM memory address range. Before the F-RAM can be accessed, it must first be activated by setting both EXTBUSEN bit 7 and FRAMEN bit 6 of the DEVMEMCFG register located at address F6h on SFR Page 0 to 1.

The F-RAM access time, during which the processor is stopped, depends on the operating frequency of the processor, as well as the configuration of the FRAMCLK[1:0] bit of the FRAMCFG1 register.

The following table provides typical durations of read/write operations in various operating modes:

TABLE 23:F-RAM READ AND WRITE TIME

	FRAMCLK[1:0] = 00 (Sysclk/2)		FRAMCLK[1:0] = 10 (Sysclk/4)	
Mode	**Read**	**Write**	**Read**	**Write**
Normal	1.9uS	2.6uS	3.6uS	4.9uS
Burst*	1.1uS	0.4uS	2.3uS	0.7uS
Read Burst*	0.7uS	0.4uS	1.6uS	0.7uS

*Based on 100 consecutive read and write operations in burst mode

The processor's program counter will stop at the MOVX instruction while a F-RAM read or write is performed. As such, it is unnecessary to verify whether the device is idle (FREADIDLE = 1) before initiating a F-RAM read/write, unless an access to FRAMCFG2 was previously initiated.

Note: An interrupt that occurs during a F-RAM access will be serviced upon completion of the access operation.

Code Example:

Performing a F-RAM Read Operation

The following assembly code provides an example of a F-RAM memory read:

```
;----F-RAM Initialization

   ORL    DEVMEMCFG,#C0h    ;Activate the F-RAM module
   MOV    FRAMCFG1,#00h

;--Check F-RAM module is ready optional and needed only if a F-RAM operation  through
;   FRAMOP[1:0] have been initiated before.

FRAMRDY:      MOV    A,FRAMCFG1
              ANL    A,#80h            ;isolate FREADIDLE bit
              JZ     FRAMRDY           ;loop until FREADIDLE = 1
;--Performing a read operation at address 8100h
   FRAMWRITE: MOV    DPTR,#8100h
              MOVX     A,@DPTR
```

Code Example:

Performing a F-RAM Write Operation

```
;----F-RAM initialization
              ORL    DEVMEMCFG,#C0h    ;Activate the F-RAM module
              MOV    FRAMCFG1,#01h     ;Set the write enable Latch

;--Check F-RAM module is ready optional and needed only if a F-RAM operation  through
;   FRAMOP[1:0] have been initiated before.

FRAMRDY:      MOV    A,FRAMCFG1
              ANL    A,#80h            ;isolate FREADIDLE bit
              JZ     FRAMRDY           ;loop until FREADIDLE = 1
              MOV    A,DATA            ;retrieve data to be written in F-RAM

;--Performing a write operation at address 8100h
              MOV    DPTR,#8100h
              MOVX  @DPTR,A
```

4.7 F-RAM Burst Mode Operation

The access to the F-RAM memory can be configured to operate in burst mode, enabling faster data transfers from/to the F-RAM memory for consecutive address read/write operations.

F-RAM Burst Write

For write operations, burst mode is activated by setting the BURSTEN bit of the FRAMCFG1 register to 1.

The burst mode operation takes advantage of the double buffering capability of the F-RAM memory module. This allows the processor to write the next data byte to the F-RAM memory module before completion of the current write cycle.

Operating in burst write mode requires that the following conditions be met:

- The MOVX write operation to the F-RAM must be performed on consecutive incremental addresses.
- The next MOVX write instruction to the F-RAM must be performed within a predefined number of system clocks.
- Once initiated, only F-RAM write operations can be performed. The program cannot perform a F-RAM write operation and then a F-RAM read operation without exiting burst mode.

The table below shows the number of system clock cycles allowed between MOVX write instructions to the F-RAM memory in burst mode:

TABLE 24:F-RAM BURST WRITE – MAX NUMBER OF CYCLES NEXT MOVX INSTRUCTION

FRAMCLK[1:0] setting	Number of processor cycles for next MOVX write
00	13
01	20
10	28
11	58

Failure to provide the next data byte within the required time, or performing a MOVX write operation at a nonconsecutive address will cause data loss. There is no flag or interrupt to indicate that such a condition has occurred.

Code Example: F-RAM Burst Mode Write

```
              MOV    DEVMEMCFG,#0C0h;Enable F-RAM
              (…)
              MOV    FRAMCFG1,#08h;   ;Set F-RAM in burst mode
              MOV    R0,#0A0h         ;Initialize a pointer to IRAM
              MOV    R2,#100          ;prepare to perform 100 write operations
              MOV    DPTR,#08000h     ;Set DPTR to point to F-RAM first address

BURSTREAD:    MOV    A,@R0            ;retrieve data from IRAM (3)
              MOVX   @DPTR,A          ;Copy Data into F-RAM (uP will stop)
              INC    DPTR             ;point to next F-RAM address (2)
              INC    R0               ;point to next IRAM address (2)
              DJNZ   R2,BURSTREAD     ;Perform 100 write (3)
```

F-RAM Burst Read

The F-RAM memory module offers two burst read modes (basic and fast).

The basic F-RAM burst read mode is activated by setting the BURSTEN bit of the FRAMCFG1 register to 1.

Fast F-RAM burst read mode is activated by setting both the BURSTEN and the FREADIDLE bits of the FRAMCFG1 register to 1.

The burst mode operation takes advantage of the double buffering capability of the F-RAM memory module. This allows the F-RAM module to prepare the next data byte to be read before the current read cycle is complete. As is the case for F-RAM burst write mode, burst read mode requires the following similar conditions to be met:

- MOVX read operation to the F-RAM must be performed on consecutive incremental addresses.
- The next MOVX read instruction from the F-RAM must be performed within a predefined number of system clocks. When the FREADIDLE bit is set to 1, the number of system clock cycles allowed between read instructions is reduced by a factor of 2.
- Once initiated, only F-RAM read operations can be performed. The program cannot perform F-RAM read operations and then F-RAM write operations without exiting burst mode.

The table below shows the number of system clock cycles allowed between MOVX read instructions to the F-RAM memory in burst mode:

TABLE 25:F-RAM BURST READ – MAX NUMBER OF CYCLES NEXT MOVX INSTRUCTION

FRAMCLK[1:0] setting	Number of cycles for next MOVX read instruction (FREADIDLE = 1 / 0)
00	14/28
01	22/43
10	30/58
11	62/118

Failure to perform a MOVX read within the predefined number of cycles or performing a MOVX read operation at a nonconsecutive address will cause data loss. There is no flag or interrupt to indicate that such a condition has occurred.

Code Example: F-RAM Burst Mode Read

```
        MOV   DEVMEMCFG,#0C0h;Enable F-RAM
        (…)
        MOV   FRAMCFG1,#88h;  ;Set F-RAM in burst mode + Burst Read mode
        MOV   R0,#0A0h        ;Initialize a pointer to IRAM
        MOV   R2,#100         ;prepare to perform 100 read operations
        MOV   DPTR,#08000h    ;Set DPTR to point to F-RAM first address

BURSTREAD: MOVX A,@DPTR       ;Fetch Data from F-RAM (uP will stop)
        MOV   @R0,A           ;Write data into IRAM (3)
        INC   DPTR            ;point to next F-RAM address (2)
        INC   R0              ;point to next IRAM address (2)
        DJNZ  R2,BURSTREAD    ;Perform 100 Read (3)
```

Exiting F-RAM Burst Mode

The FREADIDLE bit will remain at 0 as long as the device is in burst mode. Exiting burst mode may be required in the following instances:

- Changing the target address of a read or write operation to a nonconsecutive one.
- Changing the operation from read to write or write to read.

Exit burst mode by monitoring the FREADIDLE bit of the FRAMCFG1 register until it returns to 1, corresponding to the IDLE condition.

4.8 F-RAM Write Protect

Two methods can be employed to enable the F-RAM write protect feature. The first involves configuring the F-RAM access as read-only when the device is programmed. This can be executed via the options menu in Versa Ware JTAG software. This method supersedes the F-RAM block protection feature.

The second method is via the processor, by accessing the FRAMCFG1 and FRAMCFG2 registers. The values written to the FRAMBP[1:0] bits of the FRAMCFG2 register define which region of the F-RAM memory will be protected:

TABLE 26:F-RAM BLOCK PROTECT

FRAMBP[1:0]	Protected F-RAM Addresses
00	None
01	8800h – 8FFFh (upper ¼ on VRS51L3x74)
10	9000h – 9FFFh (upper ½ on VRS51L3x74)
11	8000h – 9FFFh (all)

The following steps are required to apply F-RAM memory protection from the processor:

1. Write 01h to the FRAMCFG1 register to activate the FRAMWEL bit (enables update of FRAMCFG2 register).
2. Read the FREADIDLE bit 7 of the FRAMCFG1 register and wait until it reaches 1, indicating that the FRAM module is idle.
3. Configure the value of the FRAMCFG2 register: Set the FRAMBP[1:0] bits to select which zone of the F-RAM should be write-protected and set the FRAMWP bit to 1.
4. Write 0x07 to the FRAMCFG1 register to execute the F-RAM module write protect operation.
5. Read the FREADIDLE bit 7 of the FRAMCFG1 register and wait until it reaches 1, indicating that the F-RAM module is idle.
6. Write 03h into the FRAMCFG1 register to deactivate access to write to the FRAMCFG2, (protection from inadvertent writes).
7. Read the FRAMCFG2 register to verify that the block protect operation was successful.

Code Example: F-RAM Block Protect

The following is a code example to perform a F-RAM memory block protect:

```
//-------------------------------------------------------------------------------------------------//
// void FramProtect(char frambp)   //
//-------------------------------------//
// Description: Apply Block Protect on V3K F-RAM
//
// Input parameters:
//  char value corresponding to the FRAMBP[1:0] of FRAMCFG2 register
//
// Output parameters:
//  None
//-------------------------------------------------------------------------------------------------//
void FramProtect(char frambp)
   {

   frambp &= 0x03;                  //FRAMBP is 2 bit only
   frambp = frambp << 2;            //shift  frambp value  to position at bits 3:2

   FRAMCFG1 = 0x01;                 //Set FRAMWEL = 1 Enable Write  (FRAMOP = 00)
   while(!(FRAMCFG1&0x80));         //Wait FREADIDLE == 1  (FRAM IDLE)

   FRAMCFG2 = frambp;               //Transfer frambp value into FRAMCFG2

   FRAMCFG1 = 0x07;                 //Execute Transfer of FRAMCFG2 register content
                                    //to F-RAM Module
   while(!(FRAMCFG1&0x80));         //Wait FREADIDLE == 1  (FRAM IDLE)

   FRAMCFG1 = 0x03;                 //Clear FRAMWEL (FRAMOP = 01)
   while(!(FRAMCFG1&0x80));         //Wait FREADIDLE == 1  (FRAM IDLE)

   // Reading the WEL and the BP bits (optional)
   //FRAMCFG1 = 0x05;               //read FRAMCFG2
   //while(!(FRAMCFG1&0x80));       //Wait FREADIDLE == 1  (F-RAM IDLE)

   }//end of FramProtect
```

Code Example: F-RAM Access in C Example:

The following program provides examples of F-RAM access in C.

```
//-------------------------------------------------------------------------------------------------//
// V3K_FRAM_Use_Example_SDCC.c  //
//-----------------------------------------------------//
//
// This program show how to perform the following operations:
//
// -Enables the F-RAM memory
// -Deactivate the F-RAM protection (This step is optional)
// -Fill the F-RAM with 0x55
// -Read F-RAM address 0x8100
// -Write 0x23 at F-RAM address 0x8100
// -Read Content of address 0x8100
// - Activate F-RAM Write Protect
// -"Try" clearing the (protected) F-RAM
// -Deactivate the F-RAM protection
// -Clear the F-RAM
// -Show how to Read the F-RAM Block protect configuration
//-------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

//--Init pointer to F-RAM base address
xdata at 0x8000 unsigned char frambase;          //Init  a char variable pointing to F-RAM
xdata unsigned char * data framptr = &frambase ; //Init a pointer in IRAM pointing to the F-
RAMbase var.

//-------------------------------------------//
//          MAIN FUNCTION      //
//-------------------------------------------//
 void main (void){
   volatile idata int  cptr= 0x00;                //general purpose counter
   volatile idata char framread = 0x00;

   DEVMEMCFG  |= 0xC0;                            //Activate the F-RAM

//-- Deactivate FRAM Write Protect (Not needed if F-RAM not initialy protected)
   FRAMCFG1 = 0x01;                               //Set FRAMWEL = 1
                                                  //Enable Write  (FRAMOP = 00)
   while(!(FRAMCFG1&0x80));                       //Wait FREADIDLE == 1  (F-RAM IDLE)

   FRAMCFG2 = 0x00;                               //Configure FRAMCFG2 to remove
                                                  //F-RAM content Protection
   FRAMCFG1 = 0x07;                               //Execute Transfert of FRAMCFG2
                                                  //module to F-RAM Module
```

```
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

    FRAMCFG1 = 0x03;                            //Disable the write operations
                                                //from F-RAMCFG2 to the F-RAM Module
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

//--Fill the F-RAM with 0x55
    for(cptr = 0; cptr < 0x2000; cptr++)
        *(framptr+cptr) = 0x55;

//--Read Content of address 0x8100 and place it into framread
        framread = *(framptr + 0x0100)          //framread will contain 0x55

//--Write 0x23 at address 0x8100 in F-RAM (offset of 0x0100)
        *(framptr + 0x0100) = 0x23;

//--Read Content of address 0x8100 and place it into framread
        framread = *(framptr + 0x0100);         //framread will contain 0x23
         framread = *(framptr + 0x0101);        //framread will contain 0x55


//-- Activate F-RAM Write Protect

    FRAMCFG1 = 0x01;                            //Set FRAMWEL = 1 Enable Write
                                                //(FRAMOP = 00)
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

    FRAMCFG2 = 0x0C;                            //Configure FRAMCFG2 to protect
                                                //the entire F-RAM

    FRAMCFG1 = 0x07;                            //Execute Transfert of FRAMCFG2 module
                                                //to F-RAM Module
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (FRAM IDLE)

    FRAMCFG1 = 0x03;                            //Clear FRAMWEL (FRAMOP = 01)
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

//--Clear the F-RAM content (will not work if F-RAM protected)
    for(cptr = 0; cptr < 0x2000; cptr++)
        *(framptr+cptr) = 0x00;

//-- Deactivate F-RAM Write Protect (Not needed if F-RAM not initially protected)
    FRAMCFG1 = 0x01;                            //Set FRAMWEL = 1
                                                //Enable Write  (FRAMOP = 00)
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

    FRAMCFG2 = 0x00;                            //Configure F-RAMCFG2 to remove
                                                //F-RAM content Protection
    FRAMCFG1 = 0x07;                            //Execute Transfer of FRAMCFG2 module to
                                                //F-RAM Module
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

    FRAMCFG1 = 0x03;                            //Disable the write operations
                                                //from FRAMCFG2 to the F-RAM Module
    while(!(FRAMCFG1&0x80));                    //Wait FREADIDLE == 1  (F-RAM IDLE)

//--Clear the F-RAM content (Will work unless F-RAM is configured as Read Only at
programming time )
    for(cptr = 0; cptr < 0x2000; cptr++)
        *(framptr+cptr) = 0x00;

    while(1);

//--Optional Read of the F-RAM Block protect configuration
    //FRAMCFG1 = 0x05;                  //read FRAMCFG2
    //while(!(FRAMCFG1&0x80));          //Wait FREADIDLE == 1  (F-RAM IDLE)
    //x = FRAMCFG2                      //Read the FRAMCFG2 register

    //(...)

}//end of Main
```

4.9 External Data Bus Access for VRS51L30xx devices (QFP-64)

The VRS51L30xx provides external memory bus access on the upper 32KB block of the 64KB external memory [8000h to FFFFh]. Three external data memory bus access operating modes are available:

- Multiplexed Data/Address[7:0] external data memory access
- Non Multiplexed external data memory access
- Data Bus Chip Select (DBCS) mode

The external memory address range 0000h to 3FFFh provides access to a block of 4KB of SRAM memory on the device. The XMEMXTRL register located at address D9h controls the operating mode of the external data memory bus.

TABLE 27: XMEM CONTROL REGISTER - XMEMCTRL SFR D9H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	EXTBUSCFG	External Memory Bus Configuration 0 = LSB of Address/Data are Multiplexed 1 = LSB of Address/Data are not Multiplexed
6	EXTBUSCS	Ext Memory CS Function 0 = Full Address Bit Dedicated to Addressing 1 = A12: A15 Becomes CS Lines
5	-	Not used
4	-	Not Used
3:0	STRETCH[3:0]	Number of Stretch Cycles from 0 to 15

The EXTBUSCFG bit of the XMEMCTRL register defines the hardware configuration used for external data memory access.

When the EXTBUSCFG bit is cleared, the external data memory bus will be accessed like a standard 8051, where the lower eight address and data bit are time-multiplexed. In that mode, the ALE signal serve to indicate the output of bus address, especially A[7:0] which are multiplexed with D[7:0]

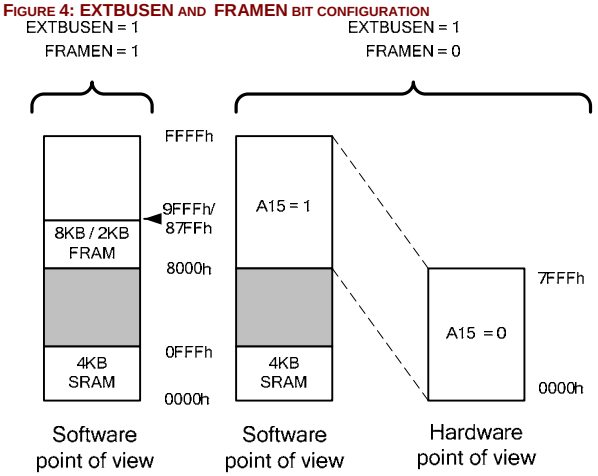
Setting the EXTBUSCFG bit to 1 will activate the access operation of the non-multiplexed external data memory bus. This new mode is intended to avoid the use of an external octal D flip-flop device to hold the LSB of the target address.

The FRAMEN bit controls the activation of the F-RAM memory module. When the FRAMEN bit is set to 1, the F-RAM module will monopolize any XDATA access (read/write) operations targeting an address >= 8000h. In order to access the external data memory bus, the EXTBUSEN bit of the DEVMEMCFG SFR must be set to 1 and the FRAMEN bit must be set to 0.

Any XDATA memory access with a target address < 8000h will have no impact on the I/O associated with the external data memory bus.

From a device connected to the VRS51L30xx's external memory bus, the address range is seen as 0000 to 7FFFh, as A15 address line is not pinned out.

FIGURE 4: EXTBUSEN AND FRAMEN BIT CONFIGURATION



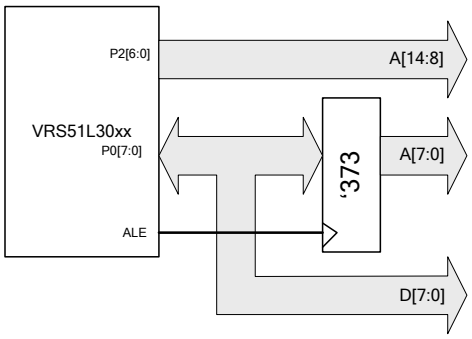
Multiplexed External Data Memory Access

The multiplexed external data memory access mode on the VRS51L30xx devices is similar to that of a standard 8051: Address bits A7 to A0 and data bits D7 to D0 are time-multiplexed on Port 0 with the ALE pin synchronizing the output of A[7:0]. Port 2 controls address bits A14 to A8.

Contrary to standard 8051 devices, the A15 line is not pinned out on the VRS51L30xx devices. Pin P2.7 I/O, which corresponds to line A15 line on a standard 8051, will remain low.

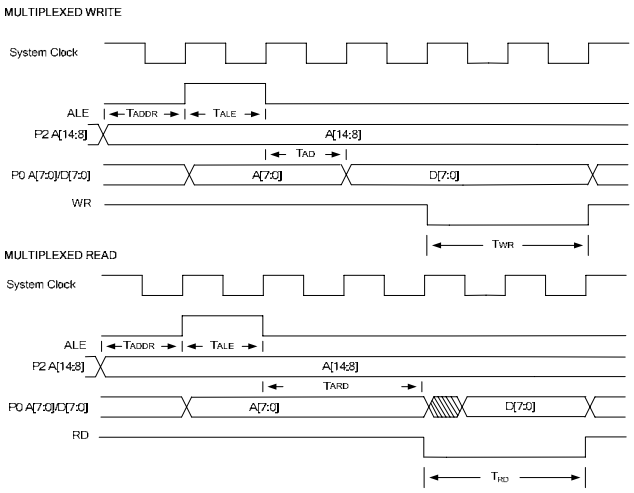
In multiplexed addressing mode, external glue logic is required to multiplex lower addresses and data. Typically, a 74x373 or 74x573 can be used for this purpose. The ALE-CM0 pin serves to latch the lower 8 bits of the address.

FIGURE 5: MULTIPLEXED EXTERNAL DATA MEMORY ACCESS REPRESENTATION



The diagram below shows the timing of the external data memory bus when configured in multiplexed mode.

FIGURE 6: MULTIPLEXED EXTERNAL DATA MEMORY ACCESS



Assuming that the system clock operates at 40MHz, typical cycles for the external memory bus are as follows:

Parameter	Description	Typical value for system clock = 40MHz
T_{ALE}	ALE High time	25ns
T_{ADDR}	Time between ADDR[14:8] and ALE rising edge	10ns
T_{AD}	Time A[7:0] is stable after ALE falling edge	20ns
T_{ARD}	Time between ALE falling edge and RD falling edge	50ns
T_{WR}	WR signal Low time	50ns
T_{RD}	RD signal Low time	50ns

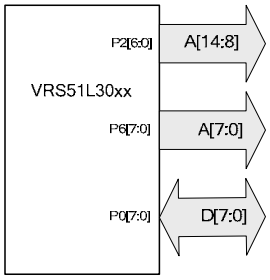
Non-Multiplexed External Data Memory Access

The VRS51L3xxx external address and data memory bus can operate in non-multiplexed mode. This mode is activated by setting the EXTBUSCFG bit of the XMEMCTRL register to 1.

In this case:

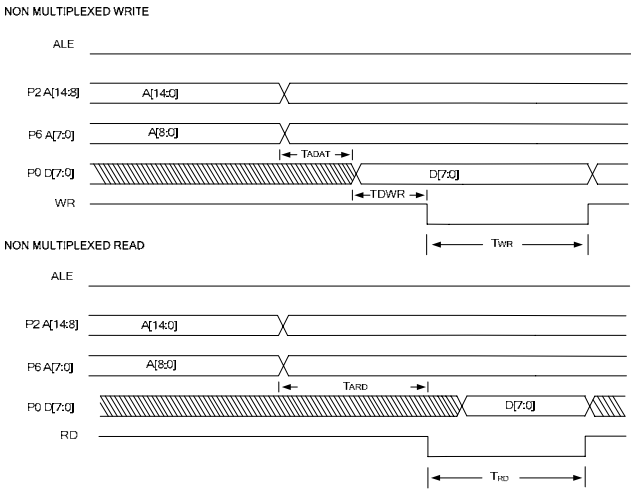
- D7:D0 will be mapped into Port 0
- A7:A0 will be mapped into Port 6
- A15:A8 will be mapped into Port 2

FIGURE 7: NON-MULTIPLEXED EXTERNAL DATA MEMORY ACCESS REPRESENTATION



The diagram below shows the timing of the external data memory bus when configured in non-multiplexed mode.

FIGURE 8: NON-MULTIPLEXED EXTERNAL DATA MEMORY ACCESS



Assuming that the system clock operates at 40MHz, typical cycles for the external memory bus are as follows:

Parameter	Description	Typical value for System clock = 40MHz
T_{ADAT}	Time between A[14:0] stable and data stable	15ns
T_{ARD}	Time between D[7:0] stable and WR falling edge	20ns
T_{WR}	WR signal Low time	50ns
T_{ARD}	Time between A[14:0] stable and RD falling edge	50ns
T_{RD}	RD signal Low time	50ns

4.10 External Data Bus Access for VRS51L31xx devices (QFP-44)

The VRS51L31xx (QFP-44) family members also provide external memory bus access on the upper 32KB block of the 64KB external memory [8000h to FFFFh]. Two external data memory bus access operating modes are available:

- Multiplexed Data/Address[7:0] external data memory access
- Data Bus Chip Select (DBCS) mode

The external memory address range 0000h to 3FFFh provides access to a block of 4KB of SRAM memory on the device. The XMEMXTRL register located at address D9h controls the operating mode of the external data memory bus.

TABLE 28: XMEM CONTROL REGISTER - XMEMCTRL SFR D9H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	EXTBUSCFG	External Memory Bus Configuration 0 = LSB of Address/Data are Multiplexed 1 = LSB of Address/Data are not Multiplexed
6	EXTBUSCS	Ext Memory CS Function 0 = Full Address Bit Dedicated to Addressing 1 = A12: A15 Becomes CS Lines
5	-	Not used
4	-	Not Used
3:0	STRETCH[3:0]	Number of Stretch Cycles from 0 to 15

The EXTBUSCFG bit of the XMEMCTRL register defines the hardware configuration used for external data memory access.

When the EXTBUSCFG bit is cleared, the external data memory bus will be accessed like a standard 8051, where the lower eight address and data bit are time-multiplexed. In that mode, the ALE signal serve to indicate the output of bus address, especially A[7:0] which are multiplexed with D[7:0]

Setting the EXTBUSCFG bit to 1 will activate the access operation of the non-multiplexed external data memory bus. This new mode is intended to avoid the use of an external octal D flip-flop device to hold the LSB of the target address.

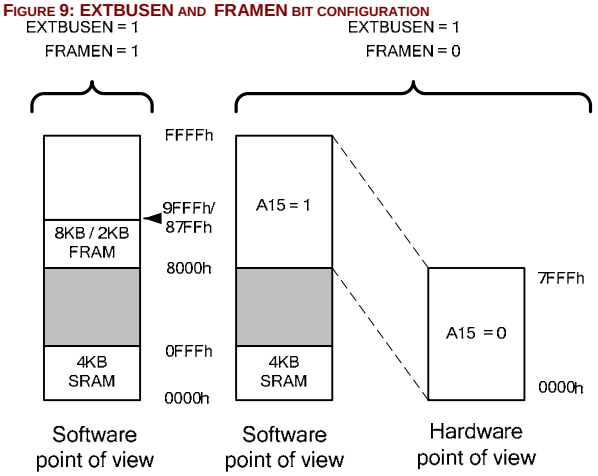
The FRAMEN bit controls the activation of the F-RAM memory module. When the FRAMEN bit is set to 1, the F-RAM module will monopolize any XDATA access (read/write) operations targeting an address >= 8000h.

In order to access the external data memory bus, the EXTBUSEN bit of the DEVMEMCFG SFR must be set to 1 and the FRAMEN bit must be set to 0.

Any XDATA memory access with a target address < 8000h will have no impact on the I/O associated with the external data memory bus.

From a device connected to the VRS51L31xx's external memory bus, the address range is seen as 0000 to 7FFFh, as A15 address line is not pinned out.

FIGURE 9: EXTBUSEN AND FRAMEN BIT CONFIGURATION



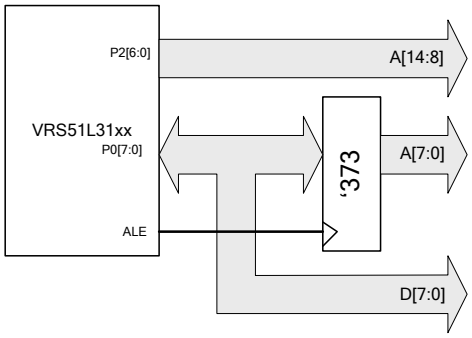
Multiplexed External Data Memory Access

The multiplexed external data memory access mode on the VRS51L31xx is similar to that of a standard 8051: Address bits A7 to A0 and data bits D7 to D0 are time-multiplexed on Port 0 with the ALE pin synchronizing the output of A[7:0]. Port 2 controls address bits A14 to A8.

Contrary to standard 8051 devices, the A15 line is not pinned out on the VRS51L31xx devices. Pin P2.7 I/O, which corresponds to line A15 line on a standard 8051, will remain low.

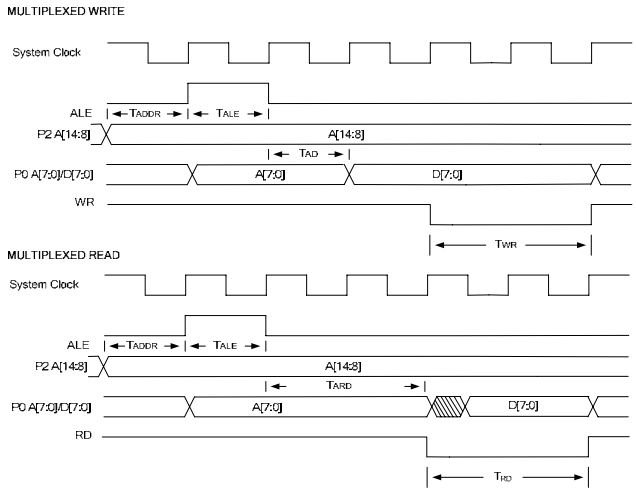
In multiplexed addressing mode, external glue logic is required to multiplex lower addresses and data. Typically, a 74x373 or 74x573 can be used for this purpose. The ALE-CM0 pin serves to latch the lower 8 bits of the address.

FIGURE 10: MULTIPLEXED EXTERNAL DATA MEMORY ACCESS REPRESENTATION



The diagram below shows the timing of the external data memory bus when configured in multiplexed mode.

FIGURE 11: MULTIPLEXED EXTERNAL DATA MEMORY ACCESS



Assuming that the system clock operates at 40MHz, typical cycles for the external memory bus are as follows:

Parameter	Description	Typical value for system clock = 40MHz
T_{ALE}	ALE High time	25ns
T_{ADDR}	Time between ADDR[14:8] and ALE rising edge	10ns
T_{AD}	Time A[7:0] is stable after ALE falling edge	20ns
T_{ARD}	Time between ALE falling edge and RD falling edge	50ns
T_{WR}	WR signal Low time	50ns
T_{RD}	RD signal Low time	50ns

4.11 External Data Bus CS (DBCS) Control Lines

In some applications, only a few external memory addresses are required to perform high speed data transfers between the microcontroller and peripherals, such as parallel access data converters. In this case, only a few address locations on the external data memory bus have to be accessed. The VRS51L3xxx provides a feature that can simplify interfacing to these peripherals .

Setting the EXTBUSCS bit of the DEVMEMCFG register to 1 will activate the external bus control lines

External bus CS mode can also work in standard 8051 external data memory buss access mode where Lower 8 bits of Address A[7:0] and Data D[7:0] are multiplexed.

When both the EXTBUSCS bit of the XMEMCTRL register and the EXTBUSGEN bit of the DEVMEMCFG register are set to 1, the P2[7:4] pins of the VRS51L3xxx will act as an active high chip select pin named DBCSB[3:0].

The value of bits 13 and 12 of the target address will define the active DBCS line. A11:A0 carries the rest of the address bits. This is represented at the register level as follows:

A15	A14	A13	A12		…		A0
X	X	CS1	CS0		…		

As such, when the CS bus control mode is activated, the upper 32KB of the external data memory bus is seen as two overlapping blocks of 16KB.

TABLE 29: EXTERNAL MEMORY BUS CS CONTROL MODE

Address range	Active as CSBx pin
0000h- 7FFFh	None (4KB SRAM from 0000h to 0FFFh)
8000h-8FFFh	DBCS0 (P2.4)
9000h-9FFFh	DBCS1 (P2.5)
A000h-AFFFh	DBCS2 (P2.6)
B000h-BFFFh	DBCS3 (P2.7)
C000h-CFFFh	DBCS0 (P2.4) Overlap
D000h-DFFFh	DBCS1 (P2.5) Overlap
E000h-EFFFh	DBCS2 (P2.6) Overlap
F000h-FFFFh	DBCS3 (P2.7) Overlap

Setting the EXTBUSCFG bit of the XMEMCTRL register has no impact on the operation in external bus CS mode.

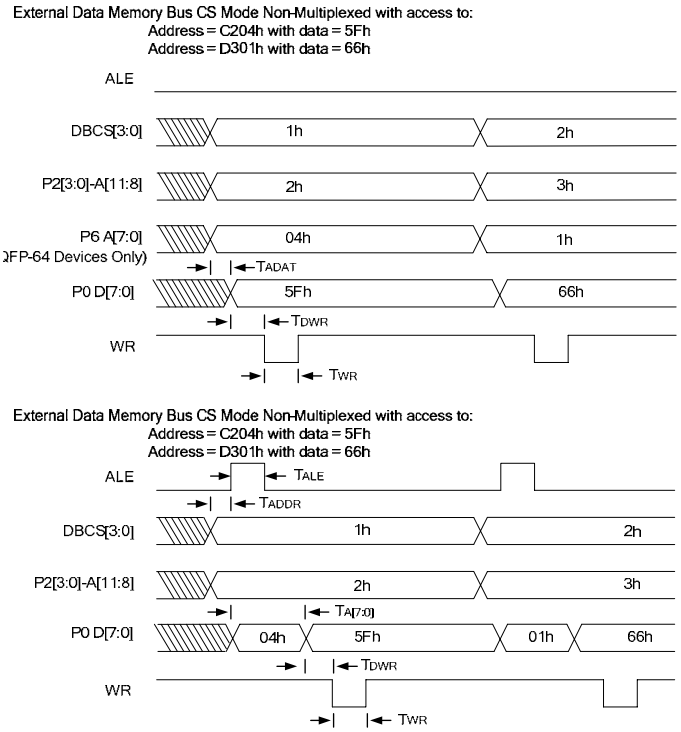
The table on next page summarizes the mapping and the activity of the I/O pins associated with the external data memory bus when configured in external bus CS mode.

TABLE 30: PIN MAPPING IN EXTERNAL MEMORY BUS MODE

Signal in Ext. Bus CS Mode	Mapping / Activity when EXTBUSCFG = 0 Multiplexed	Mapping / Activity when EXTBUSCFG = 1 Non-Multiplexed
ALE	Active	Inactive
DBCS[3:0]	P2[7:4]	P2[7:4]
Address[11:8]	P2[3:0]	P2[3:0]
Address[7:0]	P0[7:0]	P6[7:0] (VRS51L30xx, QFP-64 devices only
Data[7:0]	P0[7:0]	P0[7:0]
P6	Available as regular I/O	Address[7:0] (VRS51L30xx, QFP-64 devices only
RD	Active	Active
WR	Active	Active

Us of the external memory bus in chip select mode with the EXTBUSCFG bit set to 0 can be useful in applications where P6 may be required to function as an I/O port. Signal timing associated with the external bus mode is represented below:

FIGURE 12: EXTERNAL DATA MEMORY BUS IN CHIP SELECT MODE (NON-MULTIPLEXED)



Assuming that the system clock operates at 40MHz, typical cycles for the external memory bus are as follows:

TABLE 31: TIMING ASSOCIATED WITH EXTERNAL DATA MEMORY BUS ACCESS

Parameter	Description	Typical value for System clock = 40MHz
T_{ADAT}	Time delay between address stable and Data[7:0] in non-multiplexed mode	15ns
T_{DWR1}	Time between Data[7:0] stable and WR falling edge in non-multiplexed mode	20ns
T_{ALE}	ALE High time	25ns
T_{ADDR}	Time between ADDR[14:8] and ALE rising edge	15ns
$T_{A[7:0]}$	Time A[7:0] is stable after ALE falling edge	50ns
T_{DWR2}	Time between Data[7:0] stable and WR falling edge.	20ns
T_{WR}	WR signal Low time	50ns

4.12 Page Addressing of the External SRAM using the MPAGE Register

The MPAGE register provides access to the entire external memory using indirect addressing through registers R0 and R1. The MPAGE register can be used to hold the upper 8 bit of the target address when using the MOVX @Ri instruction which by definition is limited to a 256 Bytes range.

TABLE 32: MEMORY PAGE REGISTER - MPAGE SFR F1H

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
MPAGE[7:0] = Upper Address Byte							

4.13 Slowing down the External Data Memory Bus Access

The STRETCH[3:0] bit of the XMEMCTRL register allows the user to add cycles to the RD and WR signals. The ALE signal is not affected by the STRETCH[3:0] configuration.

For application requiring lower overall bus speed, we suggest lowering the system clock speed using the DEVCLKCFG[3:0] register.

The table below shows the impact of STRETCH[3:0] and the DEVCLKCFG[3:0] on the external data memory bus access cycle time, assuming the VRS51L3xxx operates from the 40MHz internal oscillator.

TABLE 33: IMPACT OF DEVCLKCFG & STRETCH[3:0] ON EXT. DATA MEMORY BUS TIMINGS

DEVCLKCFG[3:0]	STRETCH[3:0]	RD/WR	ALE
0	0	50ns	25ns
0	1	75ns	25ns
0	2	100ns	25ns
0	8	250ns	25ns
4	0	800ns	400ns
4	8	4uS	400ns

The STRETCH[3:0] configuration does not affect access to the 4KB SRAM and the 8KB F-RAM.

5 Device Configuration

5.1 Clock Configuration Register

The VRS51L3xxx clock system is highly configurable. All VRS51L3xxx family members feature an internal 40MHz oscillator, eliminating the need for an external oscillator or crystal. However, an external standard parallel AT or BT cut crystal can be used (frequency range of 4MHz to 40MHz).

Two SFR registers control the configuration of the clock source and the division ratio applied to the system clock source. The DEVCLKCFG1 register selects either the internal oscillator or the external crystal oscillator as the system clock source. When the OSCSELECT bit is cleared, the devices system clock comes from the external crystal oscillator (please see the next section).

TABLE 34:DEVICE CLOCK CONFIGURATION REGISTER 1 - DEVCLKCFG1 SFR F2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	1	0	0	0	0	0

Bit	Mnemonic	Description
7	SOFTRESET	Soft reset control bit
6	OSCSELECT	Oscillator Select 0 = External oscillator is selected 1 = Internal oscillator is selected
5	CLKDIVEN	Internal oscillator output clock divisor enable bit 0 = Disable Clock Divisor 1 = Enable Clock Division
4	FULLSPDINT	Full Speed Interrupt Mode 0 =Processor will run with selected clock division during interrupts 1 = Processor will run at full speed during interrupts
3:0	CLKDIV[3:0]	CLKDIV Value/Clock Division 0 = /1 1 = /2 2 = /4 3 = /8 4 = /16 5 = /32 6 = /64 7 = /128 8 = /256 9 = /512 A = /1024 B = /2048 C = /4096 D = /8192 E = /16384 F= /32768

Soft Reset Operation

A software reset can be performed on the VRS51L3xxx. This is executed via two consecutive instruction: The first instruction is to clear the SOFTRESET bit and the second is to set SOFTRESET bit to 1:

Examples of soft Reset in ASM:

```
ANL    DEVCLKCFG,#7Fh;
ORL    DEVCLKCFG,#80h;
```

In C :

```
DEVCLKCFG &= 0x7F;
DEVCLKCFG |= 0x80;
```

When using a C compiler verify the compiler does convert the abovementioned instructions on two consecutive instructions performing a write into the DEVCLKCFG register.

The DEVCLKCFG2 register activates the on-chip oscillator and the crystal oscillator. Both oscillators can be activated independently, however, as previously mentioned, only one can be used as the VRS51L3xxx system clock source.

TABLE 35:DEVICE CLOCK CONFIGURATION REGISTER 2 - DEVCLKCFG2 SFR F3H

7	6	5	4	3	2	1	0
R/W	R/W	R	R	R/W	R/W	R/W	R
0	1	0	0	1	0	0	1

Bit	Mnemonic	Description
7	CYOSCEN	Crystal Oscillator Enable 0 = Crystal oscillator is disabled (default) 1 = Crystal oscillator is enabled
6	INTOSCEN	Internal Oscillator Enable 0 = Internal oscillator is disabled 1 = Internal oscillator is enabled (default)
5	Reserved	
4	Reserved	
3:2	CYRANGE[1:0]	Crystal Oscillator Range 00 = 25MHz – 40MHz 01 = 4MHz to 25MHz 10 = 32kHz to 100KHz 11 = 32kHz to 100KHz
1	Reserved	
0	Reserved	Always read as 1

The crystal oscillator is activated by setting the CYOSCEN bit of the DEVCLKCFG2 register to 1 and selecting the CYRANGE value according to the frequency of the crystal used. The CYRANGE parameter controls the drive of the crystal oscillator circuit. The internal oscillator is activated by setting the INTOSEN bit to 1.

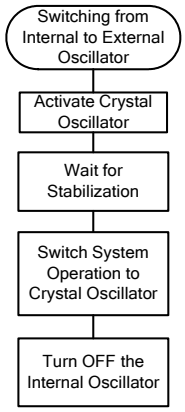
Before switching from one oscillator source to another, it is important to make sure that both oscillators are active and stable at the moment the transition is made. The minimum period required for the crystal oscillator to stabilize depends on the type of crystal and the frequency used. In general, it is recommended to wait at least 1ms for the crystal oscillator to stabilize before switching to it.

The stabilization time of the internal oscillator is much shorter than that of the crystal oscillator. Whenever the internal oscillator is reactivated, wait >2uS before switching the system clock back to the internal oscillator.

5.2 Switching from the Internal to the External Oscillator

The following steps represent the recommended procedure for switching from the internal oscillator to the crystal oscillator:

FIGURE 13: SWITCHING FROM INTERNAL OSCILLATOR TO EXTERNAL OSCILLATOR



Code Example: Internal to External

```
DEVCLKCFG2 = 0xC4;          //Enable the external oscillator:
                            //Range 4 to 25MHz and KEEP
                            //the internal oscillator active
delay(1);                   //Stabilization time
 DEVCLKCFG1 &= 0xBF;        //Select external oscillator
 DEVCLKCFG2 &= 0xBF;        //Deactivate the internal oscillato
```

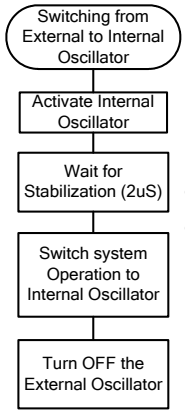
It is important to allow the crystal oscillator to stabilize before using it as the system clock. An instable oscillator may result in an operating frequency error or device volatility.

5.3 Switching from the External Oscillator Back to the Internal Oscillator

It is possible to switch system clock source to the internal oscillator while the device is running from the external oscillator. Note that before switching the internal oscillator, it must be active.

The following the sequence below is recommended in order to switch from the crystal oscillator back to the internal oscillator:

FIGURE 14: SWITCHING FROM EXTERNAL OSCILLATOR TO INTERNAL OSCILLATOR



Code Example: External to Internal

```
DEVCLKCFG2 |= 0x40;          //Enable the internal oscillator:
                             //and KEEP the external oscillato
                            //active
delay();                     //Stabilization time (>2uS)
 DEVCLKCFG1 |= 0x40;         //Select internal oscillator
delay();                     //Stabilization time (facultative)
 DEVCLKCFG2 &= 0x7F;         //Deactivate the external oscillato
```

5.4 System Clock Prescaler

Between the internal and the external oscillator modules and the main system clock tree, the VRS51L3xxx devices include a clock prescaler module enabling a dynamic division adjustment of the system clock frequency from FOSC /1 to FOSC/32768. This feature can be useful for saving power in battery-operated applications, in which the device clock speed can be adjusted to suit the processing power requirements.

After a reset, the processor will boot up from the internal oscillator and the selected operating speed will be set to 20MHz i.e.. CLKDIVEN is set to 1 and the CLKDIV value is 1 (CLK = Fosc/2). Clearing the CLKDIVEN bit will deactivate the main clock prescaler.

5.5 Interrupt Processing Speed Configuration

The VRS51L30xxx devices include a feature that allows interrupts to be processed at full speed, while the main program executes at a lower speed, as defined by the FULLSPDINT value when the CLKDIVEN bit is set to 1.This mode of operation can be useful for applications where high processing power is required for short periods of time. Significant power saving can be achieved by dynamically adjusting the system clock frequency according to the processing power required.

Code Example:

Switching from Internal to External Oscillator Example Program

```
//-------------------------------------------------------------------------------------------------//
//VRS51L3074_Int_to_ext_to_Int_osc_switching_test2-SDCC.c
//-------------------------------------------------------------------------------------------------//
//
// DESCRIPTION:
//              Test switching from internal osc to the external oscillator
//               then back to the internal oscillator...forever
//               1) The program start from the internal oscillator with
//                  duty = 50 / 50 for 100 cycles
//               2) Then it switch to external oscillator with a
//                  duty of 50/20for 100 cycles
//               3) It then switch to internal oscillator
//               4) then it execute 100 cycles with a
//                  duty of 20/50 for 100 cycles
//               5) Return to step 2
//-------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>
// --- function prototypes

void delay(unsigned int);

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

void main (void)
    {
    int cptr ;
    PERIPHEN1 = 0x01;               //Enable Timer 0
    PERIPHEN2 = 0x08;               //Enable IOPORT
    P2PINCFG = 0x00;                //Config port 2 as output (for Tests)

    for(cptr =0; cptr < 100; cptr++)  //toggle P2 100 times
        {
        P2 = 0xFF;
        delay(50);
        P2 = 0x00;
        delay(50);
        };

    do{
    //-- Enable the external oscillator

       DEVCLKCFG2 = 0xC0;          //Enable the external oscillator,
                                   //Keep external osc active
                                   //Crystal range = 1 to 20MHz
       delay(10);                  //Stabilization Time
       DEVCLKCFG1 = 0x20;          //Select External oscillator
       delay(1);                   //Stabilization Time
       DEVCLKCFG2 = 0x83;          //Keep the external oscillator,
                                   //Disable internal osc active

    for(cptr =0; cptr < 100; cptr++)  //toggle P2 100 times
        {
        P2 = 0xFF;
        delay(50);
        P2 = 0x00;
        delay(20);
        };

     //-- Return to the internal oscillator
      DEVCLKCFG2 = 0xC0;           //Keep the external oscillator enabled
                                   //Activate  the internal osc
                                   //Crystal range = 1 to 20MHz

       delay(100);                 // Stabilization Time (way too much)

       DEVCLKCFG1 = 0x60;          //Select Internal oscillator
       delay(1);                   // Stabilization Time

       DEVCLKCFG2 = 0x40;         //Disable the external oscillator,
                                   //Keep internal osc active

    for(cptr =0; cptr < 100; cptr++)  //toggle P2 100 times
        {
        P2 = 0xFF;
        delay(20);
        P2 = 0x00;
        delay(50);
        };


   }while(1);

}// End of mai006E
```

```
//-------------------------------------------------------------------------------------------------//
//---------   INDIVIDUALS FUNCTIONS      -------------//
//-------------------------------------------------------------------------------------------------//

//;-----------------------------------------------------------------------
//;- DELAY1MSTO : 1MS DELAY USING TIMER0
//; CALIBRATED FOR 40MHZ
//;-----------------------------------------------------------------------
void delay(unsigned int dlais){
idata unsigned char x=0;
idata unsigned int dlaisloop;

x = PERIPHEN1;                  //LOAD PERIPHEN1 REG
x |= 0x01;                       //ENABLE TIMER 0
PERIPHEN1 = x;
dlaisloop = dlais;
    while ( dlaisloop > 0)
       {
       TH0 = 0x63;                //TIMER0 RELOAD VALUE FOR 1MS AT 40MHZ
       TL0 = 0xC0;

       T0T1CLKCFG = 0x00;              //NO PRESCALER FOR TIMER 0 CLOCK
       T0CON = 0x04;                  //START TIMER 0,  COUNT UP
        do{
            x=T0CON;
            x= x & 0x80;
            }while(x==0);
        T0CON = 0x00;                  //Stop Timer 0
        dlaisloop = dlaisloop-1;

        }//end of while dlais...

       x = PERIPHEN1;                   //LOAD PERIPHEN1 REG
       x = x & 0xFE;                   //DISABLEBLE TIMER 0
       PERIPHEN1 = x;
    }//End of function delais
```

5.6 Processor Mode Control Register

The VRS51L3xxx devices provide two power saving modes: Idle and power-down, which are controlled by the PDOWN and IDLE bits of the PCON register at address 87h.

TABLE 36:POWER CONTRO L REGISTER - PCON SFR 87H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	1	0	0	0	0	0

Bit	Mnemonic	Description
7	OSCSTOP	Oscillator Stop Control When this bit is set to 1, the VRS51L3xxx oscillator stops. A reset pulse or a power-on reset is required to restart the device
6	INTMODEN	Interrupt Module Enable 0 = Interrupt module is disabled 1 = Interrupt module is enabled (default)
5	DEVCFGEN	Device Configuration Module Enable 0 = Device configuration module is disabled 1 = Device configuration module is enabled
4	SFRINDADR	SFR Indirect Addressing Enable 0 = NOP instruction A5h behaves normally 1 = NOP instruction A5h acts as a SFR indirect addressing instruction
3	GF1	General Purpose Flag
2	GF0	General Purpose Flag
1	PDOWN	Power-Down Mode Enable When this bit is set to 1, the processor goes into power-down mode. A reset is required to exit power-down mode
0	IDLE	Idle Mode Enable When this bit is set to 1, the processor goes into power-idle mode. A reset or an interrupt is required to exit idle mode

Oscillator Stop Mode

The oscillator stop mode goes one step further than the PDOWN mode. When the OSCSTOP bit is set, all the oscillators are stopped, achieving maximum power saving, while maintaining the I/Os in their current state. Note that in this mode, the watchdog timer will stop functioning.

In order to stop the oscillator of the VRS51L3xxx, clear the OSCSTOP bit of the PCON register and then immediately set it to 1, as shown below:

```
PCON &= 0x7F
PCON |= 0x80
```

SFR Indirect Addressing Capability

The SFR registers on the VRS51L3xxx can be accessed via indirect addressing. This is accomplished by setting the SFRINDADR bit of the PCON register.

When SFRINDADR is set, the A5h instruction functions as an SFR indirect addressing instruction (the default at reset is the NOP instruction).

PDOWN and IDLE Power Saving Mode

In idle mode, the processor clock is stopped, however the peripherals remain active. The contents of the SRAM, the state of the I/Os and the SFR registers are maintained, as are the timer, external interrupt and UART operations. Idle mode is useful for applications in which stopping the processor to save power is required. The processor will be activated when an external event, triggering an interrupt, occurs.

In power-down mode, the VRS51L3xxx oscillator is stopped. While the clock to all the peripherals is deactivated, the contents of the SRAM and the SFR registers is maintained. The only way to exit power-down mode is via a hardware reset.

In power-down and idle modes the watchdog timer continues to function.

5.7 Peripherals Enable Register

The VRS51L3xxx peripherals can be individually activated. The PERIPHEN1 and PERIPHEN2 registers are used for this purpose.

With the exception of the I/O ports, all peripherals and communication interfaces are in the disable state upon reset. When a given peripheral is inactive, read and write operations to its SFR registers will have no effect. To activate a given peripheral, the corresponding enable bit in the PERIPHENx registers must be set to 1.

The PERIPHEN1 register controls the activation of the:

- SPI Interface
- I²C Interface
- Two UARTs
- Timers

TABLE 37: PERIPHERAL ENABLE REGISTER 1 - PERIPHEN1 SFR F4H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	SPICSEN	Enable SPI CS Line 0 = SPI CS lines are disabled (accessible as I/O) 1 = SPI CS lines are enabled and reserved by SPI interface
6	SPIEN	SPI Interface Enable 0 = SPI interface is disabled 1 = SPI interface is enabled
5	I2CEN	I²C Interface Enable 0 = I²C interface is disabled 1 = I²C interface is enabled
4	U1EN	UART1 Interface Enable 0 = UART1 interface is disabled 1 = UART1 interface is enabled
3	U0EN	UART0 Interface Enable 0 = UART0 interface is disabled 1 = UART0 interface is enabled
2	T2EN	Timer2 Enable 0 = Timer 2 interface is disabled 1 = Timer 2 Interface is enabled
1	T1EN	Timer1 Enable 0 = Timer 1 interface is disabled 1 = Timer 1 interface is enabled
0	T0EN	Timer0 Enable 0 = Timer 0 interface is disabled 1 = Timer 0 interface is enabled

The PERIPHEN2 register controls the activation of the:

- Pulse Width Counter Modules
- Arithmetic Unit
- I/O Ports
- Watchdog Timer
- FPI Interface

It also activates the XRAM into code mode, in which the processor starts executing code from the 4KB block of externally mapped SRAM memory.

TABLE 38: PERIPHERA2 ENABLE REGISTER 2 - PERIPHEN2 SFR F5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	1	0	0	0

Bit	Mnemonic	Description
7	PWC1EN	Pulse Width Counter 1 Enable 0 = PWC1 is off 1 = PWC1 is on
6	PWC0EN	Pulse Width Counter 0 Enable 0 = PWC0 is off 1 = PWC0 is on
5	AUEN	Arithmetic Unit Enable 0 = Arithmetic unit is off 1 = Arithmetic unit is on
4	XRAM2CODE	When set to 1, the 4KB block of SRAM is mapped into the program code area from 0000h to 0FFFh. XRAM-based variable are not permitted when the processor is running from the XRAM. The XRAM2CODE bit must be set and cleared only when the program counter is outside the abovementioned address range.
3	IOPORTEN	I/O Port Enable 0 = I/O Ports are deactivated 1 = I/O Ports are activated
2	WDTEN	Watchdog Timer Module Enable 0 = WDT is OFF 1 = WDT is ON
1	PWRSFREN	Pulse Width Modulators SFR Enable 0 = SFR associated with PWMs are deactivated 1 = SFR associated with PWMs are activated
0	FPIEN	FPI Interface Enable 0 = FPI interface is disabled 1 = FPI interface is enabled

5.8 Peripheral I/O Mapping and Priority

The pin locations of the following peripherals can be remapped to alternate pin positions:

- Timer 2 Output
- I²C
- UART0
- UART1
- PWMs

This feature has been included to provide access to all peripherals. The following table lists the peripherals whose I/O positions are configurable:

Table 39: DEVIOMAP register - SFR E1h

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description		
7	Reserved			
6	PWMALTMAP	PWM Alternate mapping configuration 0 = PWM output on P2[7:0] 1 = PWM output on P5[7:0]		
5	I2CALTMAP	I2C Alternate mapping configuration 0 = I2C SCL, SDA on P3.4, P3.5 1 = I2C SCL, SDA on P1.6, P1.7		
4	U1ALTMAP	UART1 Alternate mapping configuration 0 = UART1 RXD1, TXD1 on P1.2, P1.3 1 = UART1 RXD1, TXD1 on pin 41, pin 40 of QFP-64 package*		
3	U0ALTMAP	UART0 Alternate mapping configuration 0 = UART0 RXD0, TXD0 on P3.0, P3.1 1 = UART0 RXD0, TXD0 on P2.4, P2.3		
2	T2ALTMAP	Timer 2 Alternate mapping configuration		
		T2 pin	**T2ALTMAP = 0**	**T2ALTMAP = 1**
		T2	P4.4	P1.2
		T2IN	P1.0	P6.1*
		T2EX	P1.1	P6.0*
1	T1ALTMAP	Timer 1 Output Alternate mapping configuration		
0	T0ALTMAP	Timer 0 Output Alternate mapping configuration		

*Only on VRS51L30xx, QFP-64 devices

When the SPI interface is enabled, the SPI CS0 line is reserved for the SPI interface, independent of the state of the SPICSEN bit.

UART1 has priority over the SPICSEN bit of the PERIPHEN1 register. As such, even if the SPI CS1, CS2 and CS3 lines are activated by setting the SPICSEN bit to 1, when UART1 is used, it will override CS2 and CS3.
Additionally, when activated, the SPI interface, has priority over the Timer 2 input, even if Timer 2 is enabled.

6 Input/Output Ports

The VRS51L30xx devices includes 56 I/O pins grouped into seven ports. The VRS51L31xx includes 40 I/O pins grouped into 5 ports

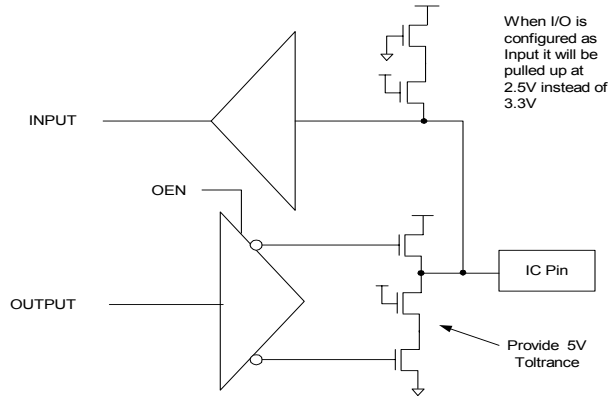
To offer the maximum number of I/O pins, the pins typically reserved for external program memory access can also be used as I/O interface pins. In addition, when the internal oscillator is enabled, the crystal oscillator pins can be used as regular I/Os.

All I/Os are 5V-tolerant except for P4.6 and P4.7, which can endure a maximum input voltage of VDD+0.5V.

6.1 Structure of the I/O Ports

All I/O ports on have the same structure. Their main difference resides in the drive capability of the I/O ports, as shown in the following diagram:

FIGURE 15: GENERAL STRUCTURE OF THE I/O PINS



When the I/O ports are configured as inputs, the pin is pulled high to a voltage of about 2.50V, instead of the device voltage, which is 3.3V. An external pull-up resistor can be added to pull the I/O pin up to 3.3 volts or to 5 volts.

6.2 I/O Ports Direction Configuration Registers

Each I/O port on the VRS51L3xxx has dedicated SFR registers for read/write operations and for I/O pin direction. The pin direction configuration registers allow the user to configure the direction of each individual I/O pin. Writing a 1 to these register bit positions configures the corresponding I/O port as an input. To configure an I/O pin as an output, the corresponding bit in the pin direction configuration register must be cleared.

Because the pin direction configuration registers are not located at addresses that are multiples of x0h or x8h, they are not bit-addressable. When a peripheral is activated, it takes control of the I/O pins and the I/O pin direction is configured automatically.

The user can monitor the activity of any peripheral module input pin current state by configuring the corresponding I/O pin as an input and reading the port pin value.

TABLE 40:PORT 0 PIN DIRECTION CONFIGURATION REGISTER - P0PINCFG -SFR F9H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P07IN1OUT0	When: 1 = I/O pin acts as a input (reset value) 0 = I/O pin acts as a output
6	P06IN1OUT0	Same as bit 7
5	P05IN1OUT0	Same as bit 7
4	P04IN1OUT0	Same as bit 7
3	P03IN1OUT0	Same as bit 7
2	P02IN1OUT0	Same as bit 7
1	P01IN1OUT0	Same as bit 7
0	P00IN1OUT0	Same as bit 7

When the external data memory bus access is activated, Port 0 functions as D7:D0 and/or address A7:A0.

TABLE 41:PORT 1 PIN DIRECTION CONFIGURATION REGISTER - P1PINCFG -SFR FAH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P17IN1OUT0	1 = I/O pin act as a input (reset value) 0 = I/O pin act as a output
6	P16IN1OUT0	Same as bit 7
5	P15IN1OUT0	Same as bit 7
4	P14IN1OUT0	Same as bit 7
3	P13IN1OUT0	Same as bit 7
2	P12IN1OUT0	Same as bit 7
1	P11IN1OUT0	Same as bit 7
0	P10IN1OUT0	Same as bit 7

TABLE 42:PORT 2 PIN DIRECTION CONFIGURATION REGISTER - P2PINCFG -SFR FBH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P27IN1OUT0	When: 1 = I/O pin acts as a input (reset value) 0 = I/O pin act as a output
6	P26IN1OUT0	Same as bit 7
5	P25IN1OUT0	Same as bit 7
4	P24IN1OUT0	Same as bit 7
3	P23IN1OUT0	Same as bit 7
2	P22IN1OUT0	Same as bit 7
1	P21IN1OUT0	Same as bit 7
0	P20IN1OUT0	Same as bit 7

When the external data memory bus is activated, except when in external bus CS mode, Port 2 functions as address bus bits A15:A8.

TABLE 43:PORT 3 PIN DIRECTION CONFIGURATION REGISTER - P3PINCFG -SFR FCH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P37IN1OUT0	When: 1 = I/O pin act as a input (reset value) 0 = I/O pin act as a output
6	P36IN1OUT0	Same as bit 7
5	P35IN1OUT0	Same as bit 7
4	P34IN1OUT0	Same as bit 7
3	P33IN1OUT0	Same as bit 7
2	P32IN1OUT0	Same as bit 7
1	P31IN1OUT0	Same as bit 7
0	P30IN1OUT0	Same as bit 7

When the external data memory bus is activated, P3.6 and P3.7 function as WR and RD.

TABLE 44:PORT 4 PIN DIRECTION CONFIGURATION REGISTER - P4PINCFG -SFR FDH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P47IN1OUT0	When: 1 = I/O pin acts as a input (reset value) 0 = I/O pin acts as a output
6	P46IN1OUT0	Same as bit 7
5	P45IN1OUT0	Same as bit 7
4	P44IN1OUT0	Same as bit 7
3	P43IN1OUT0	Same as bit 7
2	P42IN1OUT0	Same as bit 7
1	P41IN1OUT0	Same as bit 7
0	P40IN1OUT0	Same as bit 7

TABLE 45:PORT 5 PIN DIRECTION CONFIGURATION REGISTER - P5PINCFG -SFR FEH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P57IN1OUT0	When: 1 = I/O pin acts as a input (reset value) 0 = I/O pin acts as a output
6	P56IN1OUT0	Same as bit 7
5	P55IN1OUT0	Same as bit 7
4	P54IN1OUT0	Same as bit 7
3	P53IN1OUT0	Same as bit 7
2	P52IN1OUT0	Same as bit 7
1	P51IN1OUT0	Same as bit 7
0	P50IN1OUT0	Same as bit 7

*VRS51L30XX (QFP-64) DEVICES ONLY. ON VRS51L31XX (QFP-44) DEVICES THE P5PINCFG SFR CAN BE USED AS USER SCRATCHPAD REGISTER

TABLE 46:PORT 6 PIN DIRECTION CONFIGURATION REGISTER - P6PINCFG -SFR FFH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	P67IN1OUT0	When: 1 = I/O pin acts as a input (reset value) 0 = I/O pin acts as a output
6	P66IN1OUT0	Same as bit 7
5	P65IN1OUT0	Same as bit 7
4	P64IN1OUT0	Same as bit 7
3	P63IN1OUT0	Same as bit 7
2	P62IN1OUT0	Same as bit 7
1	P61IN1OUT0	Same as bit 7
0	P60IN1OUT0	Same as bit 7

*VRS51L30XX (QFP-64) DEVICES ONLY. ON VRS51L31XX (QFP-44) DEVICES THE P5PINCFG SFR CAN BE USED AS USER SCRATCHPAD REGISTER

6.3 I/O Ports Input Enable Register

Upon reset, all the I/Os are configured as inputs and the input control logic of all ports is activated. A given I/O port's input logic can be deactivated by clearing the corresponding bit in the PORTINEN register.

TABLE 47:PORTS INPUT ENABLE REGISTER - PORTINEN SFR F7H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7	Reserved (0)	Keep this bit at 0
6	P6INPUTEN*	Port 6 Input Enable Register 0 = Port 6 input logic is deactivated 1 = Port 6 input logic is activated
5	P5INPUTEN*	Port 5 Input Enable Register 0 = Port 5 input logic is deactivated 1 = Port 5 input logic is activated
4	P4INPUTEN	Port 4 Input Enable Register 0 = Port 4 input logic is deactivated 1 = Port 4 input logic is activated
3	P3INPUTEN	Port 3 Input Enable Register 0 = Port 3 input logic is deactivated 1 = Port 3 input logic is activated
2	P2INPUTEN	Port 2 Input Enable Register 0 = Port 2 input logic is deactivated 1 = Port 2 input logic is activated
1	P1INPUTEN	Port 1 Input Enable Register 0 = Port 1 input logic is deactivated 1 = Port 1 input logic is activated
0	P0INPUTEN	Port 0 Input Enable Register 0 = Port 0 input logic is deactivated 1 = Port 0 input logic is activated

*On VRS51L30xx, QFP-64 devices only

6.4 I/O Ports SFR Registers

As is the case for standard 8051 devices, the I/O ports are mapped into SFR registers that are bit-addressable. At reset, the I/O ports are activated and configured as inputs.

The I/O output drivers, unlike the original standard 8051 I/O output drivers, are of the push-pull type. Therefore the I/O pins have the same output drive capability whether they are driving a logic high or a logic low, versus the standard 8051s, which feature an active low driver with a pull-up resistor.

From a software point of view, the difference is that whenever the configuration of a given I/O has to be changed, the corresponding bit in the port direction configuration register must be set accordingly.

The following tables describe the SFR registers associated with the VRS51L3xxx I/O ports.

On the VRS51L31xx (QFP-44) devices, the SFR associated to I/O port P5 and P6 are accessible, but not pinned out. These SFR can be used as bit addressable scratch pad registers if needed.

TABLE 48:PORT 0 REGISTER - P0 SFR 80H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P0[7 :0]	Port 0

TABLE 49:PORT 1 REGISTER - P1 SFR 90H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P1[7 :0]	Port 1

TABLE 50:PORT 2 REGISTER - P2 SFR A0H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P2[7 :0]	Port 2

TABLE 51:PORT 3 REGISTER - P0 SFR B0H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P3[7 :0]	Port 3

TABLE 52:PORT 4 REGISTER - P4 SFR C0H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P4[7 :0]	Port 4

TABLE 53:PORT 5 REGISTER - P5 SFR 98H (VRS51L30XX (QFP-64) DEVICES ONLY)

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P5[7 :0]	Port 5

TABLE 54:PORT 6 REGISTER - P6 SFR C8H (VRS51L30XX (QFP-64) DEVICES ONLY)

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Bit	Mnemonic	Description
7:0	P6[7 :0]	Port 6

6.5 I/O Port Drive Capability

The current drive capability of the I/O ports is not the same for all ports. Most I/O pins can drive 2mA and others can drive more in either current source or current sink and can be used for direct LED drive. The following table summarizes the VRS51L3xxx I/O port drive capabilities:

TABLE 55:I/O PORTS DRIVING CAPABILITY

I/O Port	Max Current on Individual Pin
Port 0[7:0]	2mA
Port 1[7:5]	4mA
Port 1[4:0]	2mA
Port 2[7:0]	8mA
Port 3[7:6]	2mA
Port 3[5:4]	4mA
Port 3[3:0]	2mA
Port 4[7:0]	2mA
Port 5[7:0]*	16mA
Port 6[7:0]*	2mA

*VRS51L30xx devices only

It is not recommended to exceed the sink current specified in the table above. Doing so will likely cause the low-level output voltage to exceed device specifications and affect device reliability.

The recommended total DC load on the I/O ports should not exceed 100mA.

6.6 I/O Port Software Specifics

Some instructions allow the user to read the logic state of the output pin, while others allow the user to read the contents of the associated port register. These instructions are called *read-modify-write* instructions. A list of these instructions may be found in the following table.

Upon executing these instructions, the content of the port register (at least 1 bit) is modified. The other read instructions take the present state of the input into account. For example, instruction ANL P3,#01h obtains the value in the P3 register; performs the desired logic operation with the constant 01h and recopies the result into the P3 register.

In order to monitor the present state of the inputs of an I/O port bit, first, read the port, and second, perform an AND or an OR operation, as required by the program:

```
MOV A, P3; State of the inputs in the accumulator
ANL A, #01; AND operation between P3 and 01h
```

When the port is used as an output, the register contains information on the state of the output pins. Measuring the state of an output directly on the pin is inaccurate because the voltage level depends mostly on the type of charge that is applied to it. The functions below perform the operation on the value of the port register rather than the actual port pin itself.

TABLE 56: LIST OF INSTRUCTIONS THAT READ AND MODIFY THE PORT USING REGISTER VALUES

Instruction	Function
ANL	Logical AND ex: ANL P0, A
ORL	Logical OR ex: ORL P2, #01110000B
XRL	Exclusive OR ex: XRL P1, A
JBC	Jump if the bit of the port is set to 0
CPL	Complement 1 bit of the port
INC	Increment the port register by 1
DEC	Decrement the port register by 1
DJNZ	Decrement by 1 and jump if the result is not equal to 0
MOV P.,C	Copy the held bit C to the port
CLR* P.x	Set the port bit to 0
SETB P.x	Set the port bit to 1

***Note**: Even though the CPU does not read in this case, it is considered a *read-modify-write* instruction. In MOV dir, dir has an extra cycle when doing an SFR read during a debugger interrupt. The debugger memory is synchronous and is mapped into the SFR bus and, therefore, requires an extra read cycle.

Instruction A5, which is considered an NOP in a standard 8051, has been redefined to perform write and read SFR indirect addressing. Therefore, during a debugger interrupt, the A5 indirect read SFR addressing requires an extra cycle.

6.7 I/O Port Example Programs

Code Example: I/O Ports Toggle Example

This program shows the activation and configuration of ports P0 to P4 as outputs. The program continuously toggles their values.

```
;*************************************************
;*   VRS51L3074 I/O Ports Toggle Example  *
;*************************************************

START:MOV   PERIPHEN2,#08H   ;ENABLE IO
      MOV   P0PINCFG,#00H    ;CONFIGURE P0 AS OUTPUT
      MOV   P1PINCFG,#00H    ;CONFIGURE P1 AS OUTPUT
      MOV   P2PINCFG,#00H    ;CONFIGURE P2 AS OUTPUT
      MOV   P3PINCFG,#00H    ;CONFIGURE P3 AS OUTPUT
      MOV   P4PINCFG,#00H    ;CONFIGURE P4 AS OUTPUT

      MOV   PERIPHEN2,#00001000B ;BIT7 - PWC1EN
                                 ;BIT6 - PWC0EN
                                 ;BIT5 - AUEN
                                 ;BIT4 - XRAM2CODE
                                 ;BIT3 - IOPORTEN
                                 ;BIT2 - WDTEN
                                 ;BIT1 - PWMSFREN
                                 ;BIT0 - FPIEN
      // I/O Output Toggle Loop
LOOP:   MOV   P0,#00H            ;FORCE P0 = 00H
        MOV   P1,#00H            ;FORCE P1 = 00H
        MOV   P2,#00H            ;FORCE P2 = 00H
        MOV   P3,#00H            ;FORCE P3 = 00H
        MOV   P4,#00H            ;FORCE P4 = 00H

        MOV A,#100               ;Wait 100ms using Timer 0
        ACALL DELAY1MST0         ;See Timer  section
        MOV P0,#0FFH             ;FORCE P0 = FFH
        MOV P1,#0FFH             ;FORCE P1 = FFH
        MOV P2,#0FFH             ;FORCE P2 = FFH
        MOV P3,#0FFH             ;FORCE P3 = FFH
        MOV P4,#0FFH             ;FORC E P4 = FFH
        MOV A,#100               ;Wait 100ms using Timer0
        ACALL DELAY1MST0         ;See Timer Section
        LJMP  LOOP
```

The DELAY1MS function is described in the timers section.

Code Example: I/O Port Read Example

```
;-------------------------------------------------------------------------
;*   VRS51L3074 I/O Ports Read and Write  Example
;-------------------------------------------------------------------------
PORTREAD    EQU 021H    ;GENREAL VARIABLE

START:MOV   PERIPHEN2,#08H   ;ENABLE IO
      MOV   P0PINCFG,#00H    ;CONFIGURE P0 AS iNTPUT
      MOV   P1PINCFG,#00H    ;CONFIGURE P2 AS OUTPUT

      ; Note that the port Input logic is activated by default

      MOV   PERIPHEN2,#00001000B ;BIT7 - PWC1EN
                                 ;BIT6 - PWC0EN
                                 ;BIT5 - AUEN
                                 ;BIT4 - XRAM2CODE
                                 ;BIT3 - IOPORTEN
                                 ;BIT2 - WDTEN
                                 ;BIT1 - PWMSFREN
                                 ;BIT0 - FPIEN

;*** Read Port 0 and copy the value to P2

LOOP:   MOV   PORTREAD, P0       ;Read Prt 0 and store the value in a Variable
        MOV   P2, PORTREAD       ;Write the Variable content to P2
        AJMP  LOOP
```

In this example, the Port P0 value is stored in a variable before writing it to P2, but the user can also directly transfer P0 to P2 in one operation:

```
LOOP:   MOV   P2,P0              ;would do the same operation more efficiently
        AJMP  LOOP
```

6.8 I/O Port Pin Change Monitoring

The VRS51L3xxx includes an I/O port pin change monitoring subsystem. This module is used to monitor the activity on the selected I/O ports.

When enabled, if a pin state changes on the selected I/O port, the PMONFLAG will be set to 1 by the system. It must be cleared manually by the software.

The port pin change monitoring feature is very useful for monitoring events that can occur on a given group of I/Os without having to constantly read the I/O state. Since it is connected to the VRS51L3xxx interrupt subsystem, the port pin change monitoring system frees the processor resources for other tasks.

TABLE 57:PORT CHANGE MONITORING REGISTER - PORTCHG SFR B9H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PMONFLAG1	Port Change Monitoring Flag1 When set, monitored port state has changed
6	PCHGMSK1	Port Change Mask Register 1 0 = Port monitoring is deactivated 1 = Port monitoring is activated
5:4	PCHGSEL1[1:0]	Port Change Monitoring Register Select 1 00 = P4 Change is monitored 01 = P5 Change is monitored** 10= P6 Change is monitored** 11 = P4[3:0] Change is monitored
3	PMONFLAG0	Port Change Monitoring Flag 0 When set, monitored port state has changed
2	PCHGMSK0	Port Change Mask Register 0 0 = Port monitoring is deactivated 1 = Port monitoring is activated
1:0	PCHGSEL0[1:0]	Port Change Monitoring Register Select 0 00 = P0 Change is monitored 01 = P1 Change is monitored 10 = P2 Change is monitored 11 = P3 Change is monitored

**Available on VRS51L30xx (QFP-64) devices only

The port pin change monitoring flags, PMONFLAGx, are active at all times, even if the port change masks are not activated. The PCHGMSKx bits serve to connect the port change module to the VRS51L3xxx interrupt system. The port change monitoring flags must be cleared manually. I/O Port Pin Change Interrupt Example Programs

6.9 I/O Port Pin Change Monitoring Code Example

Code Example: Numeric Keypad Interface

```
//-------------------------------------------------------------------------------------------------------------------//
// VRS51L3074_KeypadP0_LCDP1.c /
//-------------------------------------------------------------------------------------------------------------------//
//
// DESCRIPTION:    Character LCD and Numeric Keypad Interface Example Program.
//
//              This program initialize and sends LCD strings and numeric values
//              to a character based LCD display.

//              The program also demonstrate the use of the Port Change interrupt
//              Feature of the VRS51L3074 to simplify the interface with a numeric Keypad
//              on Port 0.
//              The numeric keypad is a standard phone keypad which to connected to Port 0
//              as shown below:
//              Column 3  - P0.7
//              Column 2  - P0.6
//              Column 1  - P0.5
//              Row   4  - P0.3
//              Row   3  - P0.2
//              Row   2  - P0.1
//              Row   1  - P0.0//
//              No external pull-up / pull down resistors are required, thank to the
//              presence of internal pull-up on the VRS51L3074 I/O ports.
//
//              The interface to the LCD done through the VRS51L3074 Port 1.
//              The LCD is initialized to operate in 4 bit data Bus Mode
//
//              LCD interface structure:
//              ========================
//              P1.0     =   LCD RS
//              P1.1     =   LCD RW
//              P1.2     =   LCD E
//              P1.3     =   (not used)
//              P1[7:4]  =    LCD Data (4 bit mode)
//              Notes about standard Character LCD display interface to the VRS51L3074
//                  -Most LCD displays operates on a 4.5V to 5.5V Supply.
//                   They won't work with the 3.3V supply the VRS51L3074 operate from
//                  -On the digital side make sure the LCD module logic High level lower limit
//                   is below 3V.
//                  -The VRS51L3074 I/Os are 5V tolerant, so there is no need to add interface
//                   circuit between the LCD module's I/O and the VRS51L3074 I/O
//-------------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

//--LCD I/O definition
#define LCDPORT P1
#define LCDPORTDIR P1PINCFG
//--Keypad I/O definition
#define KEYPADPORT    P0
#define KEYPADPORTDIR  P0PINCFG

//---Keypad Function prototypes
char KeyDecode();
void KeyDisplay(char);
//---LCD Function prototypes
void lcdbusy(void);                    //LCD Busy check
void initlcd(void);                    //LCD Initialisation function
void LCDSlow(void);                    //Slow Down communication with LCD display
void int2lcd(unsigned int);            //Integer to LCD display function
void lcdstring( char code *);          //String to LCD display function
void sendlcdchar( char);               //Char to LCD Display function
void sendlcdcmd( unsigned char);       //Send LCD Command Function

//---Generic Functions prototype
void V2KDelay1ms(unsigned int);        //Standard Delay function
// LCD bit variables
bit at 0x92 LCD_E;                     //LCD E Line
bit at 0x90 LCD_RS;                    //LCD RS
bit at 0x91 LCD_RW;                    //LCD RW

// Global variables definitions
idata unsigned char cptr = 0x00;

// LCD Strings and constants definitions
code char msg1[]=  "VRS51L3074      \0";
code char msg2[]=  "Waiting for Key.\0";
code char msgkey[]= "Last key:      \0";

code char LCD_L1C1 = 0x80;         //Command LCD set CGRAM addr to Line1,column 1
code char LCD_L2C1 = 0xC0;         //Command LCD set CGRAM addr to Line2,column 1
code char LCD_L2C10 = 0xC9;        //Command LCD set CGRAM addr to Line2,column
10
code char LCD_CLEAR = 0x01;        //Command LCD Clear and return cursor home
```

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

void main (void) {

    PERIPHEN1 = 0x01;               //Enable Timer 0
    LCDPORTDIR = 0x00;              //Config LCD port as output

    //--Configure Keypad Port and port Change monitor
    KEYPADPORTDIR = 0x0F;           //KeypadPort bit 3:0 -> configured as Input (Lines)
                                    //KeypadPort bit 7:5->Configured as output (Columns)
    KEYPADPORT = 0x0F;              //Clear the Columns driver outputs
    V2KDelay1ms(100);               //Put a 100 milliseconds delay

    PORTCHG = 0x04;                 //Disable Port Change monitoring Module 1
                                    //Enable Port Change monitoring Module 0
                                    //Clear the Port Change monitoring Flag
                                    //Port 0 Change is monitored

    //-- Activate port change interrupt

    INTSRC1 &= 0xEF;                //Force Interrupt vector 4 to be routed to Port Change
                                    //module 0
    INTEN1 |= 0x10;                 //Enable the PORT CHANGE 0 Module Interrupt
    GENINTEN = 0x01;                //Activate the Global Interrupts

    //--Initialize the LCD
    initlcd();                       //Initialise the LCD Module
    sendlcdcmd(LCD_L1C1);           //Place LCD cursor on Line 1, Column 1

    cptr = 0;
    while( msg1[cptr] != '\0')      //Display "VRS51L3074" on first line of LCD display
        sendlcdchar( msg1[cptr++]);

    sendlcdcmd(LCD_L2C1);            //Place LCD cursor on Line 2, Column 1

    cptr = 0;
    while( msg2[cptr] != '\0')      //Display "Waiting for Key.\0" on 2 line of LCD display
        sendlcdchar( msg2[cptr++]);
    V2KDelay1ms(1000);              //Put a 1 seconds delay

    //--Loop Waiting for Keys to be pressed
    while(1);                       //Infinite Loop

}// End of main

//-------------------------------------------------------------------------------------------//
//----------------------Port Change Interrupt Function(s
//-------------------------------------------------------------------------------------------//
void PortChange0Int(void) interrupt 4
    {
    unsigned char keypressed = 0x00;        //var holding ASCII value of the last key
                                            //pressed (could be global)
    unsigned char keylines = 0x00;          //variable to read the actual I/O port
    unsigned char keyrow = 0x00;            //Row position of the pressed key
    unsigned char keycol = 0x00;            //Column position of the pressed key

    // rows and columns association table
    const char code keyrowmap[] =
 {0x0F,0x0F,0x0F,0x0F,0x0F,0x0F,0x0F,0x03,0x0F,0x0F,0x0F,0x02,0x0F,0x01,0x00,0x0F};
    const char code keycolmap[]={0x0F,0x0F,0x0F,0x02,0x0F,0x01,0x00,0x0F};

    // Ascii code associated with pressed key
    const char code keyascii[4][3] =   {
                                {'1','2','3'},
                                {'4','5','6'},
                                {'7','8','9'},
                                {'*','0','#'}};

    GENINTEN = 0x00;                        //Disable the Global Interrupts

    //--Retrieve the line number
    KEYPADPORT = 0x00;                  //Send 0 on each column
    V2KDelay1ms(10);                    //Put a 10 millisecond delay
    keylines = KEYPADPORT;              //Read Keypad Port
    keylines &=0x0F;                    //Isolate lower nibble

    if(keylines != 0x0F)
        {
        //-retrieve the line value
        keyrow = keyrowmap[keylines];

       //--Retrieve Column number
        KEYPADPORTDIR = 0xF0;           //columns are input / rows  are output
        KEYPADPORT = 0x00;              //Send 0 on each row

        V2KDelay1ms(10);                //Put a 10 millisecond delay
        keylines = KEYPADPORT;          //Read Keypad Port
        B = keylines;
        keylines &=0xE0;                //Isolate upper 3 bit (columns)


        keylines = (keylines >> 5);     //Position columns to lower portion
        //-retrieve the line value
        keycol = keycolmap[keylines];

        if((keyrow != 0x0F)&& (keycol != 0xFF))
            {
            //--Get the ascii value of the key
            keypressed = keyascii[keyrow][keycol];
            sendlcdcmd(LCD_L2C1);       /Place LCD cursor on Line 2, Column 1
            cptr = 0;
            while( msgkey[cptr] != '\0')    //Display "Last key:      \0"
            sendlcdchar( msgkey[cptr++])   //on second line of LCD display

            //Display the key value on the LCD display
            sendlcdcmd(LCD_L2C10);      //Place LCD cursor on Line 2, Column 10
            sendlcdchar(keypressed);
            }//end of if key row / col

            //--wait for the key to be released
        do{
            B= KEYPADPORT;
            B &= 0xE0;
            }while(B != 0xE0 );

        //--Set KEYPADPORT as before
        KEYPADPORTDIR = 0x0F;           //Columns are input / rows  are output
        KEYPADPORT = 0x0F;              //Clear the Columns driver outputs

        V2KDelay1ms(10);                // Put a 10 millisecond delay
        }//end of if keylines != 0xFF


    PORTCHG = 0x04;                     //Disable Port Change monitoring Module 1
                                        //Enable Port Change monitoring Module 0
                                        //Clear the Port Change monitoring Flag
                                        //Port 0 Change is monitored

    GENINTEN = 0x01;                    //Activate the Global Interrupts

    }//End of Port Change Interrupt

//-------------------------------------------------------------------------------------------------------//
//                                   INDIVIDUALS FUNCTIONS
//-------------------------------------------------------------------------------------------------------//

(See demonstration programs… )
```

7 Timers

The VRS51L3xxx devices include three 16-bit timers: Timer 0, Timer 1 and Timer 2. The timers include much more functionality and features than standard 8051 timers:

- Timers 0, 1 can operate as one 16-bit timer or two 8-bit timers
- Timers can count up/count down
- Each timer includes a configurable divisor
- Timers can be chained together to form 24-, 32- or 48-bit timer/counters
- Each timer features an output that can generate a pulse or toggle when the timer overflows
- Each timer provides counter input
- Each timer provides a gating pin

VRS51L3xxx devices timers include a number of parameters that can be adjusted independently, enabling countless configurations to suit a diversity of timing/counting applications. The structure of the timer configuration registers has been simplified compared to standard 8051 timer control registers.

The architecture of the registers controlling the three timers is the same for Timer 0 and Timer 1 and almost the same for Timer 2.

On the VRS51L31xx devices (QFP-44) the following Timer inputs are not accessible:

- T2EX alternate input
- T2IN alternate input
- T0EX input
- T1EX input

7.1 Timer 0 / 1 Configuration

Timer 0 and Timer 1 operation is controlled by three registers. The configuration of timers 0/1 is essentially the same.

7.2 The T0T1CFG Register Overview

The T0T1CFG register controls the gating features of both Timer 1 and Timer 0. The TxGATE bit controls the clock gating of the timers. When this bit is set to 1, the timer will only count when the INTx pin is high.

TABLE 58: TIMER 0 / TIMER1 CONFIGURATION REGISTER - T0T1CFG SFR 89H

7	6	5	4	3	2	1	0
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	-	Not used
6	T1GATE	Timer 1 Gating Enable 0 = Timer 1 gating feature is disabled 1 = Timer 1 count only when INT1 pin is high
5	T0GATE	Timer 0 Gating Enable 0 = Timer 0 gating feature is disabled 1 = Timer 0 count only when INT0 pin is high
4	T1CLKSRC	Timer 1 Clock Source 0 = Timer 1 takes its clock from system clock 1 = Timer 1 takes its clock from Timer 0 output
3	T1OUTEN	Timer 1 Output Enable 0 = Timer 1 output is deactivated 1 = Timer 1 output is connected to a pin
2	T1MODE8	Timer 1 8-bit Operating Mode Enable 0 = Timer 1 operates as a 16-bit timer 1 = Timer 1 operates as two 8-bit timers
1	T0OUTEN	Timer 0 Output Enable 0 = Timer 0 output is deactivated 1 = Timer 0 output is connected to a pin
0	T0MODE8	Timer 1 8-bit Operating Mode Enable 0 = Timer 1 operates as a 16-bit timer 1 = Timer 1 operates as two 8-bit timers

The T1CLKSRC bit defines which clock source will feed Timer 1 when it is configured to operate in timer mode. The Timer 1 clock source is defined as follows:

- T1CLKSRC = 0 System Clock
- T1CLKSRC = 1 Timer 0 Output (overflow)

When configured in timer mode, Timer 0 can only derive its clock source from the system clock with the proper prescaler value. Both timers 1 and 0 can operate as two general purpose 8-bit timers. This mode is activated by setting the corresponding TxMODE8 bit of the T0T1CFG register to 1.

TABLE 59:TIMER0 / TIMER 1 CLOCK CONFIG. REGISTER - T0T1CLKCFG SFR 99H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:4	T1CLKCFG[3:0]	Timer 1 Clock Prescaler Configuration see table below
3:0	T0CLKCFG[3:0]	Timer 0 Clock Prescaler Configuration see table below

TABLE 60:TIMER0 / TIMER 1 CLOCK DIVISION RATIO

T0/1CLKCFG (4 bit binary)	Timer Clock Div. Ratio	T0/1CLKCFG	Timer Clock Div. Ratio
0000	1	1000	256
0001	2	1001	512
0010	4	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	16384

7.3 The T0CON and T1CON Registers

The T0CON and T1CON SFR registers control the following:

- Timer operation mode (timer or counter)
- Advanced gating features of Timer 0 and Timer 1
- Timer overflow flag
- Counting direction (up/down)
- Timer reload and capture
- Timer output mode (Pulse/Toggle)

These registers are fully orthogonal, which means that for a given timer operating mode, the registers function in the same manner.

TABLE 61: TIMER 0 CONFIGURATION REGISTER - T0CON SFR 9AH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	T0OVF	Timer 0 Overflow Flag Set to 1 when timer overflow from FFFFh to 0000h. Must be cleared by software. (T0 must be activated and running (TR0 = 1) to clear T0OVF). Writing 1 into this bit will trigger a timer interrupt, if enabled
6	T0EXF	Timer 0 External Flag Gating Flag Set to 1 when timer reload of capture is caused by an high to low transition on the T0EX pin, if T0EXEN is set to 1
5	T0DOWNEN	Timer 0 Count Down Enable 0 = Timer 0 count up 1 = Timer 0 counts down
4	T0TOGOUT	Timer 0 Output Toggle Enable 0 = Timer 0 output outputs a pulse when it overflow occurs 1 = Timer 0 output toggle when it overflow occurs
3	T0EXTEN	Timer 0 External Gating Enable 0 = T0EX pin is not active 1 = Enable Timer 0 capture or reload upon a high to low transition on the T0EX pin when the timer 0 is configured in counter mode.
2	TR0	Timer 0 Run 0 = Timer 0 is stopped 1 = Timer 0 is running
1	T0COUNTEN	Timer 0 Counter Enable 0 = Timer 0 acts as a timer 1 = Timer 0 acts as a counter that is incremented (decremented) by a high to low transition on T0IN pin
0	T0RLCAP	Timer 0 Capture Enable 0 = Auto reload value is loaded in Timer 0, if a high to low transition occurs on T0EX, if T0EXTEN is set to 1. 1 = Timer 0 current value is captured when a high to low transition occurs on the T0EX pin, if T0EXTEN is set to 1

TABLE 62: TIMER 1 CONFIGURATION REGISTER - T1CON SFR 9BH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	T1OVF	Timer 1 Overflow Flag Get set to 1 when timer overflow from occurs. Set to 1 when timer overflow occurs. Must be cleared by software. (T1 must be activated and running (TR1 = 1) to clear T1OVF). Writing 1 into this bit will trigger a timer interrupt, if enabled
6	T1EXF	Timer 1 External Flag Gating Flag Get set to 1 when timer reload of capture is caused by an high to low transition on the T1EX pin, if T1EXEN is set to 1
5	T1DOWNEN	Timer 1 Count Down Enable 0 = timer 1 count up 1 = Timer 1 counts down
4	T1TOGOUT	Timer 1 Output Toggle Enable 0 = Timer 1 output outputs a pulse when timer overflow occurs 1 = Timer 1 output toggle when it overflow from FFFFh to 0000h
3	T1EXTEN	Timer 1 External Gating Enable 0 = T1EX pin is not active 1 = Enable Timer 1 capture or reload upon a high to low transition on the T1EX pin when the timer 1 is configured in counter mode.
2	TR1	Timer1 Run 0 = Timer 1 is stopped 1 = Timer 1 is running
1	T1COUNTEN	Timer 1 Counter Enable 0 = Timer 1 acts as a timer 1 = Timer 1 acts as a counter that is incremented (decremented) by a high to low transition on T1IN pin
0	T1RLCAP	Timer 1 Capture Enable 0 = Auto reload value is loaded in Timer 1, if a high to low transition occurs on T1EX, if T1EXTEN is set to 1 1 = Timer 1 current value is captured when a high to low transition occurs on the T1EX pin, if T1EXTEN is set to 1.

The TxOVF bit of the TxCON register indicates that the timer count has rolled over from FFFFh to 0000h (or 0000h to FFFFh). If the corresponding timer interrupt has been enabled, the TxOVF will raise the interrupt. TxOVF will not get set when the timer is configured in auto reload mode.

The TxEXF flags are set to 1 when a high to low transition occurs on the corresponding TxEX pin, provided that the TxEXEN pin is set to 1.

Timer 0 and Timer 1 can count up or down. By default, the timers count up. However setting the TxDOWNEN bit to 1 will make the timer count down .

The TxCOUNTEN bit allows the timer to be configured as an external event counter.

By default, the timers derive their source from the system clock or a prescaled source. Setting the TxCOUNTEN bit to 1, will configure the corresponding timer to derive its source from the timer input pin (TxIN). A high to low transition on the timer input pin

will make the timer count one step up or one step down, depending on the value of the corresponding TxDOWNEN bit.

The TxRLCAP bit defines the function of the timer capture/reload register upon a high to low transition on the TxEX timer trigger input pin.

- o TxRLCAP = 0 : Auto reload value is loaded in Timer x
- o TxRLCAP = 1 : Timer x current value will be captured

The functions associated with the TxRLCAP bit are only activated when the corresponding TxEXTEN bit is set to 1.

7.4 Timer 0 / 1 Current Value Register

Two SFR registers provide access to the current 16-bit value of Timer 0 and Timer 1.

TABLE 63:TIMER 0 LOW - TL0 SFR 8AH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	TL0[7:0]	

TABLE 64:TIMER 0 HIGH - TH0 SFR 8BH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	TH0[7:0]	

TABLE 65:TIMER 1 LOW - TL1 SFR 8CH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	TL1[7:0]	

TABLE 66:TIMER 1 HIGH - TH0 SFR 8DH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	TH0[7:0]	

7.5 Timer 0 Reload and Capture Registers

Both Timer 0 and Timer 1 have an auxiliary 16-bit reload/capture register, which is accessible through two SFR registers as follows:

TABLE 67:TIMER 0 RELOAD AND CAPTURE LOW - RCAP0L SFR 92H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	RCAP0L[7:0]	

TABLE 68:TIMER 0 RELOAD AND CAPTURE HIGH - RCAP0H SFR 93H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	RCAP0H[7:0]	

TABLE 69:TIMER 1 RELOAD AND CAPTURE LOW – RCAP1L SFR 94H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	RCAP1L[7:0]	

TABLE 70:TIMER 1 RELOAD AND CAPTURE HIGH – RCAP1H SFR 95H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	RCAP1H[7:0]	

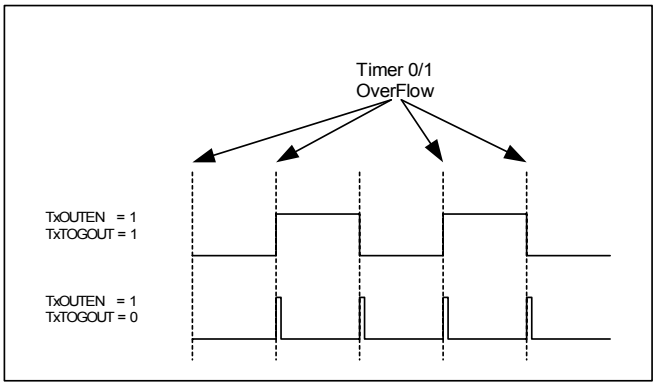
The content of the Timer reload capture registers (RCAPxH / RCAPxL) will be reloaded into the timer when the timer overflow occurs.

7.6 Timer 0 / 1 Output

Timer 0 and Timer 1 outputs can be routed to an external pin. This feature is activated by setting the TxOUTEN bit of the TxCLKCFG register to 1. By default, the timer outputs, when enabled, will generate a pulse upon timer overflow. The duration of the pulse equals 1/ SYS CLK.

Setting the TxTOGOUT bit of the TxCON register to 1 will configure the timer x output to toggle upon a timer overflow instead of generating a pulse.

FIGURE 16: TIMER 0, TIMER 1 OUTPUT MODES



7.7 Timer 0 / 1 Alternate Mapping

Bits 0 and 1 of the DEVIOMAP register (SFR E1h) control the mapping of the Timer 0 and Timer 1 peripherals as shown in the following tables.

TABLE 71: TIMER 0 PIN MAPPING

DEVIOMAP.0 Bit Value	T0IN mapping	T0EX mapping	T0OUT mapping
0 (Reset)	P3.4	P2.6	P4.5
1	-	Pin 41 (VRS51L30xx)	-

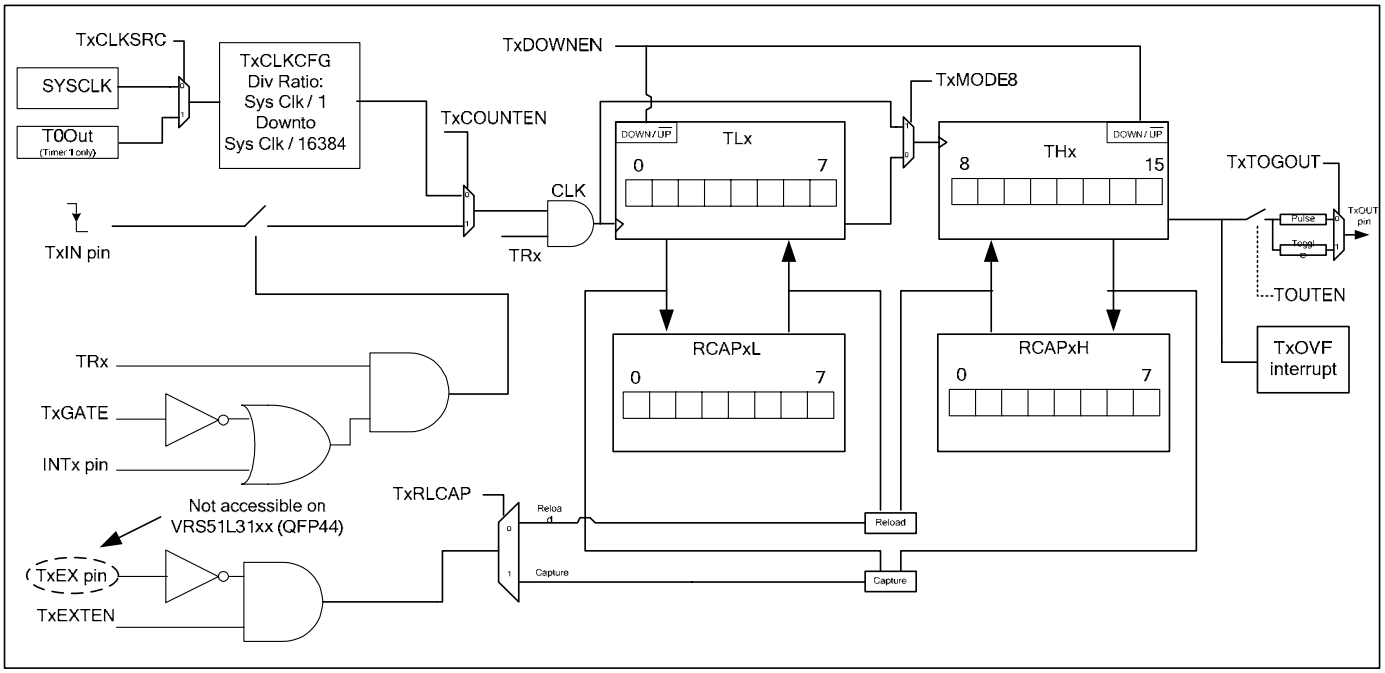
TABLE 72: TIMER 1 PIN MAPPING

DEVIOMAP.1 Bit Value	T1IN mapping	T1EX mapping	T1OUT mapping
0 (Reset)	P3.5	P2.5	P4.0
1	-	Pin 40 (VRS51L30xx)	P1.4

7.8 Timer 0 / 1 Functional Diagram

The following diagram represents the main features of timers 0 and 1

FIGURE 17: TIMER 0, TIMER 1 FUNCTIONAL DIAGRAM



7.9 Timer 0 / 1 Example Programs

Code Example:
Timer 0 1ms Delay Function

```
;*************************************************************
;* DELAY1MSTO : 1MS DELAY USING TIMER0
;*; *CALIBRATED FOR 40MHZ
;*************************************************************
;
DELAY1MST0: MOV   CPTR,A            ;GET NUMBER OF CYCLES
            MOV   A,PERIPHEN1       ;LOAD PERIPHEN1 REG
            ORL   A,#00000001B      ;ENABLE TIMER 0
            MOV   PERIPHEN1,A


DELAY1MSLP: MOV   TH0,#063H         ;T0  RELOAD VALUE FOR 1MS AT 40MHZ
            MOV   TL0,#0C0H
            ;MOV  TH0,#0A9H         ;T0 RELOAD VALUE FOR 1MS AT 22.11MHZ
            ;MOV  TL0,#058H

            MOV   T0T1CLKCFG,#00H       ;NO PRESCALER FOR T0 CLOCK
            MOV   T0CON,#00000100B      ;START T0,  COUNT UP

DWAITOVT0:  MOV   A,T0CON           ;READ T0 CONTROL, WAIT FOR
                                    ;OVERFLOW
            ANL   A,#080H           ;ISOLATE TIMER OVERFLOW FLAG
            JZ    DWAITOVT0         ;LOOP AS LONG AS T0 DON'T OVERFLOW

            MOV   T0CON,#00H        ;STOP TIMER 0
            DJNZ  CPTR,DELAY1MSLP   ;Out Loop

            MOV   A,PERIPHEN1       ;LOAD PERIPHEN1 REG
            ANL   A,#11111110B      ;DISABLEBLE TIMER 0
            MOV   PERIPHEN1,A
            RET
```

Code Example:
Timer 0, Timer 1 and Timer 2 Output Toggle

```
;*************************************************************************
;- TIMER 0, TIMER 1 AND TMER 2,  OUTPUT TOGGLE EXAMPLE   *
;*************************************************************************
;
Include <VRS51L3074.inc>

;-- Enable Timer 0, Timer 1 and Timer 2
INIT:   MOV   PERIPHEN1,#00000111B  ;BIT7 - SPICS EN
                                    ;BIT6 - SPIEN
                                    ;BIT5 - I2CEN
                                    ;BIT4 - U1EN
                                    ;BIT3 - U0EN
                                    ;BIT2 - T2EN
                                    ;BIT1 - T1EN
                                    ;BIT0 - T0EN

        MOV   PERIPHEN2,#00001000B  ;BIT7 - PWC1EN
                                    ;BIT6 - PWC0EN
                                    ;BIT5 - AUEN
                                    ;BIT4 - XRAM2CODE
                                    ;BIT3 - IOPORTEN
                                    ;BIT2 - WDTEN
                                    ;BIT1 - PWMSFREN
                                    ;BIT0 - FPIEN

      ;** CONFIGURE AND START TIMER 0, TIMER 1 & TIMER 2

        MOV   T0T1CFG,#00001010B    ;CONNECT TIMER0 OUTPUT T0 P4.5 and
                                    ;TIMER1 OUTPUT T0 P4.0, TIMER SOURCE
                                    ;FROM SYS CLK
        MOV   T2CLKCFG,#00010110B   ;T2 SSOURCE = SYS CLK, T2OUT
                                    ;ENABLED ON P1.2,  PRESCALER = SYS
                                    ;CLK/64

        MOV   T0CON,#14H            ;START TIMER0, TOGGLE OUTPUT
        MOV   T1CON,#14H            ;START TIMER1, TOGGLE OUTPUT
        MOV   T2CON,#14H            ;START TIMER2, TOGGLE OUTPUT

LOOP:  AJMP   LOOP                  ;INFINITE LOOP
```

Code Example:
Timer 0, Timer 1 and Timer 2 Output Toggle and Timer Chaining Example

```
;*************************************************************************************************
;- TIMER 0, TIMER 1 AND TMER 2,  OUTPUT TOGGLE + TIMER CHAINING EXAMPLE   *
;*************************************************************************************************
;
Include <VRS51L3074.inc>

INIT:   MOV   PERIPHEN1,#00000111B  ;BIT7 - SPICS EN
                                    ;BIT6 - SPIEN
                                    ;BIT5 - I2CEN
                                    ;BIT4 - U1EN
                                    ;BIT3 - U0EN
                                    ;BIT2 - T2EN
                                    ;BIT1 - T1EN
                                    ;BIT0 - T0EN

        MOV   PERIPHEN2,#00001000B  ;BIT7 - PWC1EN
                                    ;BIT6 - PWC0EN
                                    ;BIT5 - AUEN
                                    ;BIT4 - XRAM2CODE
                                    ;BIT3 - IOPORTEN
                                    ;BIT2 - WDTEN
                                    ;BIT1 - PWMSFREN
                                    ;BIT0 - FPIEN

        ;-- SET THE SYSTEM CLOCK PRESCALER TO MAX SPEED

      ;** CONFIGURE AND START TIMER 0, TIMER 1 & TIMER 2
        MOV   T0CON,#14H            ;START TIMER0, TOGGLE OUTPUT
        MOV   T1CON,#14H            ;START TIMER1, TIMER1 TOGGLE
                                    ;OUTPUT
        MOV   T2CON,#14H            ;START TIMER2, TIMER2 TOGGLE
                                    ;OUTPUT
        MOV   T0T1CFG,#00001000B    ;CONNECT TIMER1 OUTPUT T0 P4.0
        MOV   T2CLKCFG,#00110000B   ;TIMER 2 USES TIMER1 OUTPUT AS
                                    ;CLOCK SOURCE, T2 OUT ON P1.2,
                                    ;CLOCK PRESCALER  = 1

LOOP:  AJMP    LOOP                 ;INFINITE LOOP
```

7.10 Timer 2

The architecture of Timer 2 is very similar to that of timers 0 and 1, the main difference being that Timer 2 cannot operate as two 8-bit timers.

7.11 Timer 2 Configuration Registers

The T2CON register controls:

- o Timer operation mode (timer or counter)
- o Timer 2 advanced gating features
- o Timer 2 overflow flag
- o Timer 2 counting direction (up/down)
- o Timer 2 reload and capture
- o Timer 2 output mode (pulse/toggle)

The T2CON register has the same structure as the T0CON and T1CON registers.

TABLE 73:TIMER 2 CONFIGURATION REGISTER - T2CON SFR 9CH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	T2OVF	Timer 2 Overflow Flag Set to 1 when timer overflow from FFFFh to 0000h. Must be cleared by software. (T2 must be activated and running (TR2 = 1) to clear T2OVF). Writing 1 into this bit will trigger a timer interrupt, if enabled
6	T2EXF	Timer 2 External Flag Gating Flag Set to 1 when timer reload of capture is caused by an high to low transition on the T2EX pin, if T2EXEN is set to 1
5	T2DOWNEN	Timer 2 Count Down Enable 0 = Timer 2 count up 1 = Timer 2 counts down
4	T2TOGOUT	Timer 2 Output Toggle Enable 0 = Timer 2 output outputs a pulse when it overflows from FFFFh to 0000h 1 = Timer 2 output toggles when it overflows from FFFFh to 0000h
3	T2EXTEN	Timer 2 External Gating Enable 0 = T2EX pin is not active 1 = Enable Timer 2 capture or reload upon a high to low transition on the T2EX pin when the timer 2 is configured in counter mode.
2	TR2	Timer2 Run 0 = Timer 2 is stopped 1 = Timer 2 is running
1	T2COUNTEN	Timer 2 Counter Enable 0 = Timer 2 acts as a timer 1 = Timer 2 acts as a counter that is incremented (decremented) by a high to low transition on T2IN pin
0	T2RLCAP	Timer 2 Capture Enable 0 = Auto reload value is loaded in Timer 2 if a high to low transition occurs on T2EX, if T2EXTEN is set to 1 1 = Timer 2 current value is captured when a high to low transition occurs on the T2EX pin, if T2EXTEN is set to 1

The T2OVF bit of the T2CON register indicates whether the timer count has rolled over from FFFFh to 0000h. If the corresponding timer interrupt has been activated, the T2OVF will raise the Timer 2 interrupt..

The T2EXF flags are set to 1 when a high to low transition occurs on the T2EX pin, provided that the T2EXE pin is set to 1.

As is the case for timers 0 and 1, Timer 2 can be configured to count up or down. By default, Timer 2 counts up. However setting the T2DOWNEN bit to 1 will configure Timer 2 to count down. When the timer counts downwards, the overflow flag will be set when the timer counts from 0000h to FFFFh.

The T2COUNTEN bit- enables the configuration of Timer 2 as a external event counter. By default, Timer 2 derives its source from the system clock or a prescaled system clock. Setting the T2COUNTEN bit to 1 will configure Timer 2 to derive its source from the T2IN input pin. A high to low transition on the T2IN pin will initiate a timer count one step up or down, depending on the value of the corresponding T2DOWNEN bit.

The T2RLCAP bit controls the function of the timer capture/reload register when a high to low transition occurs on the T2EX timer trigger input pin.

- o T2RLCAP = 0 : Auto reload value is loaded in Timer 2
- o T2RLCAP = 1 : Timer 2 current value will be captured in the RCAP2L and RCAP2H registers

The functions associated with the T2RLCAP bit are only activated when the T2EXTEN bit is set to 1.

TABLE 74:TIMER 2 LOW - TL2 SFR 8EH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	TL2[7:0]	

TABLE 75:TIMER 2 HIGH – TH2 SFR 8FH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	TH2[7:0]	

7.12 Timer 2 Reload and Capture Registers

TABLE 76:TIMER 2 RELOAD AND CAPTURE LOW – RCAP2L SFR 96H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	RCAP2L[7:0]	

TABLE 77:TIMER 2 RELOAD AND CAPTURE HIGH – RCAP2H SFR 97H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	RCAP2H[7:0]	

The content of the Timer 2 reload capture registers (RCAP2H / RCAP2L) will be reloaded into the timer when the timer overflow occurs.

7.13 The Timer 2 Clock Configuration Register

The T2CLKCFG register is used to configure the clock source for Timer 2. The source can be either a prescaled value of the system clock or the output of Timer 1.

The Timer 2 clock source is also controlled by the T2CLKSRC bit. When this bit is set to 1, Timer 2 derives its source from the Timer 1 overflow. If T2CLKSRC is set to 0, Timer 2 will derive its source from a prescaled value of the system clock. The division factor applied to the system clock is defined by T2CLKCFG[3:0]

TABLE 78:TIMER2 CLOCK CONFIGURATION REGISTER - T2CLKCFG SFR 9DH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	-	
6	-	
5	T2CLKSRC	Timer 2 Clock Source 0 = Timer 2 take its clock from system clock 1 = Timer 2 takes its clock from Timer 1 output
4	T2OUTEN	Timer 2 Output Enable 0 = Timer 2 output is deactivated 1 = Timer 2 output is connected to a pin
3:0	T2CLKCFG[3:0]	Timer 2 Clock Prescaler Configuration See Table below

The following table outlines the Timer 2 prescaler values according to the value of the T2CLKCFG[3:0] bits.

TABLE 79:TIMER 2 CLOCK DIVISION RATIO

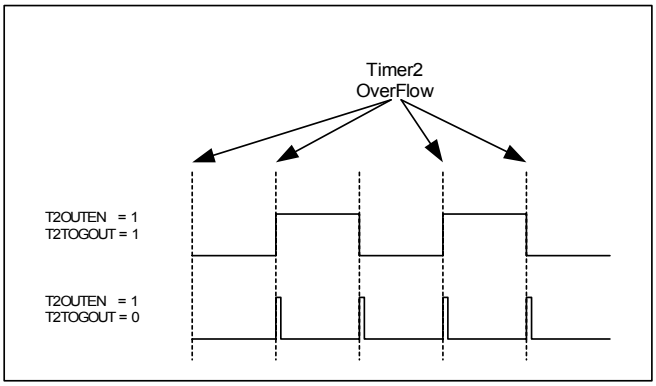
T2CLKCFG (4 bit binary)	Timer Clock Div. Ratio	T2CLKCFG	Timer Clock Div. Ratio
0000	1	1000	256
0001	2	1001	512
0010	4	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	16384

7.14 Timer 2 Output

As is the case for timers 0 and 1, Timer 2's output can be routed to an external pin. This feature is activated by setting the T2OUTEN bit of the T2CLKCFG register to 1. By default, the Timer 2 output, when enabled, will generate a pulse upon Timer 2 overflow. The duration of the pulse is (1/ SYS CLK).

Setting the T2TOGOUT bit of the T2CON register to 1 will configure Timer 2's output to toggle upon a Timer 2 overflow instead of outputting a pulse.

FIGURE 18: TIMER 2 OUTPUT MODES



7.15 Timer 2 Alternate Mapping

Bit 2 of the DEVIOMAP register (SFR E1h) controls the mapping of the Timer 2 interface as shown in the following table:

TABLE 80: TIMER 2 PIN MAPPING

DEVIOMAP.2 Bit Value	T2IN mapping	T2EX mapping	T2OUT mapping
0 (Reset)	P1.0	P1.1	P4.4
1	P6.1**	P6.0**	P1.2

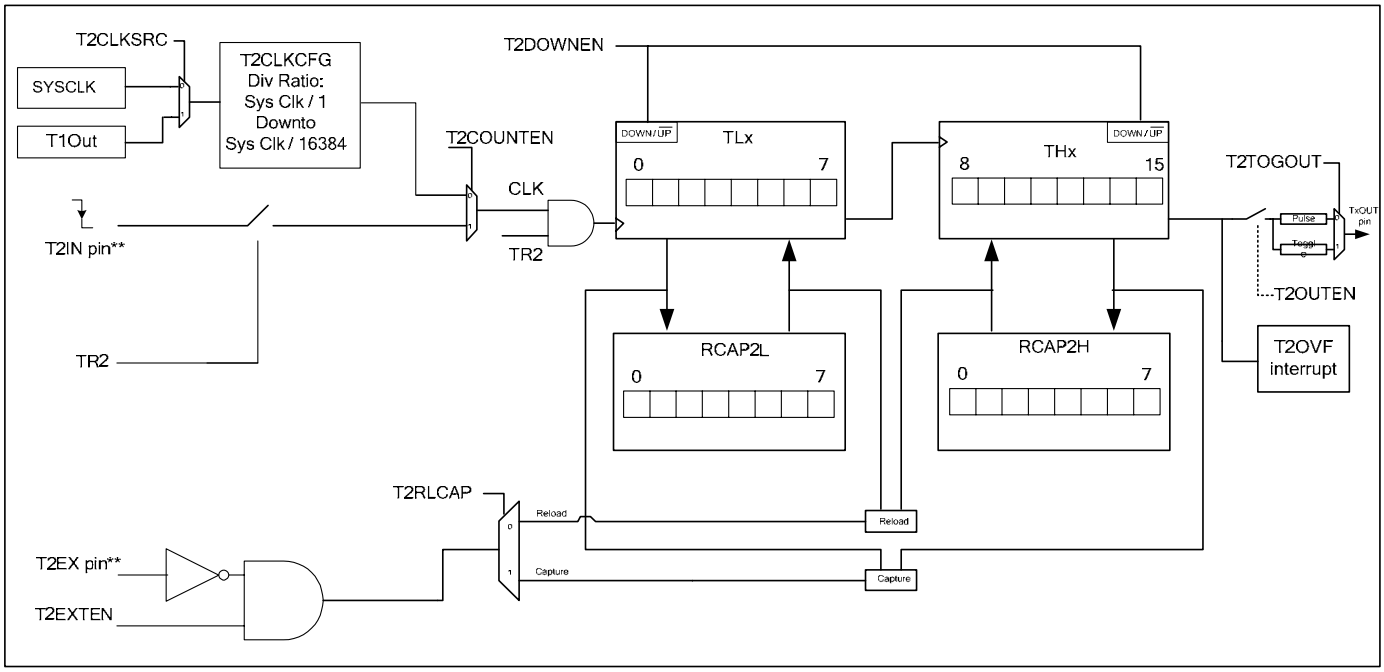
**Only on VRS51L30xx (QFP-64) devices.

Alternate mapping allows Timer 2's output to be mapped into P1.2 instead of P4.4 This can be useful for applications where both UART0 and UART1 are required.

7.16 Timer 2 Functional Diagram

The following diagram describes the main features of Timer 2.

FIGURE 19: TIMER 2 FUNCTIONAL DIAGRAM



**The T2IN alternate and T2EX alternate input are only accessible on VRS51L30xx (QFP-64) devices

7.17 Timers Chaining

The three timers of the VRS51L3xxx devices can be chained together to form a 24-, 32- or 48-bit timer that can be used for very long delay timing. Longer delays can be achieved by using the system clock prescalers.

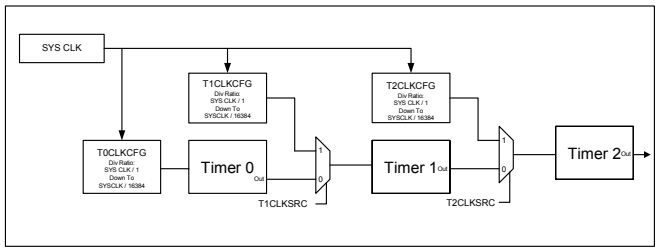
The following provides an example of time delays that can be achieved by timer chaining:

TABLE 81: TIME DELAYS VS. TIMER SIZE FOR 40MHZ SYSTEM CLOCK

Timer Size	Time out period
16 bit	1.638 milliseconds
24 bit	419 milliseconds
32 bit	107 sec-seconds
48 bit	7.037x10E6 seconds (1954.6 hours)

The following diagram provides a schematic representation of timer chaining.

FIGURE 20: TIMER CHAINING



Note that timer chaining does not affect other timer features such as:

- Timer capture
- Timer auto-reload
- Timer output

It is also possible to couple the timer chaining capability with the pulse width counter (see next section), to count long duration events.

8 Pulse Width Counters (PWC)

The VRS51L3xxx devices provide two independent pulse width counter modules associated with timers 0 and 1. The pulse width counter modules provide advanced timer control, the user to define which event will make the timer start and stop. Contrary to standard timer capture module units, the PWC unit can be used to measure the duration of an event.

The following two diagrams provide a schematic view of the PWC modules' structure and functionality.

FIGURE 21: PWC0 MODULE STRUCTURE

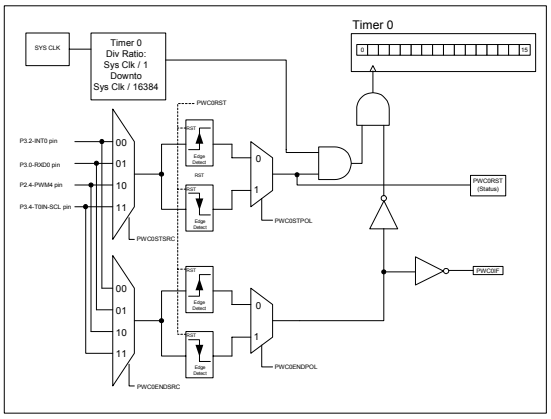
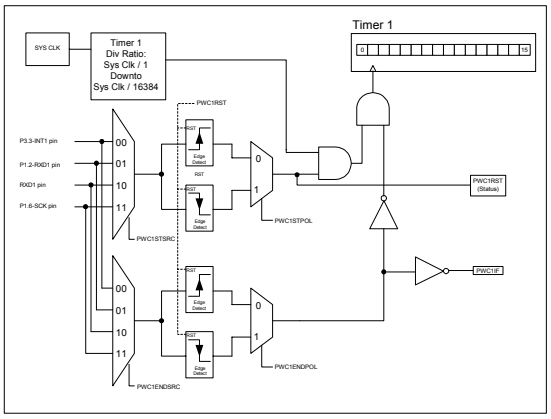


FIGURE 22: PWC1 MODULE STRUCTURE



Note: The PC1.2 input is not accessible on the VRS51L31xx (QFP-44) devices

The PWC modules interact with timers 0 and 1. Combining the PWC module configuration with the timer configuration provides added flexibility to the operating modes.

Two SFR registers (PWC0CFG and PWC1CFG located at addresses 9Eh and 9Fh, respectively) are dedicated to PWC configuration.

TABLE 82:PULSE WIDTH COUNTER 0 CONFIG. REGISTER - PWC0CFG SFR 9EH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PWC0IF	Pulse Width Counter Module 0 Interrupt Flag 0 = No PWC0 interrupt occurred 1 = PWC0 interrupt occurred
6	PWC0RST	**Read:** Pulse Width Counter Operation Status 0 = PWC0 is waiting for start condition 1= PWC0 is currently counting
		Write: Pulse Width Counter Reset 0 = No action 1 = Reset PWC0 operation and PWC0IF PWC0 will wait for a start condition
5	PWC0ENDPOL	PWC0 End Event Polarity 0 = PWC0 end event is a rising edge 1 = PWC0 end event is a falling edge
4	PWC0STPOL	PWC0 Start Event Polarity 0 = PWC0 start event is a rising edge 1 = PWC0 start event is a falling edge
3:2	PWC0ENDSRC [1:0]	PWC0 End Source 00 = P3.2 01 = P3.0 10 = P2.4 11 = P3.4
1:0	PWC0STSRC [1:0]	PWC0 Start Source 00 = P3.2 01 = P3.0 10 = P2.4 11 = P3.4

TABLE 83:PULSE WIDTH COUNTER 1 CONFIG. REGISTER - PWC1CFG SFR 9FH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PWC1IF	Pulse Width Counter Module 0 Interrupt Flag 0 = No PWC1 interrupt occurred 1 = PWC1 interrupt occurred
6	PWC1RST	**Read:** Pulse Width Counter Operation Status 0 = PWC1 is waiting for start condition 1= PWC1 is currently counting
		Write: Pulse Width Counter Reset 0 = No action 1 = Reset PWC1 operation and PWC0IF PWC0 will wait for a start condition
5	PWC1ENDPOL	PWC1 END Event Polarity 0 = PWC1 end event is a rising edge 1 = PWC1 end event is a falling edge
4	PWC1STPOL	PWC1 Start Event Polarity 0 = PWC1 start event is a rising edge 1 = PWC1 start event is a falling edge
3:2	PWC1ENDSRC [1:0]	PWC1 End Source 00 = P3.3 01 = P1.2 10 = RXD1 (alternate) ** 11 = P1.6
1:0	PWC1STSRC [1:0]	PWC1 Start Source 00 = P3.3 01 = P1.2 10 = RXD1 (alternate) ** 11 = P1.6

** Accessible on VRS51L30xx (QFP-64) devices only

The configuration of the PWC module involves the following steps:

- o Activate the PWC module
- o Activate timer and configure in gating mode
- o Configure PWC start and stop source
- o Configure PWC start and stop event
- o Initialize timer to 0x0002
- o Activate PWC interrupt, if required

8.1 PWC Module Initialisation

The PWC0/1 modules operate in conjunction with timers 0/1. The timer must be activated and configured in gating mode immediately after the PWC modules have been enabled. To obtain a precise measurement of the event duration, the timer registers [THx,TLx] must be initialized to 00, 02h.

Once a stop event occurs, the event duration in terms of system cycles is stored in the timer registers. Once the timer has been read, the software must clear it for the next event.

```
// PWC0 Timer initialization

   ptr = (char idata *) &result_dump_start_address_pwc0;

   PERIPHEN2  |= 0x40;        //Enable pwc0 (enabled first to gate timer
                              //before timer enable !!!)
   PERIPHEN1  |= 0x01;        //Enable Timer 0
   T0T1CFG = 0x02;            //Set Timer 0 in gate mode
   TL0 = 0x02;                /Initialize Timer
   TH0 = 0x00;

   PWC0CFG |= 0x15;           //Configure PWC0 module to start on a Falling edge and
                              //End on a Rising edge on pin P3.0 for both events
```

The timer start source can differ from the timer stop source and the start event can differ from the end event. The PWC start and end sources are defined by the PWCxSTSRC bits of the PWCxCFG register as shown in the following tables:

TABLE 84:PULSE WIDTH COUNTER 0 START / STOP SOUCE CONFIGURATION

PWC0STSRC	PWC0 Start Source	PWC0ENDSRC	PWC0 End Source
00	P3.2 – INT0	00	P3.2 – INT0
01	P3.0 – RXD0 default	01	P3.0 – RXD0 default
10	P2.4 – RXD0 alternate	10	P2.4 – RXD0 alternate
11	P3.4 – T0IN	11	P3.4 – T0IN

TABLE 85:PULSE WIDTH COUNTER 1 START / STOP SOUCE CONFIGURATION

PWC1STSRC	PWC1 Start Source	PWC1ENDSRC	PWC1 End Source
00	P3.3 – INT1	00	P3.3 – INT1
01	P1.2 – RXD1 default	01	P1.2 – RXD1 default
10	RXD1 alternate **	10	RXD1 alternate **
11	P1.6	11	P1.6

** Accessible on VRS51L30xx (QFP-64) devices only

Start and stop events must be triggered by either a rising edge or a falling edge of the selected start and stop source.

The PWC start source polarity is defined by the PWCxSTPOL and the stop source polarity is defined by the PWxCENDPOL. When these bits are cleared, the PWC module will be triggered by a rising edge (low to high). Setting these bits to 1 configures the PWC to be triggered by a falling edge (high to low).

8.2 PWC Module Reset and Interrupt Flags

The PWCxRST bit, when set to 1 will reset the PWC module and clear the PWCxIF flag if it is set. The PWC module will then wait for the start condition. The PWCxRST flag provides the current state of the PWC module as follows:

TABLE 86: DEFINITION OF PWCXRST BIT WHEN READ

PWCxRST reads as	Then…
0	PWC module is waiting for a start condition
1	PWC module is currently counting

The PWCxIF bit will be set to 1 when a stop condition is encountered by the PWC module. The PWCxIF must be cleared by the program. One interrupt vector (Int 11) is allocated for the two PWC modules and its vector address is 005Bh.

Note:

- The PWCxIF flag remains active even if the corresponding PWC interrupt is disabled.
- The PWCxIF flags are not automatically cleared when exiting the interrupt service routine. They must be cleared manually by the software.

8.3 PWC Example Program

Code Example:

The following example program demonstrates how to configure and use the PWC1 module in pooling mode.

```
//-------------------------------------------------------------------------------------------------------------//
// V2K_PWC1p1in_T2out_SDCC.c       //
//-------------------------------------------------------//
//
// DESCRIPTION:
For this demonstration program Timer 2 is configured to
//      continuously run in Output Toggle Mode on its alternate output (P1.2) and
//      is used to generate the stimuli required for the PWC1 module input.
//              The port 0 is used to monitor the activity of the PWC1 module.
//
// TARGET:   VRS51L2xxx/VRS51L3xxx
/-------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

void main (void) {

   //Enable Timer 0 and Timer 2

   //--Initialize PWC1
   PERIPHEN2 |= 0x088;          //Enable the PWC1 module & IOport
   PERIPHEN1 |= 0x02;           //Enable Timer 1
   P0PINCFG = 0x00;             //P0 = Output

   T0T1CFG |= 0x40;             //Set Timer 1 in Gating mode
   TH1 = 0x00;                  //Initialize Timer 1 to 0x02
   TL1 = 0x02;
//  T1CON |= 0x04;               //Run Timer 1

   //Configure Timer 2 as a Timer with output toggle
   PERIPHEN1 |= 0x04;           // Timer 2
   TH2 = 0xA0;                  //Config Timer 2 initial value
   TL2 = 0x00;
   RCAP2H = 0xA0;               //Config Timer 2 Reload value
   RCAP2L = 0x00;
   //Configure Timer Clock source & output Enable
   T2CLKCFG = 0x10;             //T2 Clk source = System Clock
                                //T2 Output Enable
                                //Prescaler = Fosc / 1
   //Configure Timer 2 Alternate output
   DEVIOMAP |= 0x04;

   //Config T2 output toggle and Start Timer
   T2CON = 0x14;                //Timer 2 output toggle
                                //Timer 2 Run
                                //Timer mode from Sys Clk

   //Configure PWC1 to Start T1 on a rising edge & Stop T1 on a falling edge
   //(will measure T2 period)
   PWC1CFG = 0x65;              //Bit 6 = 1: Reset PWC (bit 6 = 1)
                                //Bit 5 = 1: Start on Rising Edge
                                //Bit 4 = 0  Stop on rising edge
                                //Bit 3:2 = 01 PWC1 START / STOP input = P1.2- T2out*

   //Infinite loop of PWC1 module monitoring by pooling
   //The P0 is used to monitor the activity of the PWC1 module
   //When the PWC1 Start condition is met, the program set P0 to 0x00
   //and return it to 0xFF when the Stop condition occurs

   P0 = 0xFF;                   //Set P0 to 0xFF (PWC not running)

   do{
   //PWC1CFG |= 0x40;           //Force the PWC1 module to wait for a START condition

   while(!(PWC1CFG&0x40));      //wait PWC to start
   P0 = 0x00;                   //clear P0
   while(!(PWC1CFG&0x80));      //wait PWC stop condition to occurs ie interrupt found
   P0 = 0xFF;                   //return P0 to FF to indicate PWC stopped
   PWC1CFG &= 0x7F;
   TL1 = 0x02;                  //Initialize Timer 1 to 0x02
   TH1 = 0x00;
   }while(1);

}// End of main
```

9 UART Serial Ports

The serial ports on the VRS51L3xxx devices operate in full duplex mode. However, the communication speed will be the same for transmission and reception. Communication speed is derived from an internal 16-bit baud rate generator dedicated to each of the UARTs.

9.1 UART0 RX / TX Data Buffer

The serial port features double buffering on the receiving side. The SFR register, UART0BUF, provides access to the transmit and receive registers of the serial port.

When a read operation is performed on the UART0BUF register, it will access the receive register double buffer. When a write operation is performed on the UART0BUF, the transmit register will be loaded with the value to be transmitted.

TABLE 87:UART0 DATA RX / TX REGISTER UART0BUF SFR A3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	UART0BUF[7:0]	Read: UART0 Receive Buffer Write: UART0 Transmit Buffer

9.2 UART0 Configuration Registers

The configuration of the UART0 is controlled by the UART0CFG, the UART0BRH and UART0BLH registers and the UART0EXT registers.

TABLE 88:UART0 CONFIGURATION REGISTER - UART0CFG SFR A2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	0	0	0	0	0

Bit	Mnemonic	Description
7:4	BRADJ[3:0]	UART0 Baud Rate Fine Adjustment * see formula below
3	BRCLKSRC	Baud Rate Clock Source 0 = Baud rate generator uses oscillator 1 = Baud rate generator uses external clock source
2	B9RXTX	Read: Last received 9th bit Write: 9th bit to transmit
1	B9EN	9th Bit Mode Enable 0 = Data transfer are in 8-bit format 1 = Data transfer are in 9-bit format
0	STOP2EN	Enable Two Stop Bit Mode 0 = One stop bit 1 = Two stop bit

TABLE 89:UART0 BAUD RATE REGISTER LOW – UART0BRL SFR A4H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	UART0BRL[7:0]	UART0 LSB of Baud Rate Generator

TABLE 90:UART0 BAUD RATE REGISTER HIGH – UART0BRH SFR A5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	UART0BRH[7:0]	UART0 MSB of Baud Rate Generator

TABLE 91:UART0 EXTENSIONS CONFIGURATION - UART0EXT SFR A6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	1	0	0	0	0	0

Bit	Mnemonic	Description
7	U0TIMERF	UART0 Timer Flag
6	U0TIMEREN	UART0 Timer Enable
5	U0RXSTATE	UART0 RX Line State
4	MULTIPROC	When set, RX_available only raise if the ninth received bit is '1'
3	Reserved	Leave as 0000

9.3 UART0 Interrupt Configuration

The activation of the UART0 interrupt is a two-stage process that involves enabling the interrupts at the UART0 module level and then activating the UART0 interrupt at the system level through the INTEN1 register. The upper nibble of the UART0INT register contains the UART0 interrupt activation bits and the lower nibble contains the UART0 interrupt flags in the same order.

Two interrupt vectors are associated with UART0. The first interrupt vector is at address 002Bh and handles all UART0 interrupt conditions, except for the UART0 data collision interrupt (vector address 0053h), which is shared with the UART1 data collision and the I²C master lost arbitration interrupts.

The interrupt flags allow the interrupt service routine to define which condition triggered the interrupt, and to react accordingly. Note that the interrupt flags do not require the interrupt to be enabled in order to be operational. They can be monitored by the software at any time.

TABLE 92: UART0 INTERRUPT REGISTER - UART0INT SFR A1H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R, W	R/W	R, W	R
0	0	0	0	0	0	0	1

Bit	Mnemonic	Description
7	COLEN	UART0 Collision Interrupt Enable 0 = Collision interrupt is deactivated 1 = Collision interrupt is enabled
6	RXOVEN	UART0 RX Overrun Interrupt Enable 0 = RX Overrun interrupt is deactivated 1 = RX Overrun interrupt is enabled
5	RXAVAILEN	UART0 RX Available Interrupt Enable 0 = RX Available interrupt is deactivated 1 = RX Available interrupt is enabled
4	TXEMPTYEN	UART0 TX Empty Interrupt Enable 0 = TX Empty interrupt is deactivated 1 = TX Empty interrupt is enabled
3	COLENF	(Read) Collision Interrupt Flag When this flag is set by the UART0 module, it indicates that a collision occurred
		(Write) 0 = Collision detection is disabled and the collision COLENF is reset 1 = A bus collision stops the transmission and raises the COLENF flag
2	RXOVF	UART0 RX Overrun Flag When set to 1 by the UART0 interface, it indicates that a data collision occurred in the UART0BUF register
1	RXAVENF	R: UART0 RX Available Flag When set to 1 by the UART0 interface, it indicates that data has been received in the UART0BUF register. Will be automatically cleared when UART0BUF is read W: UART RX enable Writing 1 into this bit position will activate reception on UART0. Clearing this bit 0 will deactivate the reception on UART0
0	TXEMPTYF	UART0 TX Empty Flag When set to 1, it indicates that the transmit portion of the UART0BUF is ready to receive another byte

9.4 UART1 RX/TX Data Buffer

The SFR register (UART1BUF) provides access to the transmit and receive registers of the serial port. When a read operation is performed on the UART1BUF register, it will access the receive register. When a write operation is performed on the UART1SBUF, the transmit register will be loaded with the value to be transmitted.

TABLE 93:UART1 DATA RX / TX REGISTER UART1BUF SFR B3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	UART1BUF[7:0]	Read: UART1 Receive Buffer Write: UART1 Transmit Buffer

9.5 UART1 Configuration Registers

The configuration of the UART1 is controlled by the UART1CFG, UART1BRH and UART1BLH registers and the UART1EXT registers.

TABLE 94:UART1 CONFIGURATION REGISTER - UART1CFG SFR B2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	0	0	0	0	0

Bit	Mnemonic	Description
7:4	BRADJ[3:0]	UART1 Baud Rate Fine Adjustment * see formula below
3	BRCLKSRC	Baud Rate Clock Source 0 = Baud rate generator uses oscillator 1 = Baud rate generator uses external clock source
2	B9RXTX	Read: Last received 9th bit Write: 9th bit to transmit
1	B9EN	9th Bit Mode Enable 0 = Data transfer are in 8-bit format 1 = Data Transfer are in 9-bit format
0	STOP2EN	Enable Two Stop Bit Mode 0 = One stop bit 1 = Two stop bit

TABLE 95:UART1 BAUD RATE REGISTER LOW – UART1BRL SFR B4H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	UART1BRL[7:0]	UART1 LSB of Baud Rate Generator

TABLE 96:UART1 BAUD RATE REGISTER HIGH – UART1BRH SFR B5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	UART1BRH[7:0]	UART1 MSB of Baud Rate Generator

TABLE 97:UART1 EXTENSIONS CONFIGURATION - UART1EXT SFR B6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	1	0	0	0	0	0

Bit	Mnemonic	Description
7	U1TIMERF	UART1 Timer Flag
6	U1TIMEREN	UART1 Timer Enable
5	U1RXSTATE	UART1 RX Line State
4	MULTIPROC	When set, RX_available, only raise if the ninth received bit is '1'
3:0	Reserved	Reserved. Leave these bit to 0

9.6 UART1 Interrupt Configuration

The activation of UART1's interrupt is a two stage process that involves enabling the interrupts at the UART1 module level and then activating the UART1 interrupt at the system level through the INTEN1 register.

TABLE 98: UART1 INTERRUPT REGISTER - UART1INT SFR B1H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R, W	R/W	R, W	R
0	0	0	0	0	0	0	1

Bit	Mnemonic	Description
7	COLEN	UART1 Collision Interrupt Enable 0 = Collision interrupt is deactivated 1 = Collision interrupt is enabled
6	RXOVEN	UART1 RX Overrun Interrupt Enable 0 = RX Overrun interrupt is deactivated 1 = RX Overrun interrupt is enabled
5	RXAVAILEN	UART1 RX Available Interrupt Enable 0 = RX Available interrupt is deactivated 1 = RX Available interrupt is enabled
4	TXEMPTYEN	UART1 TX Empty Interrupt Enable 0 = TX Empty interrupt is deactivated 1 = TX Empty interrupt is enabled
3	COLENF	(Read) Collision Interrupt Flag When this flag is set by the UART1 module, it indicates that a collision has occurred
		(Write) 0 = Collision detection is disabled and the collision COLENF is reset 1 = A bus collision stops the transmission and raises the COLENF flag
2	RXOVF	UART1 RX Overrun Flag When set to 1 by the UART1 interface, it indicates that a data collision has occurred in the UART0BUF register
1	RXAVENF	R: UART1 RX Available Flag When set to 1 by the UART0 interface, it indicates that data has been received in the UART1BUF register. Will be automatically cleared when UART1BUF is read W: UART RX enable Writing 1 into this bit position will activate reception on UART1. Clearing this bit 0 will deactivate the reception on UART0
0	TXEMPTYF	UART1 TX Empty Flag When set to 1, it indicates that the transmit portion of the UART1BUF is ready to receive another byte

9.7 UART0, UART1 Baud Rate Formula

The UART0 baud rate is programmed using the following formula:

$$\text{Baud Rate} = \frac{\text{Fclk}}{32\text{x (UARTxBR[15:0] + BRADJ[3:0]/16 + 1)}}$$

The BRADJ[3:0] bits are used for fine adjustment of the baud rate.

The following steps demonstrate using the UARTxBR[15:0] and BRADJ[3:0] registers to set the appropriate baud rate.

Step 1: Defining the Optimal UARTxBR[15:0] Value

Use the following formula to set the UARTxBR[15:0] register to the integer component of UARTxBRideal:

$$\text{UARTxBR}_{\text{ideal}} = \frac{\text{Fclk}}{32\text{x (Baud Rate)}} - 1$$

Note that the baud rate will likely contain a fractional component.

Valid UARTxBR[15:0] values range from 0x0000 to 0xFFFF.

Step 2: Defining the Optimal BRADJ[3:0] Value

Use the following formula to set the BRADJ[3:0]:

$$\text{BRADJ[3:0]} = \text{ROUND[(UARTxBRideal – UARTxBR[15:0]) * 16]}$$

The BRADJ[3:0] register can only contain an integer value between 0x00 and 0x0F.

Step 3: Calculating the Error

The actual baud rate vs. the ideal baud rate can be calculated using the following formula:

$$\text{Error \%} = \frac{100\text{x [(Fclk /32*(UARTxBR[15:0]+BRADJ[3:0]/16 +1))-Baud Rate]}}{\text{Baud Rate}}$$

In order to achieve reliable communication, the error should be below 2 percent.

The following table provides configuration examples for typical baud rates when the internal 40MHz oscillator is used:

TABLE 99: UARTS BAUD RATE CONFIGURATION EXAMPLES (SYS CLK =40MHZ)

Com Speed	UARTxBR [15:0]	BRADJ [3:0]	Actual Baud Rate	Error (%)
230400bps	0004h	07h	229885.1	-0.22
115200bps	0009h	0Eh	114942.5	-0.22
57600bps	0014h	0Bh	57636.9	0.06
38400bps	001Fh	09h	38387.7	-0.03
31250bps	0027h	00h	31250.0	0
28800bps	002Ah	06h	28818.4	0.06
19200bps	0040h	02h	19193.9	-0.03
9600bps	0081h	03h	9601.5	0.01
4800bps	0103h	07h	4799.6	-0.01
2400bps	0207h	0Dh	2400.1	0
1200bps	0410h	0Bh	1200.0	0
300bps	1045h	0Bh	300.0	0

9.8 UART0, Alternate Mapping

Upon reset, UART0's RXD0 and TXD0 signals are mapped into pins P3.0 and P3.1, respectively. It is possible to re-map the RXD0 and TXD0 signals into pins P2.4 and P2.3.

Bit 3 of the DEVIOMAP register (SFR E1h) controls the mapping of the UART0 interface, as shown in the following table:

TABLE 100: UART0 RXD0 / TXD0 PIN MAPPING

DEVIOMAP.3 Bit Value	RXD0 Mapping	TXD0 mapping
0 (Reset)	P3.0	P3.1
1	P2.4	P2.3

When alternate mapping for UART0 is used, the UART0 will have priority over the PWM3 and PWM4 outputs.

9.9 UART1, Alternate Mapping

Upon reset, UART1's RXD1 and TXD1 signal are mapped into pins P1.2 and P1.3, respectively. It is possible to map UART1's RXD1 and TXD1 signals into pins 41 and 40 of the VRS51L30xx (QFP-64) devices.

Bit 4 of the DEVIOMAP register (SFR E1h) controls the mapping of the UART1 interface as shown in the following table:

TABLE 101: UART1 RXD1 / TXD1 PIN MAPPING

DEVIOMAP.3 Bit Value	RXD1 mapping	TXD1 mapping
0 (Reset)	P1.2	P1.3
1	Pin 41 **	Pin 40 **

** VRS51L30xx (QFP-64) devices only

9.10 UART Example Program

Configuration of UART0 is essentially the same as UART1

Code Example:

UART Echo and External Interrupt Configuration

```
//-------------------------------------------------------------------------------------------------------//
// V3K_UART0_Echo_RxInt_INT0_INT1_SDCC.c //
//-------------------------------------------------------------------------//
//
// This program initialise the UART0 at 115200 (Running from 40MHz internal oscillator)
// It then transmit "UART0 Echo:  Waiting for char on RXD0...or INT0\0" on TXD0.
// The UART, INT0 and INT1 are configured and the program enters in infinite loop waiting
for an interrupt
// As soon as a character is received it is transmitted back on TXD0
// If INT0 or INT1 is received the program transmit "EXT INT0 received" or EXT INT1
received"
// on TXD0 depending on which interrupt was received.
// While waiting for interrupts, the software toggle P4 every 10ms
//-------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

// --- function prototypes
void txmit0( unsigned char charact);
void uart0config(void);
void delay(unsigned int );

code char msg[] = "UART0 Echo:  Waiting for char on RXD0...or INT0 / INT1...\0";
code char msgint0[] = "EXT INT0 received";
code char msgint1[] = "EXT INT1 received";

//-------------------------------------------------------------------//
//                  MAIN FUNCTION
//-------------------------------------------------------------------//
 void main (void){
    int  cptr= 0x00;              //general purpose counter

    PERIPHEN1 = 0x08;             //Enable UART0
    PERIPHEN2 = 0x08;             //Enable IO Ports
    P4PINCFG = 0x00;              //Configure P4 as output

    uart0config();                //Configure Uart0

    //-- Send "UART0 Echo:  Waiting for char on RXD0...\0" on UART0
    do{
       txmit0(msg[cptr++]);
       }while(msg[cptr]!= '\0');

       txmit0(13);                //Send Carriage Return
       txmit0(10);                //Send Line Feed

    //--Wait for Character on UART0 interrupt
    // Once a character is received, grab it and send it back

    GENINTEN = 0x02;              //Set the PININTCFG bit before configuring
                                  //the INT0 pin event.  This will prevent receiving
                                  //an inadvertent INT0 interrupt to be triggered
                                  //at the moment INT0 triggering event is
                                  //configured as Rising edge

    INTSRC1 = 0x03;               //INT0 vector source = INT0, INT1 vector source = INT1
pin
    IPINSENS1 = 0x03;             //Set INT0, INT1 sensitive on edge(1) or Level(0)
    IPININV1 = 0x00;              //Set INT0, INT1 Pin sensitivity on Low Level/Inversion
    INTEN1 = 0x23;                //Enable INT0 (bit0), INT1 (bit1) and UART0 (bit5) Interrupt

    GENINTEN = 0x01;              //Enable Global interrupt

    do{
    P4 = ~P4;
    delay(10);
    }while(1);

}//end of Main

//-----------------------------------//
//----- Interrupt INT0 ------//
//-----------------------------------//
void INT0Interrupt(void) interrupt 0
    {
    //-- Send "EXT INT0 Received" on UART0
    char cptr = 0x00;             // Init cptr to pint to message beginning
    INTEN1 = 0x00;                //Disable Interrupt

    do{
       B = msgint0[cptr++];
       txmit0(B);
```

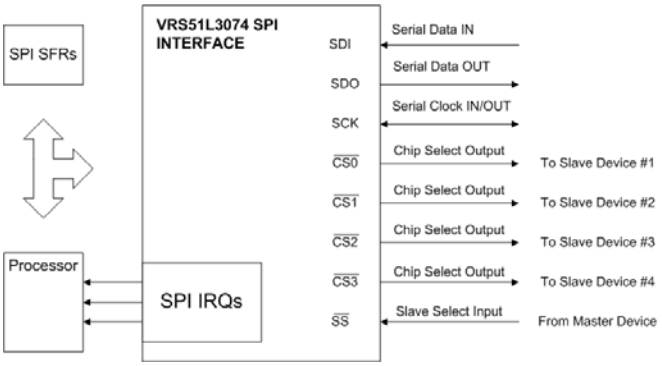
```
    }while(msgint0[cptr]!= '\0');

    txmit0(13);                 //Send Carriage Return
    txmit0(10);                 //Send Line Feed

  INTEN1 = 0x23;                //Enable UART0 Interrupt + INT0 + INT1
  }//end of INT0 interrupt

//---------------------------------//
//----- Interrupt INT1 ------//
//---------------------------------//
void INT1Interrupt(void) interrupt 1
  {
  char cptr = 0x00;             // Init cptr to pint to message beginning

  INTEN1 = 0x00;                //Disable Interrupt
  //-- Send "EXT INT1 Received" on UART0
  do{
    B = msgint1[cptr++];
    txmit0(B);
    }while(msgint1[cptr]!= '\0');

    txmit0(13);                 //Send Carriage Return
    txmit0(10);                 //Send Line Feed

  INTEN1 = 0x23;                //Enable UART0 Interrupt + INT0 + INT1
  }//end of INT0 interrupt

//-------------------------------------//
//--- UART0 Interrupt --------//
//-------------------------------------//
void UART0Interrupt(void) interrupt 5
{
char genvar;

  INTEN1 = 0x00;                //Disable UART0 Interrupt

  //Check if interrupt was caused by RX AVAIL
  genvar = UART0INT;

  if(genvar & 0x02)
    txmit0(UART0BUF);           //Send back the received character

//Check if interrupt was caused by RX OVERRUN

  if(genvar &= 0x04)
    {
    genvar = UART0BUF;          //Read S0BUF to clear RX OV condition...
                                //***This is mandatory because otherwise
                                //the RX OV condition keep
                                //interrupt activated
    txmit0(' ');                //Send " OV!" on serial port
    txmit0('O');                //
    txmit0('V');                //
    txmit0('!');                //
    }if(genvar &= 0x04)


  INTEN1 = 0x23;                //Enable UART0 Interrupt + INT0 + INT1

}//end of uart0 interrupt

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

//-------------------------------------------------------------------------------------//
// UART0 CONFIG
//
// Configure the UART0 to operate in RS232 mode at 115200bps.
// VRS51L2070 running from the internal 40MHz oscillator
//
//-------------------------------------------------------------------------------------//
void uart0config()
  {
//--initialize UART0 at 115200bps @ 40MHz

  UART0CFG = 0xE0;              //Fine adjustment on baud rate
                                //Use internal clock
                                //9th bit not used
                                //only one stop bit

  UART0INT = 0x62;              //Enable RX AV + RXO V int + Enable Reception
  UART0EXT = 0x00;              //Not using UART0 Extensions

  UART0BRL = 0x09;              //Reload value for 115200
  UART0BRH = 0x00;              //
  }//end of uart0ws0relcfg() function

//-------------------------------------------------------------------------------------//
// TXMIT0
//
// Transmit one byte on the UART0
//-------------------------------------------------------------------------------------//
void txmit0( unsigned char charact){

  UART0BUF = charact;           //Send Character
  while(!(UART0INT & 0x01));

  }//end of txmit0() function


//;---------------------------------------------------------------------
//;- DELAY1MSTO : 1MS DELAY USING TIMER0
//;---------------------------------------------------------------------
void delay(unsigned int dlais){

idata unsigned char x=0;
idata unsigned int dlaisloop;

  PERIPHEN1 |= 0x01;            //ENABLE TIMER 0

  dlaisloop = dlais;
  while ( dlaisloop > 0)
    {
    TH0 = 0x63;                 //TIMER0 RELOAD VALUE FOR 1MS AT 40MHZ
    TL0 = 0xC0;

    T0T1CLKCFG = 0x00;          //NO PRESCALER FOR TIMER 0 CLOCK
    T0CON = 0x04;               //START TIMER 0,  COUNT UP

    do{
      T0CON &= 0x80;

      }while(x==0);

    T0CON = 0x00;             //Stop Timer 0
    dlaisloop = dlaisloop-1;

    }//end of while dlaisloop...
    PERIPHEN1 &= 0xFE;          //DISABLE TIMER 0

  }//End of function delay
```

10 SPI Interface

The SPI interface of the VRS51L3xxx devices provide numerous enhancements compared to other vendor offerings. The SPI interface's key features include:

- Supports four standard SPI modes (clock phase/polarity)
- Operates in master and slave modes
- Automatic control of up to four chip select lines
- Configurable transaction size (1 to 32 bits)
- Transaction size of >32 bits is possible
- Double Rx and TX data buffers
- Configurable MSB or LSB first transaction
- Generation frame select/load signals

FIGURE 23: SPI INTERFACE OVERVIEW



Before the SPI can be accessed it must first be enabled by setting the SPIEN bit of the PERIPHEN1 register to 1.

10.1 SPI Control Registers

The SPICTRL register controls the operating modes of the SPI interface in master mode.

TABLE 102:SPI CONTROL REGISTER - SPICTRL SFR C1H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

Bit	Mnemonic	Description
7:5	SPICLK[2:0]	SPI Communication Speed (Master Mode) 000 = Sys Clk / 2 (/ 8 if SPISLOW = 1) 001 = Sys Clk / 4 (/ 16 if SPISLOW = 1) 010 = Sys Clk / 8 (/ 32 if SPISLOW = 1) 011 = Sys Clk / 16 (/ 64 if SPISLOW = 1) 100 = Sys Clk / 32 (/ 128 if SPISLOW = 1) 101 = Sys Clk / 64 (/ 256 if SPISLOW = 1) 110 = Sys Clk / 128 (/ 512 if SPISLOW = 1) 111 = Sys Clk / 256 (/ 1024 if SPISLOW = 1)
4:3	SPICS[1:0]	SPI Active Chip Select Line (Master Mode) 00 = CS0 is active 01 = CS1 is active 10 = CS2 is active 11 = CS3 is active
2	SPICLKPH	SPI Clock Phase 0 = SD0 output on rising edge and SDI sampling on falling edge 1= SD0 output on falling edge and SDI sampling on rising edge
1	SPICLKPOL	SPI Clock Polarity 0 = SCK stays at 0 when SPI is inactive 1 = SCK stays at 1 when SPI is inactive
0	SPIMASTER	SPI Master Mode Enable 0 = SPI operates in slave mode 1 = SPI operate in master mode (default)

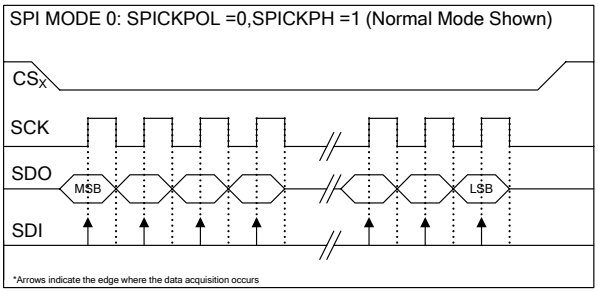
When the SPIMASTER bit is set to 1, the SPI interface operates in master mode. This is the default operating mode of the SPI interface after reset.

10.2 Setting Up Clock Phase and Polarity

The clock phase and polarity is controlled by the SPICLKPH and SPICLKPOL bits, respectively. The following diagrams show the communication timing associated with the clock phase and polarity.

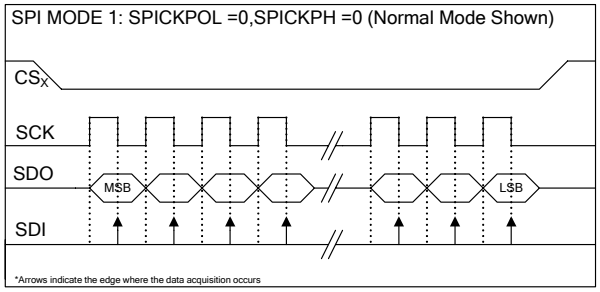
SPI Mode 0:

FIGURE 24: SPI MODE 0



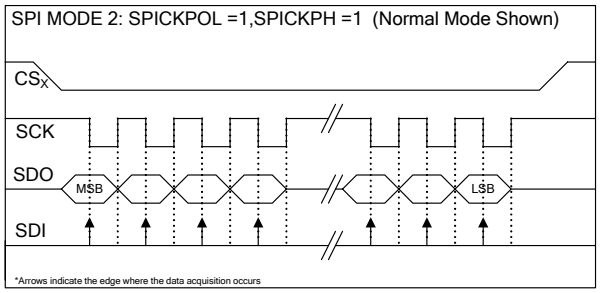
SPI Mode 1:

FIGURE 25: SPI MODE 1



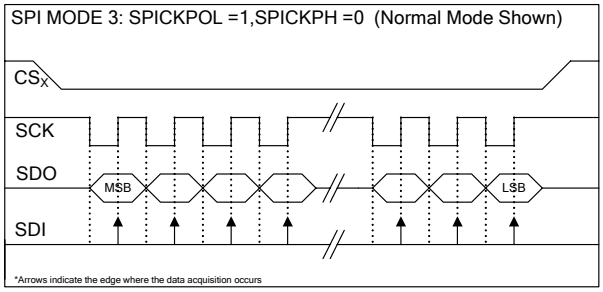
SPI Mode 2:

FIGURE 26: SPI MODE 2



SPI Mode 3:

FIGURE 27: SPI MODE 3



10.3 Defining active chip select line

As previously mentioned, only one chip select line is activated when communicating with an external SPI slave device. The SPICS bits of the SPICTRL register are used to select which CS line will be activated during the transfer.

Note that with the exception of the CS0 line, the SPICSEN bit of the PERIPHEN1 register must be set to 1 in order for the SPI be able to control the SPI CS lines.

10.4 Setting the SPI Communication Speed (Master Mode)

In master mode, the SPI interface communication speed is adjustable from “system clock /2” down to “system clock / 1024”. Slower communication speeds can be useful for interfacing with slower devices or to adjust the communication speed to specific bus conditions.

The SPICLK[2:0] of the SPICTRL and the SPISLOW bit of the of the SPICONFIG SFR register control the SPI communication speed.

The SPI communication speed in master mode can be calculated using the following formula:

$$\text{SPI speed} = \frac{\text{Sys Clk}}{[\ 2^{(\text{SPICLK[2:0]}+1)} \times 4^{\text{SPISLOW}}\ ]}$$

Where:

- Sys Clk = Processor operating clock
- SPISLOW = can be either 0 or 1
- SPICLK[2:0] = from 0 to 7

The following tables provide example setting for SPI communication speeds with various system clock and SPICLK[2:0] and SPISLOW bit settings.

TABLE 103:SPI COMMUNICATION SPEED EXAMPLE (SPISLOW = 0)

SPICLK	Com Speed @ 40MHz	Com Speed @ 22.18MHz	Com Speed @ 4MHz
000	20 MHz	11.05 MHz	2 MHz
001	10 MHz	5.53 MHz	1 MHz
010	5 MHz	2.76 MHz	500 kHz
011	2.5 MHz	1.38 MHz	250 kHz
100	1.25 MHz	691.2 kHz	125 kHz
101	625 kHz	345.6 kHz	62.5 kHz
110	312.5 kHz	172.8 kHz	31.3 kHz
111	156.3 kHz	86.4 kHz	15.6 kHz

TABLE 104:SPI COMMUNICATION SPEED EXAMPLE (SPISLOW = 1)

SPICLK	Com Speed @ 40MHz	Com Speed @ 22.18MHz	Com Speed @ 4MHz
000	5 MHz	2.76 MHz	500 kHz
001	2.50 MHz	1.38 MHz	250 kHz
010	1.25 MHz	691.2 kHz	125 kHz
011	625 kHz	345.6 kHz	62.5 kHz
100	312.5 kHz	172.8 kHz	31.3 kHz
101	156.3 kHz	86.4 kHz	15.6 kHz
110	78.1 kHz	43.2 kHz	7.8 kHz
111	39.1 kHz	21.6 kHz	3.9 kHz

10.5 SPI Configuration /Status Registers

The SPI configuration and status registers allow the activation and the monitoring of the SPI interface interrupts. They also provide access to the advanced features of the SPI interface such as:

- o Frame select/load generation on CS3
- o Activating manual control of the chip select lines
- o Bit reversed mode (Bitwise Endian Control)
- o Interrupt activation and monitoring
- o Monitoring the state of the SS pin

TABLE 105:SPI CONFIGURATION REGISTER - SPICONFIG - C2H

7	6	5	4	3	2	1	0
R/W	W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	SPIMANCS	SPI Manual CS Mode Enable 0 = SPI Chip select control is fully automatic 1 = SPI Chip select will be brought low by the SPI interface, and will stay low until 0 is written into SPIMANCS bit
6	SPIUNDERC	SPI Clear TX Underrun Flag (SPIUNDERF) Writing a 1 into this bit will clear the SPIUNDER bit of the SPISTATUS register This bit always reads 0
5	FSONCS3	Frame Select Pulse on CS3 0 = CS3 acts in standard ways 1 = The SPI interface will send an active low frame select pulse on CS3 Frame select has priority on SPILOAD function
4	SPILOADCS3	Load Pulse on CS3 0 = CS3 acts in standard way or as frame select pulse, if FSONCS3 is set to 1 1 = The SPI interface sends an active low load pulse on the CS3 pin, if FSONCS3 is cleared
3	SPISLOW	SPI Slow Speed mode 0 = SPI transaction occurs at normal speed 1 = SPI transaction is 4x slower
2	SPIRXOVEN	SPI RX Overrun Interrupt Enable 0 = SPI RX overrun interrupt is deactivated 1 = SPI RX overrun interrupt is enabled
1	SPIRXAVEN	SPI RX Available Interrupt Enable 0 = SPI RX available interrupt is deactivated 1 = SPI RX available interrupt is enabled
0	SPITXEEN	SPI TX Empty Interrupt Enable 0 = SPI TX empty interrupt is deactivated 1 = SPI TX empty interrupt is enabled

The SPISTATUS register's role is mainly for monitoring purposes.

TABLE 106:SPI STATUS REGISTER - SPISTATUS SFR C9H

7	6	5	4	3	2	1	0
R/W	R	R	R	R	R	R	R
0	0	0	1	1	0	0	1

Bit	Mnemonic	Description
7	SPIREVERSE	SPI Reverse Mode 0 = SPI operates in normal mode (MSB First) 1 = SPI operates in reverse mode (LSB First)
6	-	Not used
5	SPIUNDERF	SPI TX Underrun Flag 0 = No underrun condition noticed 1 = Indicates that the SPI transmit buffer has not been fed in time. This condition is likely to occur when the Transaction size is > 32 bits This bit is cleared by setting to 1, the SPIUNDEC of the SPICONFIG register.
4	SSPINVAL	Slave Select Pin Value 0 =SS pin is low 1 = SS pin is high
3	SPINOCS	SPI No Chip Select 0 = At least one chip select line is active 1 = Indicates that all the chip select lines are inactive (high)
2	SPIRXOVF	SPI RX Overrun InterruptFlag 0 = No SPI RX Overrun condition detected 1 = SPI Data collision occurred
1	SPIRXAVF	SPI RX Available Interrupt Flag 0 = SPI receive buffer is empty 1 = Data is present in the SPI RX buffer
0	SPITXEMPF	SPI TX Empty Interrupt Flag 0 = SPI transmit buffer is full 1 = SPI transmit buffer is ready to receive new data

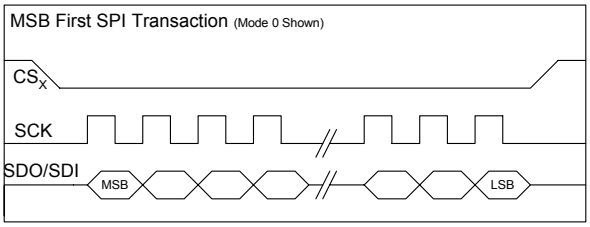
10.6 SPI Transaction Directions

The SPI interface can perform transactions in the standard SPI format (MSB first) as well as in the reverse format (LSB first). In applications where data must be transmitted (or received) in LSB first format, the user would normally need to perform bit reversal manually at the processor level and then send the data through the SPI interface. The SPI interface can automatically handle the bit reversal operations, unloading the processor for other tasks.

When the SPIREVERSE bit of the SPISTATUS register is set to 0, the SPI transactions will take place in MSB first format.

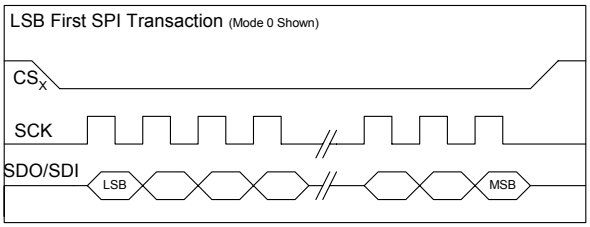
The following examples show the timing related to these transaction directions:

FIGURE 28: SPI MSB FIRST TRANSACTION



When the SPIREVERSE is set to 1, the SPI transactions are done in LSB first format, as shown in the next figure.

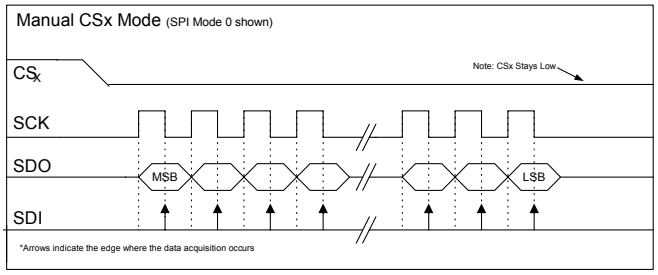
FIGURE 29: SPI LSB FIRST TRANSACTION



10.7 Manual Chip Select Control

When the SPIMANCS bit of the SPICONFIG register is set to 1, the active chip select line will stay at a logic low after the SPI master mode transaction is completed, as shown in the following figure.

FIGURE 30: SPI MANUAL CHIP SELECT



The chip select will remain at logic 0 until the SPIMANCS bit is cleared by the software.

10.8 SPI Interrupts

The SPI can trigger three interrupt sources that are handled by two interrupt vectors, as shown in the following table:

TABLE 107: SPI INTERRUPT SOURCES

Interrupt	Interrupt Number	Interrupt Vector
SPI TX Empty	Int_1	000Bh
SPI RX Available	Int_2	0013h
SPI RX Overrun		

The TX empty interrupt is set when the SPI transmit buffer is ready to receive more data. A double buffer is used in the SPI transmitter. Once transmission begins (after a write to the SPIRXTX0 register), the data is transferred to the final transmission buffer. This frees up the SPIRXTX SFR register, raises the SPITXEMPF flag of the status register and triggers an SPI TX empty interrupt if enabled. The SPI TX empty interrupt is enabled by setting the SPITXEEN bit of the SPICONFIG register to 1.

The priority of the SPI TX empty interrupt is set high in order to avoid buffer overrun in 32-bit SPI transfers.

The SPI RX available interrupt is activated when receive data has been transferred from the SPI RX buffer to the SPIRXTX register. The SPIRXTX register must be read by the processor before the next SPI bus data sequence is completed. The SPI RX available interrupt is enabled by setting the SPIRXAVEN bit of the SPICONFIG register to 1. The SPIRXAVF flag of the SPISTATUS register, when set to 1, indicates that data can be read. The SPIRXAVF flag is automatically reset when the SPIRXTX0 register is read.

The SPI RX overrun interrupt indicates that an overrun condition has taken place. The SPI RX overrun interrupt is enabled by setting the SPIRXOVEN bit of the SPICONFIG register to 1. The SPIRXOVF flag of the SPISTATUS register, when set to 1, indicates that a data collision has occurred.

All the SPI interface interrupt flags are active even if the associated interrupt is not activated and they can be monitored by the user program at any time.

Please consult the Interrupt Section for more details on the SPI interface interrupts and their interaction with other peripherals

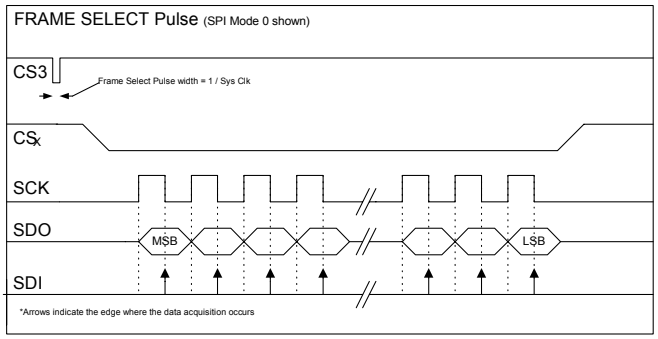
10.9 Alternate CS3 functions

For external SPI devices which require the use of a load or a frame select signal, the VRS51L3xxx can be configured to either generate an active low frame select or active high load signal when operating in master mode.

Frame Select signal on CS3

When the FONCS3 bit of the SPICONFIG register is set to 1, the SPI interface will generate an active low frame select pulse on the CS3 pin (see the following timing diagram).

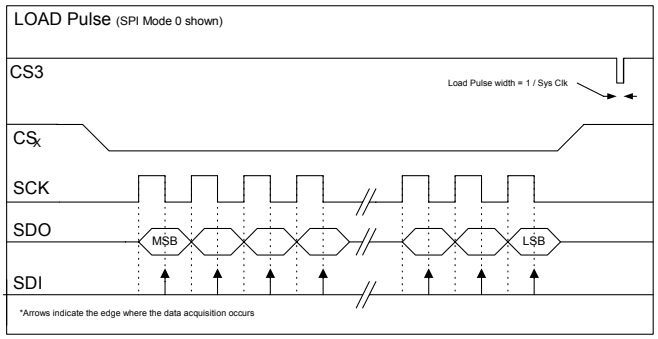
FIGURE 31: SPI FRAME SELECT PULSE TIMING



Load Signal on CS3

When the SPILOADCS3 bit of the SPICONFIG register is set to 1 *and* the FSONCS3 bit is cleared, an active low load signal will be generated on the CS3 line of the SPI interface.

FIGURE 32: SPI LOAD PULSE TIMING



Note that the frame select alternate function has priority over the load function. This means that if the FSONCS3 bit is set, the alternate function selected will be the frame select, independent of the value of the SPILOAD bit.

10.10 SPI Activity Monitoring

The ability to monitor the state of communication of the SPI interface can be useful in highly modular applications in which the SPI interface is handled by interrupts. The SPISTATUS register contains two flags that can be used to monitor the CS and SS signals of the SPI interface.

The SPINOCS bit of the SPISTATUS register returns the logical AND of all the SPI CS lines. If all the CS lines are inactive (logic high), the SPI interface sets the SPINOCS to 1. The SPINOCS bit is used to verify that the SPI interface is idle before reconfiguring it or starting a new transaction.

The SPINOCS bit of the SPISTATUS register is valid four system clock cycles after the SPI transmission begins. This delay is independent of the SPI transaction speed.

As such, after a write operation to the SPIRXTX0 register, which will trigger a SPI transaction in master mode, a NOP instruction (1 cycle) must be added before the MOV Rn, SPISTATUS instruction (3 cycles).

The SSPINVAL bit of the SPISTATUS register returns the logic level on the SS pin.

10.11 SPI TX Under Run Flag

The SPI interface provides an under run condition flag that can be used to flag whether the software has failed to update transmission buffer in time for the next transfer. This is especially useful when the SPI interface is used to transmit packets greater than 32 bits in length.

If an under run condition occurs, the SPIUNDERF bit of the SPI status register will be set to 1. This bit can be cleared by writing a 1 to the SPIUNDERC bit of the SPICONFIG register.

Note that SPI under run monitoring is not linked to any of the SPI interrupts, therefore, this flag can only be cleared manually by software

10.12 SPI Transaction Size

The standard SPI protocol is based on 8-bit transactions. However, many devices on the market, specifically A/D and D/A converters, require transactions greater than 8 bits. To communicate with these types of devices using a standard SPI interface, the user has no choice but to send multiple 8-bit streams and use I/O pins to control the chip select line.

The SPI interface supports 8-bit transactions and can also be configured to support transactions from 1 to 32 bits in both transmit and receive directions. The value written into the SPISIZE register controls the transaction size. Upon reset, the SPI interface is configured for 8-bit transactions.

The SPI interface does also support transaction of more than 32 bit in size. For transaction > 32bit in size the granularity is 8 bit and the maximum transaction size is 228 bytes.

Care must be taken to avoid SPI double data buffer over / under run condition when transaction size is > 32 bit.

TABLE 108:SPI TRANSACTION SIZE – SPISIZE SFR C3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	1	1

Bit	Mnemonic	Description
7:0	SPISIZE[7:0]	SPI transaction Size If < 32 : Transaction Size = SPISIZE + 1 If >= 32: Transaction Size = (SPISIZE *8) - 216 Default Transaction Size = 8 bits

Four formulas control the SPI transaction size:

For Transactions Size <= 32 bits

Transaction Size = SPISIZE[7:0] +1

Or

SPISIZE[7:0] = Transaction Size - 1

For Transactions Size > 32 bits

Transaction Size = [(SPISIZE[7:0] * 8) –216]

Or it can be expressed by:

$$SPISIZE[7:0] = \frac{[\text{ Transaction Size} + 216]}{8}$$

The following table provides examples:

TABLE 109: TRANSACTION SIZE VS. SPISIZE[7:0]

SPISIZE[7:0]	Transaction Size
0x07	8-bit
0x0B	12-bit
0x0D	14-bit
0x10	17-bit
0x17	24-bit
0x1F	32-bit
0x20	40-bit
0x21	48-bit
0x23	64-bit

The transaction size must also be configured when the operating the SPI interface in slave mode.

10.13 SPI RX/TX Data Registers

Four SFR registers provide access to the SPI interface's receive and transmit data buffer. Performing a write operation to the SPI RX/TX buffer transfers the data to the transmit portion of the SPI interface, while a read operation reads the contents of the receive data buffer. The SPI 32-bit receive and transmit data buffers are double buffered to minimize the risk of data collision and to achieve optimal performance.

The SPI RXTX0 register contains bits 7:0 of the SPI interface RX/TX register.

TABLE 110: SPIRXTX0 REGISTER CONTENT FOR NORMAL AND REVERSED TRANSACTIONS

Operation	SPI Mode	SPIRXTXx Data is…
Read	MSB First	Right Justified
	LSB First	Left Justified
Write	MSB First	Left Justified
	LSB First	Right Justified

When the SPI is configured in master mode, writing to the SPIRXTX0 will trigger a data transmission. For this reason, when the transaction size is larger than 8 bits, the SPIRXTX0 register must be written last.

TABLE 111:SPI RX / TX0 DATA REGISTER – SPIRXTX0 SFR C4H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	SPIRXTX0[7:0]	Read: SPI RXData[7:0] Right justified in normal mode, left justified in bit reversed mode Reading this register, clears the SPIAVF and SPIRXOVF flags
		Write: SPI TXData[7:0] if SPISIZE = 0x07 (8 bit) SPITXData[31:24] if SPISIZE = 0x1F (32 bit) Left justified in normal mode, right justified in bit reversed mode In master mode, writing to SPIRXTX0 triggers the transmission

TABLE 112:SPI RX / TX1 DATA REGISTER – SPIRXTX1 SFR C5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	SPIRXTX1[7:0]	Read: SPI RXData[15:8] if SPISIZE = 0x1F (32 bit) Right justified in normal mode, left justified in bit reverse mode
		Write: SPITXData[23:16] if SPISIZE = 0x1F (32 bit) Left justified in normal mode, right justified in bit reverse mode

TABLE 113:SPI RX / TX2 DATA REGISTER – SPIRXTX2 SFR C6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	SPIRXTX2[7:0]	Read: SPI RXData[23:16] if SPISIZE = 0x1F (32 bit) Right justified in normal mode, left justified in bit reverse mode
		Write: SPITXData[15:8] if SPISIZE = 0x1F (32 bit) Left justified in normal mode, right justified in bit reverse mode

TABLE 114:SPI RX / TX3 DATA REGISTER – SPIRXTX3 SFR C7H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	SPIRXTX3[7:0]	Read: SPI RXData[31:24] Right justified in normal mode, left justified in bit reverse mode
		Write: SPITXData[7:0] if SPISIZE = 0x1F (32 bit) Left justified in normal mode, right justified in bit reverse mode

10.14 SPI Data Input /Output

The SPI interface has the ability to perform data transactions in MSB first mode or LSB first. The SPIREVERSE bit of the SPISTATUS register controls whether the data will be transmitted MBS first or LSB first. Upon device reset, the SPIREVERSE bit equals 0 and data is transmitted in MSB first format.

The SPIREVERSE bit state will also affect the data transmission and the data reception buffer structure as shown in the following diagrams.

FIGURE 33: SPI TRANSACTION STANDARD MODE (SPIREVERSE = 0 : MSB FIRST)

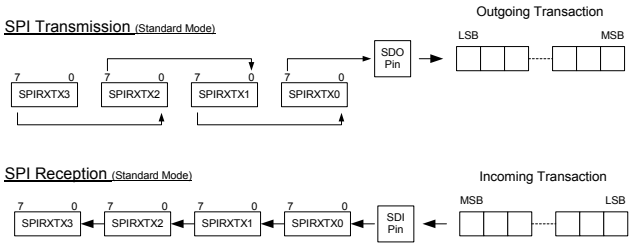
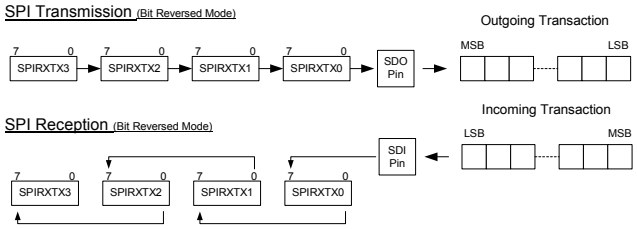


FIGURE 34: SPI TRANSACTION BIT REVERSE MODE (SPIREVERSE = 1: LSB FIRST)



The next tables gives examples of SPI transmission and reception in different modes if the SPI SDO pin is connected to the SDI pin.

SPISIZE = 0x0F (16 bit) / SPIREVERSE= 0 (MSB First

SPITX [3:0]				SPIRX [3:0]			
xx	xx	D3h	42h	xx	xx	42h	D3h
xx	xx	54h	A6h	xx	xx	A6h	54h

SPISIZE = 0x0F (32 bit) / SPIREVERSE= 0 (MSB First

SPITX [3:0]				SPIRX [3:0]			
45h	A3h	B2h	DF	DFh	B2h	A3h	45h
C3h	8Ah	49h	24h	24h	49h	8Ah	C3h

SPISIZE = 0x0F (32 bit) / SPIREVERSE= 1 (LSB First

SPITX [3:0]				SPIRX [3:0]			
45h	A3h	B2h	DF	DFh	B2h	A3h	45h
C3h	8Ah	49h	24h	24h	49h	8Ah	C3h

10.15 Non 8 bit multiple Transmission

For a non 8 bit multiple data transmission in master mode (when the data is not transmitted in multiples of 8 bits), the most significant bit of the data to be transmitted must first be placed at position 7 of the SPIRXTX0, with the remaining bits positioned as shown in the SPI transaction figures on the previous page.

For example if SPISIZE = 0x0B and SPIREVERSE = 0, the data transaction will measure 12 bits, MSB first. For the transmission to occur in the correct order, the lower 4 data bits must first be placed into bit positions 7:4 of the SPIRXTX1 register, with bits 11:4 written into bit position 7:0 of the SPIRXTX0 register. This will trigger the transmission.

The following is a sequence of steps to transmit 12 bits of data contained in an integer variable called *txmitdata*.

1. Clear the SPIRXTX3 and SPIRXTX2 registers (optional)
2. Put the lower quartet of the 12-bit data (bits 3:0) into the upper quartet of the SPIRXTX1 register
3. Write bit 7:0 of the 12-bit data into the SPIRXTX0 register
4. This will trigger a data transmission

In C, this is expressed as follows:

```
(…)
   SPIRXTX3 = 0x00;
    SPIRXTX2 = 0x00;
    SPIRXTX1 = (txmitdata << 4)&0xF0;        //Write the lower quartet of data
                                             //into the upper quartet of SPIRXTX1 register

   readflag = SPIRXTX0          //-Dummy Read the SPI RX buffer to clear the RXAV Flag
                                ///(Facultative if SPINOCS is monitored)

  SPIRXTX0 = dacdata >> 4;   //Writing to SPIRXTX0 will trigger the transmission
```

For example to output 0x3A2 through the SPI interface configured in master mode and MSB first format, write 0x20 into the SPIRXTX1 SFR register and followed by 0x3A into the SPIRXTX0 register.

The reception of non multiple of 8 data when the SPI interface is configured to MSB first transaction is very straight forward as the data enters into the receiving buffer through the bit 0 of the SPIRXTX0 register and propagates towards the bit 7 of SPIRXTX3 register.

10.16 SPI Example Programs

Code Example:

UART to SPI Data Transmission Example

```
//-------------------------------------------------------------------------------------------------//
// SPI Transmit example.c
//-------------------------------------------------------------------------------------------------//
// This program sends characters received  on the UART to the SPI Interface
//-------------------------------------------------------------------------------------------------//

#include <VRS51L3074_SDCC.h>

//-----Global variables -------//

int  cptr = 0x00;       //general purpose counter

// --- function prototypes
void txmit0( unsigned char charact);
void uart0config(void);

//-------------------------------------------------------------//
//                    Main Function
//-------------------------------------------------------------//

void main (void){

    char value = 0x00;              //general purpose variable
    PERIPHEN1 = 0xC0;               //Enable SPI Interface

    INTCONFIG = 0x02;         //Erase  Bypass global int, before configuring the INT0 pin
event
                                    //This fix inadvertent INT0 interrupt that occurs when
                                    //INT0 cause  is set to Rising edge

    INTSRC1 = 0x01;                 //INT0 vector source = INT0 pin
    INTPINSENS1 = 0x01;             //Set INT0 sensitive on edge(1) or Level(0)
    INTPININV1 = 0x00;              //Set INT0 Pin sensitivity on Normal Level(0) / Inverted (1)
    INTEN1 = 0x01;                  //Enable INT0 (bit0) Interrupt

    INTCONFIG = 0x01;               //Enable Global interrupt

    while(1);

}//end of Main

//-------------------------------------------------------------------------------------------------//
//---------------------------- Interrupt Functions ------------------------------//
//-------------------------------------------------------------------------------------------------//

//-------------------------------------------------------------------------------//
//      Interrupt INT0                                                           //
//      Send character received on the SPI Interface                             //
//-------------------------------------------------------------------------------//

void INT0Interrupt(void) interrupt 0
   {
   //-- Send "EXT INT0 Received" on UART0
   cptr = 0x00;                     // Init cptr to pint to message beginning
   INTEN1 = 0x00;                   /Disable  Interrupts
   SPICTRL = 0xE1;                  //SPI CLK = div by 256
                                    //SPI CS0 Active
                                    //SPI Mode 0
                                    //SPI Master

   SPISIZE = 0x07;                  //SPI SIZE = 8bit
   SPICONFIG = 0x10;                //LOAD on CS3
   SPIRXTX0 = S0BUF;                //Send Data Byte on SPI Interface

   INTEN1 = 0x01;                   //Enable Interrupt INT0
   }//end of INT0 interrupt
```

Code Example:
SPI Interface to 12-bit ADC and DAC

The following example program shows the initialization and use of the SPI module as an interface to serial ADC and DAC.

```
.//-------------------------------------------------------------------------------------------------------//
// VRS51L3xxx_Generic_SPI_based_ADC_DAC_Interf1.c
//-------------------------------------------------------------------------------------------------------//
// DESCRIPTION:
//              This Program demonstrates the configuration and use of the SPI interface
//              for interface to typical serial 12 bit A/D and D/A Converters.
//              The program read the A/D and output the read value out on a D/A converter
//              To perform the conversion the ADC requires 16 clock cycles and
//              the DAC requires 12 clock cycles.
//-------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

//---Functions prototypes
void ReadGEN_12BIT_ADC(void);                   //GEN_12BIT_ADC Read
void WriteGEN_12BIT_DAC(unsigned int );         //GEN_12_BIT_DAC Write

void V2KDelay1ms(unsigned int);                 //Standard Delay function

// Global variables definitions

idata unsigned char cptr = 0x00;
unsigned int at 0x0060 adcdata= 0x00;

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

        do{
          ReadGEN_12BIT_ADC();              //Read the A/D Converter
          WriteGEN_12BIT_DAC(adcdata);      //write into the D/A Converter
        }while(1);

}// End of main

//-------------------------------------------------------------------------------------------//
// NAME:                ReadGEN_12BIT_ADC
//-------------------------------------------------------------------------------------------//
// DESCRIPTION:
//                  Read the GEN_12BIT_ADC A/D
//                 ADC is connected to SPI interface using CS0
//                 Max clk speed is 3.2MHz, Fosc = 40MHz assumed
//-------------------------------------------------------------------------------------------//
void ReadGEN_12BIT_ADC()
    {
    int cptr = 0x00;
    char readflag = 0x00;

    //SPI Configuration Section
    //(Can be moved to Main function if only one device is connected to the SPI Interface)

    //Make sure the SPI Interface is activated
    PERIPHEN1 |= 0xC0;

    //--Wait activity stops on the SPI interface (Monitor SPINOCS)
    while(!(SPISTATUS &= 0x08));

    SPICTRL = 0x65;                 //SPICLK = /16 (2.5MHz)
                                    //CS0 Active
                                    //SPI Mode 1 Phase = 1, POL = 0
                                    //SPI Master Mode

    SPICONFIG = 0x40;               //SPI Chip select is automatic
                                    //Clear SPIUNDEFC Flag
                                    //SPILOAD = 0 -> Manual CS3 behaviour
                                    //No SPI Interrupt used

    SPISTATUS = 0x00;               //SPI transactions are in MSB First Format

    SPISIZE = 0x0E;                 //SPI Transaction Size are 15 bit

    //-Dummy Read the SPI RX buffer to clear the RXAV Flag
    readflag = SPIRXTX0;
    //-Perform the SPI read
     SPIRXTX0 = 0x00;                           //Writing to the SPIRXTX0 will trigger the SPI
                                                //Transaction

    while(!(SPISTATUS &= 0x02));            //Wait for the SPI RX AV Flag being set
    /*
    // -- It is possible to monitor the SPINOCS Flag instead of the SPIRXAV Flag
          //The code piece below shows how to do it.  However in that case,
          //No that the reading of the SPISTATUS register must be done at
          //least 4 System clock cycles after the Write operation to the SPIRXTX0 register
    //-Wait for SPINOCS Flag have time to be updated
    _asm
        NOP;
    _endasm;

    while(!(SPISTATUS &= 0x08));            //Wait activity stops on the SPI interface
    */
    //Read SPI data
    adcdata= (SPIRXTX1 << 8);
    adcdata+= SPIRXTX0;
    adcdata&= 0x0FFF;                       //isolate the 12 lsb of the read value
    }//end of ReadGEN_12BIT_ADC

//-------------------------------------------------------------------------------------------//
// NAME:                WriteGEN_12BIT_DAC
//-------------------------------------------------------------------------------------------//
// DESCRIPTION:
//                  Write 12bit Data into the GEN_12BIT_DAC device
//                  ADC is connected to SPI interface using CS1
//                  Max clk speed is 12.5MHz, Fosc = 40MHz assumed
//                  We will set the SPI prescaler to sysclk / 8
//-------------------------------------------------------------------------------------------//
void WriteGEN_12BIT_DAC(unsigned int dacdata)
    {
    char subdata = 0x00;
    char readflag = 0x00;
    PERIPHEN1 |= 0xC0;              //Make sure the SPI Interface is activated

    //--Wait activity stops on the SPI interface (Monitor SPINOCS)
    while(!(SPISTATUS &= 0x08));

    //SPI Configuration Section
    //Can be moved to Main function if only one device is connected to the SPI Interface

    SPICTRL = 0x4D;                 //SPICLK = /8 (MHz)
                                    //CS1 Active
                                    //SPI Mode 1 Phase = 1, POL = 0
                                    //SPI Master Mode

    SPICONFIG = 0x40;               //SPI Chip select is automatic
                                    //Clear SPIUNDEFC Flag
                                    //SPILOAD = 0 -> Manual CS3 behaviour
                                    //No SPI Interrupt used

    SPISTATUS = 0x00;               //SPI transactions are in MSB First Format
    SPISIZE = 0x0B;                 //SPI Transaction Size are 12 bit

    //-Format the 12 bit data so data bit 11 is positioned on bit 7 of SPIRXTX0
    // and data bit 0 is positioned on bit 4 of SPIRXTX1 and Perform the SPI write operation

    dacdata &= 0x0FFF;              //Make sure dacdata is <= 0FFFh  (12 bit)

    SPIRXTX3 = 0x00;
    SPIRXTX2 = 0x00;
    SPIRXTX1 = (dacdata << 4)&0xF0;

    //-Dummy Read the SPI RX buffer to clear the RXAV Flag
    // (Facultative if SPINOCS is monitored)
    readflag = SPIRXTX0;

    SPIRXTX0 = dacdata >> 4;        //Writing to SPIRXTX0 will trigger the transmission

    //--Wait the SPI transaction completes
    //      This section can be omitted if a check of activity on the SPI Interface
    //      is made before each access to it in master mode

    //Wait for the SPI RX AV Flag being set
    while(!(SPISTATUS &= 0x02));

    // -- It is possible to monitor the SPINOCS Flag instead of the SPIRXAV Flag
          //The code piece below shows how to do it.  However in that case,
          //No that the reading of the SPISTATUS register must be done at
          //least 4 System clock cycles after the Write operation to the SPIRXTX0 register
    /*
    //-Wait for SPINOCS Flag have time to be updated
    _asm
        NOP;
    _endasm;
    //--Wait activity stops on the SPI interface  (monitor SPINOCS Flag)
    while(!(SPISTATUS &= 0x08));
    */
    }//end of WriteGEN_12BIT_DAC
```

Code Example:
Performing > 32bit Data Transfer on SPI

The following demo program demonstrates the use of the SPI interface to perform 131 bytes data transfer Read / Write on the SPI interface.

```
//-------------------------------------------------------------------------------------------------------------//
// V2K_SPI_131BComm_FM25CL64_SDCC.c
//-------------------------------------------------------------------------------------------------------------//
// DESCRIPTION:
//   This Program demonstrates the configuration and use of the SPI interface
//   to access the FM25CL64.  This demonstration program takes advantage of the
//   ability to perform > 32bit transactions.
//   In this demo program, transactions of 131 bytes are performed
//
//   The program do the following operations
//       1) Initialize the SPI interface
//       2) Init XRAM address 0x0000 - 0x007F with LSB of XRAM address
//       3) Init XRAM address 0x0080 to 0x0FFF == 0x00
//       4) Send a WREN command
//       5) Copy XRAM address 0x0000 to 0x007F into address 0x0000 to 0x00F
//          of the FM25CL64 in one SPI Transaction
//       6) Read FM25CL64 address 0x0000 to 0x007F of the FM25CL64 and copy
//          the data read into XRAM from addres 0x0100 to 0x017F
//          in one SPI Transaction (by pooling)
//       7) Read back address 0115h
//-------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

//--Init pointer to XRAM base address
xdata at 0x0000 unsigned char xrambase;          //Init  a char variable pointing to XRAM
xdata unsigned char * data xramptr = &xrambase ; //Init a pointer in IRAM pointing to the
                                                 //xrambase var.

//-------------------------------------------------------------------------//
//                        MAIN FUNCTION
//-------------------------------------------------------------------------//
void main (void) {
    char readflag = 0x00;
    char count;

    //SPI Configuration Section
    PERIPHEN1 |= 0xC0;                           //Activate the SPI Interface

    //--Wait activity stops on the SPI interface (Monitor SPINOCS)
    while(!(SPISTATUS & 0x08));

    readflag = SPIRXTX0;                         //Dummy Read to clear the RXAV Flag
    SPICTRL = 0x85;                              //SPICLK = Fosc/32 (1.25MHz)
                                                 //CS3 Active
                                                 //SPI Mode 0 Phase = 1, POL = 0
                                                 //SPI Master Mode

    SPICONFIG = 0x40;                            //SPI Chip select is automatic
                                                 //Clear SPIUNDEFC Flag
                                                 //No SPI Interrupt used
    SPISTATUS = 0x00;                            //SPI transactions -> MSB First Format

    //-------------------------------------------------------------------------
    // Init XRAM address 0x0000 to 0x07F == LSB of address
    // Init XRAM address 0x0080 to 0x0FFF == 0x00
    //-------------------------------------------------------------------------
    xramptr = &xrambase;                         //Init xramptr

// Init XRAM address 0x0000 to 0x07F == LSB of address
    do{
     *xramptr =  (char) xramptr;
     xramptr++;
     }while(xramptr < 0x80);

    // Init XRAM address 0x0080 to 0x0FFF == 0x00
     do{
           *xramptr =  0x00;
           xramptr++;
           }while(xramptr < 0x1000);

    //-----------------------------------------------------------------//
     // Send the WREN command to the FM25CL64
     //-----------------------------------------------------------------//
    //First, Send a WREN command to the FM25CL64
    readflag = SPIRXTX0;                         //Dummy Read to clear the RXAV Flag
     SPISIZE = 0x07;                             //SPI Transaction Size is 8 bit
     SPIRXTX0 = 0x06;                            //WREN command

     //--Wait activity stops on the SPI interface (Monitor SPINOCS)
    _asm;    //SPINOCS require 4 cycles before sampling
    NOP;
    NOP;
    NOP;
    NOP;
    _endasm;
    while(!(SPISTATUS & 0x08));                  //Wait for SPINOCS to go high

    //-------------------------------------------------------------------------------------------------------------//
    // Write 129 bytes into the FM25CL64 in one SPI Transaction (by pooling)
    //-------------------------------------------------------------------------------------------------------------//
    //--Get ready to copy 5 bytes to FM25CL64
    xramptr = &xrambase;                         //Init pointer to XRAM to retrieve data to
                                                 //be written into F-RAM
    readflag = SPIRXTX0;                         //Dummy Read to clear the RXAV Flag
    SPISIZE = 0x9E;                              //Transaction Size 8 bytes total

    //--Fill the 32bit SPIRXTX Buffer
    SPIRXTX3 = *xramptr++;                       //Write First data byte
    SPIRXTX2 = 0x00;                             //Set LSB of FM25CL64 Address
    SPIRXTX1 = 0x00;                             //Set MSB of FM25CL64 Address
    SPIRXTX0 = 0x02;                             //Write memory command  -> this start
                                                 //the transaction

    //Wait for the SPI TXEMPTY Flag to get set before sending new data
    //

    do{
        while(!(SPISTATUS & 0x01));
        SPIRXTX3 = *(xramptr + 3);               //Write next data bytes into SPI Tx buffer
        SPIRXTX2 = *(xramptr + 2);               //
        SPIRXTX1 = *(xramptr + 1);               //
        SPIRXTX0 = *(xramptr);                   //
        xramptr += 4;                            //get ready for next data
        }while(xramptr < 0x83);

    //--Wait activity stops on the SPI interface (Monitor SPINOCS)
    _asm;    //SPINOCS require at least 4 cycles to be sampled
    NOP;
    NOP;
    NOP;
     NOP;
     _endasm;
     while(!(SPISTATUS & 0x08));                 //Wait for SPINOCS to go high

     //-------------------------------------------------------------------------------------------------------------//
     // Read 128 bytes from the FM25CL64 in one SPI Transaction (by pooling)
     //-------------------------------------------------------------------------------------------------------------//
    //SPI Configuration Section
     readflag = SPIRXTX0;                        //Dummy Read to clear the RXAV &
                                                 //RXOV Flags if set
     SPISIZE = 0x9E;                             //Set SPISIZE[7:0] for 8 bytes total

      xramptr = (&xrambase + 0x100);             //Set pointer to XRAM

    //--Fill the 32bit SPIRXTX Buffer
     SPIRXTX3 = 0x00;                            //
     SPIRXTX2 = 0x00                             //Set LSB of FM25CL64 Address
     SPIRXTX1 = 0x00;                            //Set MSB of FM25CL64 Address
     SPIRXTX0 = 0x03                             //READ command-> this start
                                                 //the transaction
     //Wait for the SPI RX AV Flag to be set -> RX buffer full
    while(!(SPISTATUS & 0x02));

    *xramptr++ = SPIRXTX0;                       //Read the first Data Byte
                                                 //from FM25CL64
    //Retrieve the next 124 Data bytes stored into the FM25CL64
    do
        {
        //Wait for the SPI RX AV Flag being set -> RX buffer full
        while(!(SPISTATUS & 0x02));
        *xramptr++ = SPIRXTX3;
        *xramptr++ = SPIRXTX2;
        *xramptr++ = SPIRXTX1;
        *xramptr++ = SPIRXTX0;
       }while(xramptr < 0x017D);

    // retrieve last 3 bytes...

     while(!(SPISTATUS & 0x02));
        *xramptr++ = SPIRXTX2;
        *xramptr++ = SPIRXTX1;
        *xramptr++ = SPIRXTX0;
  while(1);
}// End of main
```

11 I²C Interface

All VRS51L3xxx include an I²C interface that can operate in master and slave mode. In master mode, the communication speed on the I²C is programmable, optimizing communication between I²C-based devices. Long or heavily loaded I²C bus applications are likely to require slower communication speeds.

11.1 I²C Bus Pull-Up Resistors

By definition, the I²C requires that the user include external pull-up resistors on the SCL and SDA lines. The pull-up voltage can be either 3.3 or 5 volts. The i/O associated with SCL and SDA are 5V–tolerant, making it possible to interface 5V, I²C-based devices with the VRS51L3xxx.

The proper value for the pull-up resistor and the proper communication speed depend on bus characteristics such as length and capacitive load.

Note that the pull-up resistor value should not be below 1.25K ohms if running the I²C bus at 5V; and 750 ohms if operating at 3.3V. This is required in order to limit the current to 4mA (maximum current of the I/O port connected to the I²C interface).

11.2 I²C Interface alternate pins

Upon reset, the I²C interface signal SCL and SDA are mapped into pins P3.4 and P3.5, respectively. However it is also possible to map these signal into the P1.6 and P1.7 pins.

Bit 5 of the DEVIOMAP register (SFR E1h) is used to configure the mapping of the I²C interface at the I/O level, as shown in the following table:

TABLE 115: I²C MODULE MAPPING

DEVIOMAP.5 Bit Value	SCL Mapping	SDA Mapping
0 (Reset)	P3.4	P3.5
1	P1.6	P1.7

11.3 I²C Control and Status Registers

Four SFR registers are dedicated to the I²C interface. The I²C configuration register I2CCONFIG enables:

- Selection of master or slave operation
- Forcing a start condition after an acknowledge phase
- Manual control of the SCL line
- Activation of the master arbitration monitoring mechanism
- Interrupt activation

TABLE 116:I2C CONFIGURATION REGISTER - I2CCONFIG SFR D1H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0

Bit	Mnemonic	Description
7	MASTRARB	Master Lost Arbitration and Mechanism and Interrupt 0 = Deactivated 1 = Master lost arbitration monitoring and interrupt is enabled
6	I2CRXOVEN	I²C RX Overrun Interrupt Enable 0 = I²C RX Overrun interrupt is deactivated 1 = I²C RX Overrun interrupt is enabled
5	I2CRXAVEN	I²C RX Available Interrupt Enable 0 = I²C RX Available interrupt is deactivated 1 = I²C RX Available interrupt is enabled
4	I2CTXEEN	I²C TX Empty Interrupt Enable 0 = I²C TX empty interrupt is deactivated 1 = I²C TX empty interrupt is enabled
3	I2CMASTART	I²C Master Create Start 0 = No start condition is created after data acknowledge phase 1 = Master will create a start condition after the next data acknowledge phase This bit will be cleared when the I²C is idle
2	I2CSCLLOW	Keep the I²C SCL Low Setting this bit to 1 will force the SCL line low. This bit is read by the I²C interface when it enters in the data I²C. This bit must not be set during the acknowledge phase.
1	I2CRXSTOP	I²C Reception Stop 0 = The I²C received will acknowledge after receiving the next data byte 1 = The I²C receiver will not acknowledge after the next data byte is received
0	I2CMODE	I2C Mode Enable 0 = I²C interface operates in slave mode 1 = I²C Interface operates in master mode

The I2CMODE bit of the I2CCONFIG register, when set to 1, will configure the I²C interface as a master.

In master mode, the VRS51L3xxx I²C interface controls the I²C bus. It can initiate and end I²C transactions. In master mode, the I²C interface also controls the communication speed.

Clearing the I2CMODE bit of the I2CCONFIG register will configure the I²C interface as a slave. Slave mode can be useful for applications in which the VRS51L3xxx operates as a peripheral in a host-controlled system.

When in master mode, the I²C interface can be configured to generate a start condition after the next data acknowledge phase. This is done by setting the I2CMASTART bit to 1.

When the MASTRARB bit is set to 1, communications of the I²C will be monitored and an interrupt will be generated if arbitration with slave devices on the bus is lost. The interrupt flag associated with this process is the I2CERROR bit of the I2CSTATUS register.

If the I2CRXSTOP bit is set to 1, the I²C interface will not acknowledge it has received the next data byte: It will generate a stop condition instead, which will end the transaction.

When set to 1, the I2CSCLLOW bit will force the I²C interface to pull the SCL line low during the next data acknowledge phase. This feature enables the user to add the equivalent of wait states to the transfer in order to support "slow" devices connected to the I²C bus.

The state of the I2CSCLLOW is sampled when the I²C interface enters the data phase. This bit must not be set during the data acknowledge phase, which can be monitored using the I2CACKPH bit of the I2CSTATUS register.

The I²C interface includes support for four interrupt conditions via two interrupt vectors.

- RX Data Available
- RX Overrun
- TX Empty
- Master Lost Arbitration

The following table summarizes the possible interrupt sources at the I²C interface level.

TABLE 117: I²C INTERRUPT SOURCES

I²C Interrupt	I2CCONFIG bit (Set to 1 to activate)	Interrupt Vector
RX Data Available	I2CRXAVEN	4Bh (Int 9)
RX Overrun	I2CRXOVEN	0x4B (Int 9)
TX Empty	I2CTXEEN	0x4B (Int 9)
Master Lost Arbitration	MASTRARB	0x53 (Int 10)

To activate the I²C interface interrupts, the corresponding enable bit of the I2CCONFIG register must be set to 1. This will allow the I²C interrupt to propagate to the VRS51L3xxx's interrupt controller. In order for the I²C interrupt to be recognized by the processor, the corresponding bit of the INTEN2 and INTSRC2 registers must be configured accordingly. See the interrupt section for more details.

11.4 I²C Timing Control Register

The I2CTIMING register controls the communication speed when the I²C interface is configured in master mode. When in slave mode, it defines the values of the setup and hold times.

TABLE 118:I²C TIMING REGISTER - I2CTIMING SFR D2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	1	1	0	0

Bit	Mnemonic	Description
7:0	I2CTIMING[7:0]	I²C master/slave timing configuration register See Below

The following formulas demonstrate the impact of the I2CTIMING value on the communication speed and setup/hold times.

In master mode:

$$\text{SCL period} = \frac{\text{I2CCLK}}{32*(\text{I2CTIMING[7:0]} + 1)}$$

The following table provides examples of the I2CTIMING values and the corresponding communication speed:

TABLE 119: I²C COMMUNICATION SPEED VS. I2CTIMING REGISTER VALUE (FOSC = 40MHZ)

I2CTIMING	I2C Com Speed
00h	1.25 MHz
02h	416.77 kHz
0Ch (Reset)	96.15 kHz
7Ch	10kHz
FFh	4.88kHz

In Slave Mode:

Set-up/Hold Time = I2CCLKperiod * I2CTIMING[7:0]

In this case, the precision is: 2 x I2CCLKperiod

TABLE 120: I²C SETUP AND HOLD TIME VS. I2CTIMING REGISTER VALUE (FOSC = 40MHZ)

I2CTIMING	Setup/Hold Time
00h	0 uS
0Ch	0.3 uS
FFh	6.38 uS

11.5 I2CSTATUS Register

Monitoring of the I²C interface can be done via the I2CSTATUS register located at SFR address D4h. The I2CSTATUS register is read-only.

TABLE 121: I²C STATUS REGISTER - I2CSTATUS SFR D4H

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

Bit	Mnemonic	Description
7	I2CERROR	Slave Mode Error Flag: 0 = No Error 1 = Indicates that the I²C interface received an unexpected stop This flag is reset the next time the I²C interface exits from an idle state (see below)
		Master Mode 0 = No Arbitration Error 1 = I²C interface has lost arbitration This flag is reset the next time the I²C interface exits from an idle state (see below)
6	I2CNOACK	I²C Acknowledge Error Flag 0 = Acknowledge was received normally 1 = No acknowledge was received during the last acknowledge phase This flag is reset the next time the I²C interface exit from the idle state (see below)
5	I2CSDA	I²C SDA
4	I2CACKPH	When set, this flag indicates that the I²C interface is in 'Data Acknowledge Phase.' 5 phases of I²C protocol: 1. Idle 2. Device ID 3. Device ID Acknowledge 4. Data 5. Data Acknowledge
3	I2CIDLEF	I²C is idle 0 = I²C interface is communicating 1 = I²C interface is inactive (idle phase) and the SCL and SDA lines are high
2	I2CRXOVF	I²C RX Overrun Interrupt Flag 0 = No I²C RX overrun condition detected 1 = I²C data collision occurred
1	I2CRXAVF	I²C RX Available interrupt Flag 0 = I²C receive buffer is empty 1 = Data is present in the I²C RX buffer
0	I2CTXEMPF	I²C TX Empty interrupt Flag 0 = I²C transmit buffer is full 1 = I²C transmit buffer is ready to receive new data

The I2CERROR flag indicates that an error condition has occurred on the I²C interface. In master mode, the I2CERROR flag will be set by the I²C interface, if it loses bus arbitration.

In slave mode, if an unexpected stop is received, the I2CERROR flag will be set. The I2CERROR flag will be automatically reset by the I²C interface the next time it exits an idle state.

If the I2CNOACK flag is set to 1, it signifies that the slave device did not acknowledge the last data byte it received.

The I²C interface also monitors the synchronization of the SDA line. When synchronization is lost, the I2CSDASYNC bit of the I2CSTATUS register will be set by the I²C interface.

The I2CSDASYNC bit of the I2CSTATUS register returns the value of the SDA line the moment a read operation is performed on the I2CSTATUS register.

When set, the I2CACKPH bit indicates that the I²C interface is currently in the data acknowledge phase.

The I2CSDASYNC and I2CCKPH bits can read to determine whether the slave device has acknowledged. If both bits are set to 1 at a given time, the slave device did not acknowledge.

When set, the I2CIDLEF indicates that the I²C bus is idle and that a transaction can be initiated. Before initiating an I²C data transfer, it is recommended to check the state of the I2CIDLEF bit. This bit indicates whether or not a data transfer is currently in progress.

When new data is received in the I²C receive buffer, the I2CRXAVF interrupt flag will be set. Data must be retrieved from the I2CRXTX buffer before the reception of the next data byte is complete.

When set, the I2CRXOVF flag indicates an overrun condition in the I²C interface receive buffer and the data is potentially corrupted. The I2CTXEMPF interrupt flag is set by the I²C interface when the transmit data buffer is ready to receive another data byte.

11.6 I2CRXTX Register

The I²C interface transmit and receive buffers are accessed via the I2CRXTX SFR register, which is accessible at SFR address D5h.

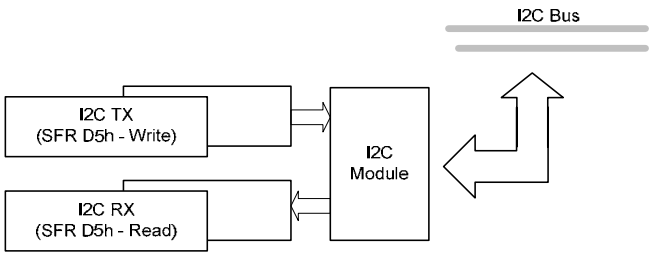
TABLE 122:I²C DATA RX / TX REGISTER I2CRXTX - SFR D5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	I2CRXTX[7:0]	Read: I²C Receive Buffer Reading the I2CRXTX register will clear the I2CRXAV and I2CRXOV flags
		Write: I²C Transmit Buffer In Master mode, writing into the I2CRXTX register will trigger the transmission

The I2CRXTX SFR is made up of two distinct registers set: One for data transmission and one for data reception. Each register set features a double data buffer as shown in the diagram below.

FIGURE 35: I2CRXTX[7:0] DATA BUFFER STRUCTURE



When the I2C is configured in master mode, writing any data to the I2CRXTX register will trigger a data transmission. Unless the I²C interface is currently transmitting a byte, if the TX Empty interrupt is activated, a TX Empty interrupt will occur as soon as a new data byte is placed into the I2CRXTX register.

In order to read data from the I²C interface, when configured in master mode, the device ID with the R/W bit set to 1 must first be sent to the device. Then the program should wait for the I2CRXAV flag of the I2CSTATUS register to be set. At that point, the data can be read from the I2CRXTX register.

As previously mentioned, the data must be retrieved from the I2CRXTX register before the next data byte reception is complete. Otherwise, an I2C RX Overrun condition will occur.

Reading the content of the I2CRXTX register will automatically clear the I2CRXAV and I2CRXOV flags, which are read-only.

11.7 Checking Presence of the I²C Slave Device

When the I²C is configured in master mode, the bus master should check for the presence and the readiness of the slave I²C devices connected to the bus at least once after I²C bus initialization.

It is also recommended to check whether the slave device is present and ready before issuing a I²C transaction. This would allow the I²C Master to verify the I²C Slave device connected to the bus is not busy accomplishing other tasks and not responding.

This is especially true in the case of I²C EEPROM devices, which will not respond to the master when they are busy performing data writing operations. It is unnecessary to check before each transaction whether F-RAM memory devices are present and ready.

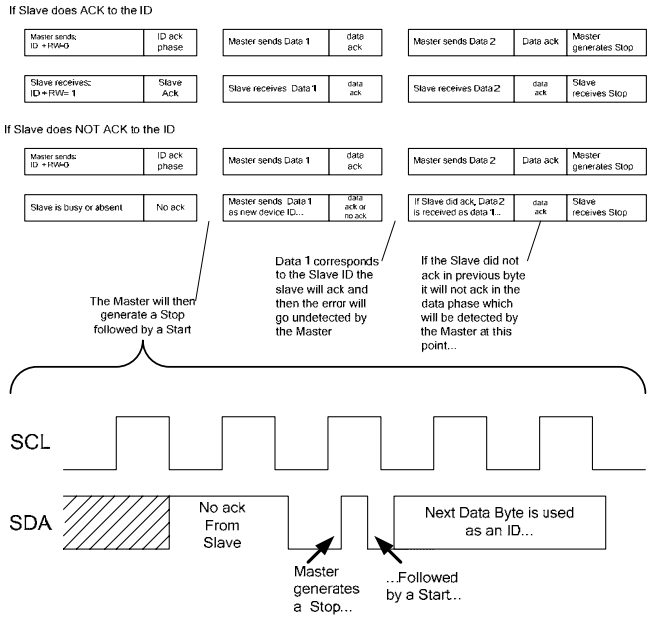
If the slave device fails to respond to the device ID sent by the master, the master will generate a stop condition immediately.

The I2CACKPH flag of the I2CSTATUS register is set to 1 during the I²C data acknowledge phase. However, this flag will remain inactive during the device ID acknowledge phase.

As a result, if the slave device fails to acknowledge following the transmission of the slave device ID, the I²C interface will only detect the error condition after data acknowledge phase.

The following diagram represents what could happen if the slave device fails to generate an ck signal following the device ID sent by the master:

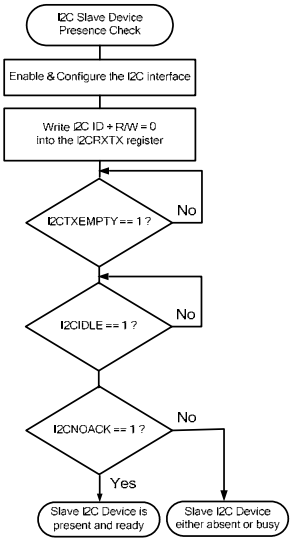
FIGURE 36: I2C INTERFACE BEHAVIOR WHEN SLAVE DEVICE DOES NOT ACK TO DEVICE ID



To work around this situation, send a device ID with R/W = 0 followed by a stop. If the device fails to respond the I²C ID, the I2CNOACK bit of the I2CSTATUS register will be set to 1 when the I²C bus becomes idle.

If the I2CNOACK is not set, then the I²C master can proceed with the I²C transaction starting with the device ID + R/W followed by the data to be written or read.

FIGURE 37: CHECKING PRESENCE AND READINESS IF I2C SLAVE



Code Example:

Slave I²C device presence check

The function example below checks whether the device pointed to by the device ID responds to the master command. As mentioned in the previous paragraph, this check should be done to ensure the targeted I²C slave device is present on the I²C bus and is ready.

```
//-------------------------------------------------------------------------------------------------------------------//
// Function I2CSlaveReady(char)        //
//-----------------------------------------//
//  Descrition:
//   Master I2C Slave Ready Check function
//  Check if the device pointed by the device ID does respond to the Master command
//  This check should be done to ensure the targeted I2C Slave device is
//   present on the I2C bus and is ready.
//   The Slave Ready Check should be performed at least once after I2C module
//   initialisation.
//   It should also be performed before each transactioln sequences to slow
//   devices such as EEPROM.  It is not necessary to perform a I2C Slave ready
//   check when the target device is I2C F-RAM device.
//-------------------------------------------------------------------------------------------------------------------//
char I2CSlaveReady(char id)
    {
    char addrtemp= 0x00;              //temporary address holding variable

   I2CCONFIG = 0x01;

   I2CRXTX = id;                  //Write I2C device ID + W
   WaitTXEMP();

    //--Wait for I2C IDLE  (This will generate a STOP)
   WaitI2CIDLE();

    if((I2CSTATUS & 0x40)!= 0x00 )  //If I2CNOACK == 1 -> No answer -> Return 01
      return    0x00;
      else
       return    0x01;

   }//end of I2CSlaveReady(char id);
```

11.8 I2C Slave Device ID and Advanced Configuration

When operating in slave mode, the device ID on the I²C interface is configurable. The seven upper bits of the I2CIDCFG register contain the user-selected device ID. Bit 0 of the I2CIDCFG register has two distinct roles.

TABLE 123:I²C DEVICE ID CONFIGURATION - I2CIDCFG SFR D3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
[7:1]	I2CID[6:0]	Slave I²C device ID as selected by user
0	I2CADVCFG	Read: Indicates that the I²C slave has received ID that is different from the I2CID[6:0]. This flag is cleared when the received ID corresponds with the I2CID
		Writing: Slave Mode: 1= The I2CRXAV flag is Not raised when the I²C slave receives a device ID 0 = The I2CRXAV flag will be raised for each data byte on the I2C bus. However if the received device Is does not match the I2CID[6:0] the I2C module will not acknowledge. Master Mode: 1 = The Master will not enter in a “wait State” if there is a mismatch between the expected SCL state and what it should be. 0 = Enables monitoring of the SCL line in wait state mode in case of mismatch of the SCL line vs. the expected value

The I2CAVCFG provides advanced control on I²C interface operations. Its functionality depends whether the I²C interface is configured in master or in slave mode.

In Master Mode:

When the I²C interface operates in master mode and the I2CADVCFG is cleared, the I²C interface module will continuously monitor the SCL line. If the slave device drives the SCL line into an incorrect state, the I²C interface will enter wait state mode until the slave device releases the SCL line. This mode can be useful for a I²C communication debug.

When the I2CADVCFG bit is set and the device operates in master mode, no monitoring of the SCL line will be executed by the I²C module and the transaction will proceed independently of the level of the SCL line.

In Slave Mode

When the I²C interface is configured in slave mode, the state of the I2CADVCFG bit when read indicates whether the ID received matches the current device ID. If the I2CIDCFG reads as 1, then the received ID corresponds to the I2CID[6:0] value.

The value written to the I2CADVCFG bit will define whether the I²C interface will monitor the transactions on the I²C bus or not. When the I2CADVCFG bit is cleared, the I²C interface will monitor the activity on the I²C bus and will raise an RXAV interrupt for each data byte received from the I²C bus, including the device ID at the beginning of a transaction.

Setting the I2CADVCFG bit to 1 will deactivate the I²C interface transactions monitoring feature for transaction beginning with a device ID that does not match the I2CID[6:0] value.

Independent of the value written in the I2CADVCGF bit of the I2CIDCFG register, if the I²C transaction device ID does not match the I2CID[6:0] value, the I²C interface will not acknowledge the incoming data nor will it transmit data to the master.

11.9 I²C Interface Example Programs

Code Example:
F-RAM / EEPROM Interface Programs

The following program provides example code for I²C control of F-RAM or EEPROM devices.

```
//-------------------------------------------------------------------------//
// VRS2k-I²C _FRAM/EEPROM.c  //
//-----------------------------------//
// This example program demonstrate the use of the I²C
// interface to perform basic read and write operations on a
// Standard EEPROM device.
//-------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

//----Global variables ------//
int  cptr = 0x00;       //general purpose counter

// --- Function prototypes
char EERandomRead(char,int);
char EERandomWrite(char, char, int);
void WaitTXEMP(void);
void WaitRXAV(void);
void WaitI2CIDLE(void);
void wait();
//-------------------------------------------------------------//
//--------------  MAIN FUNCTION ------------------//
//-------------------------------------------------------------//
void main (void){
    PERIPHEN1 = 0x20;       //Enable I²C  Interface

   INTCONFIG = 0x02;      //Erase Bypass global int, before configuring the INT0 pin event
                          //This fix inadvertent INT0 interrupt that occurs when
                          //INT0 cause  is set to Rising edge

     INTSRC1 = 0x01;          //INT0 vector source = INT0 pin
     INTPINSENS1 = 0x01;    //Set INT0 sensitive on edge(1) or Level(0)
     INTPININV1 = 0x00;       //Set INT0 Pin sensitivity on Normal Level(0) / Inverted (1)
     INTEN1 = 0x01;           //Enable INT0 (bit0) Interrupt

     INTCONFIG = 0x01;       //Enable Global interrupt

     while(1);
}//end of Main

//-------------------------------------------------------------------------//
//------------------------ Interrupt Functions ----------------------------//
//-------------------------------------------------------------------------//

//-----------------------------//
//---- Interrupt INT0 ----//
//-----------------------------//
void INT0Interrupt(void) interrupt 0
    {
    char x;

    //-- Send I²C  stuff
    cptr = 0x00;            // Init cptr to pint to message beginning
    INTEN1 = 0x00;          //Disable  Interrupts

    x =  EERandomWrite(0xA0, 0x36, 0x0206);       //Perform Write operation
    Delay1ms(100);
    x = EERandomRead( 0xA0, 0x0206);              //Perform Read operation

    INTEN1 = 0x01;          //Enable Interrupt INT0
    }//end of INT0 interrupt

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

//-------------------------------------------------------------------------//
//----  Function EERandomRead(char eeidw,int address)     ----//
//-------------------------------------------------------------------------//
char EERandomRead(char eeidw,int address){
    I2CTIMING = 0x20;              // I²C  Clock Speed = about 100kHz
    I2CCONFIG = 0x01;              //I²C  is Master
    I2CRXTX = eeidw;               //Write I²C  device ID + W
    WaitTXEMP();
    I2CRXTX = address >> 8;      //Write I²C  ADRSH
    WaitTXEMP();
    I2CRXTX = address;             //Write I²C  ADRSL

    //--Wait for I²C IDLE  (This will generate a STOP)
    WaitI2CIDLE();

    //--Start a Preset ADRS read (This will generate a START)
    I2CRXTX = eeidw+1;             //Write I²C  device ID + R
    WaitTXEMP();
    I2CCONFIG |= 0x02;           //Force I²C  to Not Acknowledge after
                                 //receiving the next data byte
    WaitRXAV();                  //Wait for RX Available bit, This will trigger I²C  Reception
    return I2CRXTX;              //Return Data Byte

    }//End of EERandomRead

//-------------------------------------------------------------------------------------//
//-----  Function EERandomWrite(char eeid,char data, int address) ----------//
//-------------------------------------------------------------------------------------//
char EERandomWrite(char eeidw, char eedata, int address){
    I2CTIMING = 0x20;          // I²C  Clock Speed = about 100kHz
    I2CCONFIG = 0x01;          //I²C  is Master
    I2CRXTX = eeidw;           //Write I²C  device ID + W

    WaitTXEMP();

    I2CRXTX = address >> 8; //Write I²C  device ID + W
    WaitTXEMP();

    I2CRXTX = address;         //Write I²C  device ID + W
    WaitTXEMP();

    I2CRXTX = eedata;          //Write I²C  device data
    WaitTXEMP();

    return I2CRXTX;            //Return Data Byte

    }//End of EERandomWrite

//-----------------------------------------------------//
//--------  Function WaitTXEMP() ----------//
//-----------------------------------------------------//
void WaitTXEMP()
    {
    wait();
    do{

    USERFLAGS = I2CSTATUS;
    USERFLAGS &= 0x01;           //isolate the I²C  TX EMPTY flag

    }while( USERFLAGS == 0x00);     //Wait for I²C  TX EMPTY
    }//end of Void WaitTXEMP()


//-----------------------------------------------------//
//--------  Function WaitRXAV() ------------//
//-----------------------------------------------------//
```

```
void WaitRXAV()
    {
    wait();
    do{

    USERFLAGS = I2CSTATUS;
    USERFLAGS &= 0x02;          //isolate the I2CRXAV flag

    }while( USERFLAGS == 0x00);     //Wait for I²C  RX AVAILABLE

    }//end of Void WaitRXAV()

//-----------------------------------------------//
//--------  Function WaitI2CIDLE() ---------//
//-----------------------------------------------//
void WaitI2CIDLE()
    {
    wait();
     do{
    USERFLAGS = I2CSTATUS;
    USERFLAGS &= 0x08;          //isolate the I²C  idle flag
    }while( USERFLAGS == 0x00);

    }//end of Void WaitI2CIDLE()

//-----------------------------------------//
//--------  Function Wait() ----------//
//-----------------------------------------//
void wait(){
    char i=0;
    while (i<25) {i++;};
    }
```

Code Example:

I2C Interface used in slave mode

The following example program presents how to configure and use the I²C interface in slave mode to create an I/O port expander using the VRS51L3xxx.

```
//-------------------------------------------------------------------------------------------------------------------/
//  V2K_I2C_Slave_IO_Exp_SDCC.c  //
///-----------------------------------------------//
// Description:
//
//   This demo program illustrate how to use the VRS51L3074 I2C interface configured
//   in Slave Mode to create an I2C based I/O expander.
//
//   The Memory Map of Slave I2C device as seen from the Master I2C device is
//   as shown below. The address is constituted of 1 byte.
//
//     I2C regs   V3K
//     address    Register
//     -------  --------
//       0x00     P0       //IO port 0 register
//       0x01     P1       //IO port 1 register
//       0x02     P2       //IO port 2 register
//       0x03     P3       //IO port 3 register
//       0x04     P4       //IO port 4 register
//       0x05     P5       //IO port 5 register
//       0x06     P6       //IO port 6 register
//       0x07~0x0F ~~~~~~~~  //Undefined / Restricted
//       0x10     P0PINCFG  //IO port 0 configuration register (1= Input, 0=Output )
//       0x11     P1PINCFG  //IO port 1 configuration register (1= Input, 0=Output )
//       0x12     P2PINCFG  //IO port 2 configuration register (1= Input, 0=Output )
//       0x13     P3PINCFG  //IO port 3 configuration register (1= Input, 0=Output )
//       0x14     P4PINCFG  //IO port 4 configuration register (1= Input, 0=Output )
//       0x15     P5PINCFG  //IO port 5 configuration register (1= Input, 0=Output )
//       0x16     P6PINCFG  //IO port 6 configuration register (1= Input, 0=Output )
//     >= 0x17      Undefined / Restricted
//
//   Multiple read or write operations can be performed in one access sequence
//   The first byte of each transaction does sets the address byte.
//
//   The first byte of all I2C write operation is the address byte.
//   Read operations are performed from the current address
//   when multiple byte read or write operation, the address value is incremented
//   after each data byte read from or written to the device.
//
//   The address value at reset is 0x00
//-------------------------------------------------------------------------------------------------------------------/
#include <VRS51L3074_SDCC.h>

//----Global variables ------//
char i2cvalue = 0x00;               //general purpose variable
char count = 0x00;
char address = 0x00;                    //initialise address
char state = 0x00;                     //init the I2C state indicator
char rxdata;
char bytecount;

// --- function prototypes
void WaitTXEMP(void);
void WaitRXAV(void);
void WaitI2CIDLE(void);
void wait();

//-------------------------------------------------------------------------/
//                    Main Function
//-------------------------------------------------------------------------/
 void main (void){
    //configure the I2C in slave mode
    PERIPHEN1 |= 0x20;                  //Enable I2C Interface

    I2CRXTX = 0x00;                     //Init the transmit portion of the I2CRXTX buffer
    I2CCONFIG = 0x70;                   //I2C Slave, Int RXOV, RX Available,
                                        //TX Empty Interrupts enabled

    I2CIDCFG = 0x41;                    //If I2C ID = 0x40,
                                        //Rx AV Flag will be raised only when the
                                        //correct ID is received
    I2CRXTX = 0x00;                     //Perform a dummy write into
                                        //the I2CRXTX register
                                        //to deactivate the TXEmpty Flag
    I2CTIMING = 0x01;

    //-Configure the I2C Slave interrupt
    INTSRC2 &= 0xFD;                    //Set INT9 vector source = I2C module
    IPININV2 = 0x00;
    IPINSENS2 = 0x00;                   //Set interrupt sensitivity to Level(0)
    INTEN2 = 0x02;                      //Enable I2C interrupt (bit1)
    GENINTEN = 0x01;                    //Enable Global interrupt

    //-Wait for I2C interrupt
     do{

    if (I2CSTATUS & 0x08)               //Check if I2C interface is IDLE
        state = 0x00;                   //init the I2C state indicator
        bytecount = 0x00;
    }while(1);
}//end of Main

//-----------------------------//
//----- Interrupt I2C ---//
//-----------------------------//
void I2CInterrupt(void) interrupt 9
    {
    char status;
     count++;
    INTEN2 = 0x00;                      //Disable Interrupt
    //Add code for Slave I2C

     status = I2CSTATUS;                //Read the I2C status register

     //---Case of I2C RX available
    if((status  & 0x02))                //RX available
        {
        bytecount++;
        if(state == 0x00)               //If state == 0x00 -> a new transmission begins
            {
            address = I2CRXTX;
            bytecount = 0x00;           //Reset byte counts
            //-- Prepare the transmit portion of the I2CRXTX buffer
            //   for the case where the address is set only to read a data
            switch(address)
                {
                case 0x00:    {I2CRXTX = P0; break;}
                case 0x01:    {I2CRXTX = P1; break;}
                case 0x02:    {I2CRXTX = P2; break;}
                case 0x03:    {I2CRXTX = P3; break;}
                case 0x04:    {I2CRXTX = P4; break;}
                case 0x05:    {I2CRXTX = P5; break;}
                case 0x06:    {I2CRXTX = P6; break;}
                case 0x10:    {I2CRXTX = P0PINCFG; break;}
                case 0x11:    {I2CRXTX = P1PINCFG; break;}
                case 0x12:    {I2CRXTX = P2PINCFG; break;}
                case 0x13:    {I2CRXTX = P3PINCFG; break;}
                case 0x14:    {I2CRXTX = P4PINCFG; break;}
                case 0x15:    {I2CRXTX = P5PINCFG; break;}
                case 0x16:    {I2CRXTX = P6PINCFG; break;}
                //Case of invalid address
                default:     {
                            I2CSTATUS |= 0x02;    //Force the I2C interface to not acknowledge
                                                  //after next data byte
                            while(!(I2CSTATUS & 0x08));   //wait for I2C IDLE
                            address = I2CRXTX;             //perform a dummy read
                            I2CRXTX = P0;                  //put P0 on I2CRXTX to prevent
                                                           //inadvertant I2CTX int
                            address = 0x00;           //reset address
                            state = 0x00;             //reset State
                            bytecount = 0x00;         //Reset byte counts
                            break;
                            }
```

```
            }//end of switch(address)
        state = 0x01;
        }//end of if state == 0x00

    else
        if(bytecount!= 0x00)
            {
            //rxdata = I2CRXTX;                   //Retrieve data

            switch(address)
                {
                case 0x00:    {P0 = I2CRXTX; I2CRXTX = P1; address++; break;}
                case 0x01:    {P1 = I2CRXTX; I2CRXTX = P2; address++; break;}
                case 0x02:    {P2 = I2CRXTX; I2CRXTX = P3; address++; break;}
                case 0x03:    {P3 = I2CRXTX; I2CRXTX = P4; address++; break;}
                case 0x04:    {P4 = I2CRXTX; I2CRXTX = P5; address++; break;}
                case 0x05:    {P5 = I2CRXTX; I2CRXTX = P6; address++; break;}
                case 0x06:    {P6 = I2CRXTX;
                               I2CRXTX = P0PINCFG;
                               address = 0x10;   //jump address value to point to P0PINCFG
                               break;}
                case 0x10:    {P0PINCFG = I2CRXTX;
                               I2CRXTX = P1PINCFG;
                               address++;
                               break;}
                case 0x11:    {P1PINCFG = I2CRXTX;
                               I2CRXTX = P2PINCFG;
                               address++;
                               break;}
                case 0x12:    {P2PINCFG = I2CRXTX;
                               I2CRXTX = P3PINCFG;
                               address++;
                               break;}
                case 0x13:    {P3PINCFG = I2CRXTX;
                               I2CRXTX = P4PINCFG;
                               address++;
                               break;}
                case 0x14:    {P4PINCFG = I2CRXTX;
                               I2CRXTX = P5PINCFG;
                               address++;
                               break
case 0x15:     {P5PINCFG = I2CRXTX;
                               I2CRXTX = P6PINCFG;
                               address++;
                               break;}
                case 0x16:   {P6PINCFG = I2CRXTX;
                               I2CRXTX = P0;
                               address = 0x00;
                               break;}
                }//end of switch
            }//end of if bytecount
    }//end of RX Available Flag handling

//---Case of I2C TX Empty available
if(status & 0x01)                   //TX Empty
    {
    //Data of current address already gone
    //must prepare Tx portion of I2CRXTXD for next data
     switch(address)
        {
        case 0x00:    {I2CRXTX = P1; break;}
        case 0x01:    {I2CRXTX = P2; break;}
        case 0x02:    {I2CRXTX = P3; break;}
        case 0x03:    {I2CRXTX = P4; break;}
        case 0x04:    {I2CRXTX = P5; break;}
        case 0x05:    {I2CRXTX = P6; break;}
        case 0x06:    {I2CRXTX = P0PINCFG; address = 0x10; break;}
        case 0x10:    {I2CRXTX = P1PINCFG; break;}
        case 0x11:    {I2CRXTX = P2PINCFG; break;}
        case 0x12:    {I2CRXTX = P3PINCFG; break;}
        case 0x13:    {I2CRXTX = P4PINCFG; break;}
        case 0x14:    {I2CRXTX = P5PINCFG; break;}
        case 0x15:    {I2CRXTX = P6PINCFG; break;}
        case 0x16:    {I2CRXTX = P0; address = 0xFF; break;}
        }//end of switch
        address++;
  }//End of If TX Empty...

    if((status & 0x06))                     //RX Overrun -- this should not happen
      i2cvalue = I2CRXTX;                   //empty RXTX buffer

    INTEN2 = 0x02;                          //Enable I2C Interrupt
    }//end of I2C interrupt


//---------------- Individual Functions ---------------------
//-----------------------------------------------------------------------//
// Auxiliary sub functions used by I2C functions
//-----------------------------------------------------------------------//

//--------  Function WaitTXEMP() ----------//
void WaitTXEMP()
    {
    wait();
    do{
    USERFLAGS = I2CSTATUS;
    USERFLAGS &= 0x01;                      //Isolate the I2CTXEMPTY flag
    }while( USERFLAGS == 0x00);             //Wait for I2C TX EMPTY
    }//end of Void WaitTXEMP()

//--------  Function WaitRXAV() ----------//
void WaitRXAV()
    {
    wait();
    do{
    USERFLAGS = I2CSTATUS;
    USERFLAGS &= 0x02;                      //isolate the I2CRXAV flag
    }while( USERFLAGS == 0x00);             //Wait for I2C RX AVAILLABLE
    }//end of Void WaitRXAV()

//--------  Function WaitI2CIDLE() ----------//
void WaitI2CIDLE()
    {
    wait();
    do{
        USERFLAGS = I2CSTATUS;
        USERFLAGS &= 0x08;                  //Isolate the I2C IDLE flag
    }while( USERFLAGS == 0x00);
    }//end of Void WaitI2CIDLE()

//--------  Function Wait() ----------//
void wait(){
    char i=0;
    while (i<25) {i++;};
    }
```

12 Pulse Width Modulators (PWMs)

The VRS51L3xxx devices include eight independent PWM channels, each based on a 16-bit timer.

All of the PWM modules can be configured to operate as a regular PWM with adjustable resolution, or as a general purpose 16-bit timer. The PWMEN register is used to enable the different PWM modules.

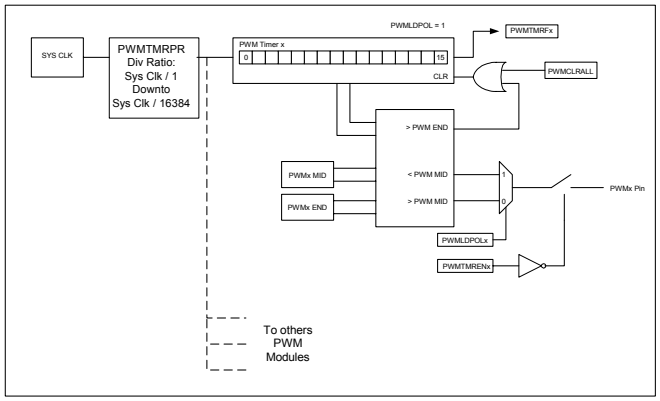
TABLE 124: PWM ENABLE REGISTER - PWMEN SFR AAH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PWM7EN	PWM7 Channel Enable 0 = PWM channel 7 is deactivated 1 = PWM channel 7 is activated
6	PWM6EN	PWM6 Channel Enable 0 = PWM channel 6 is deactivated 1 = PWM channel 6 is activated
5	PWM5EN	PWM5 Channel Enable 0 = PWM channel 5 is deactivated 1 = PWM channel 5 is activated
4	PWM4EN	PWM4 Channel Enable 0 = PWM channel 4 is deactivated 1 = PWM channel 4 is activated
3	PWM3EN	PWM3 Channel Enable 0 = PWM channel 3 is deactivated 1 = PWM channel 3 is activated
2	PWM2EN	PWM2 Channel Enable 0 = PWM channel 2 is deactivated 1 = PWM channel 2 is activated
1	PWM1EN	PWM1 Channel Enable 0 = PWM channel 1 is deactivated 1 = PWM channel 1 is activated
0	PWM0EN	PWM0 Channel Enable 0 = PWM channel 0 is deactivated 1 = PWM channel 0 is activated

The following figure provides an overview of the PWM modules.

FIGURE 38: PWM MODULES OVERVIEW



12.1 PWM MID and END registers

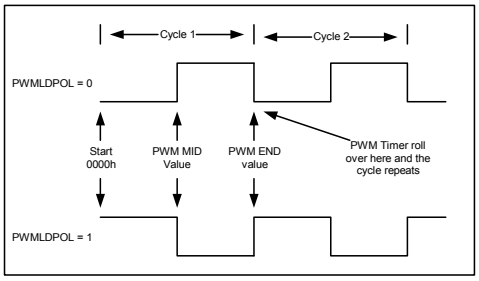
Each PWM module includes two 16-bit registers:

- PWM MID value register
- PWM END value register

The PWM MID register is a 16-bit register that configures the point at which the PWM output will change it's polarity.

The PWM END register is a 16-bit register that defines the maximum PWM internal timer count value, after which it rolls over to 0000h. See the following timing diagram.

FIGURE 39: PWM POLARITY SETTING



This configuration allows the user to adjust the resolution of the PWM up to 16 bits. Access to the PWM internal registers and the PWM configuration is handled by the PWMCFG register located at address A9h.

TABLE 125:PWM CONFIGURATION REGISTER - PWMCFG SFR A9H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	-	
6	PWMWAIT	PWM Waits Before Loading New Configuration 0 = New PWM configuration is loaded at the end of PWM cycle 1 = The update of the PWM configuration only occurs when the end of the PWM is reached and the bit is set to 0
5	PWMCLRALL	PWM Clears All Channels 0 = No Action 1 = Simultaneously clears all the flags and all the PWM channel timers This bit is automatically cleared by hardware
4	PWMLSBMSB	PWM LSB/MSB Select 0 = Selected PWM LSB SFR is addressed 1 = Selected PWM MSB SFR is addressed
3	PWMMIDEND	PWM MID/END Register 0 = Selected PWM MID SFR is addressed 1 = Selected PWM END SFR is addressed
2:0	PWMCH[2:0]	PWM Channel Select 000 = PWM0 on P2.0 or P5.0 001 = PWM1 on P2.1 or P5.1 010 = PWM2 on P2.2 or P5.2 011 = PWM3 on P2.3 or P5.3 100 = PWM4 on P2.4 or P5.4 101 = PWM5 on P2.5 or P5.5 110 = PWM6 on P2.6 or P5.6 111 = PWM7 on P2.7 or P5.7

The PWM channels are configured one at the time. This topology has been adopted in order to minimize the number of SFR registers required to access the PWM modules. In applications where multiple PWM channels need to be configured simultaneously, the user can set the PWMWAIT bit of the PWMCFG register, configure each one of the PWM channels, and then clear the PWMWAIT bit. The PWM configurations will then be updated at the end of the next PWM cycle, after the PWMWAIT bit has been cleared.

TABLE 126:PWM DATA REGISTER SFR ACH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	PWMDATA[7:0]	PWM Data Register

The PWM data register serves to configure the selected channel MSB/LSB value of either the MID or END point, as specified in the PWMCFG register.

The PWMIDx defines the actual timer value and the PWMEND defines the maximum timer count value before it rolls over.

The PWMLDPOL register controls the output polarity of each one of the PWM modules or clears the timer's value when the PWM modules operate as general purpose timers.

TABLE 127:PWM POLARITY AND CONFIG LOAD STATUS – PWMLDPOL ABH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PWMLDPOL7	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 7 cycle starts with a low level 1 = PWM 7 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 7 value is cleared to 0
6	PWMLDPOL6	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 6 cycle starts with a low level 1 = PWM 6 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 6 value is cleared to 0
5	PWMLDPOL5	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 5 cycle starts with a low level 1 = PWM 5 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 5 value is cleared to 0
4	PWMLDPOL4	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 4 cycle starts with a low level 1 = PWM 4 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 4 value is cleared to 0
3	PWMLDPOL3	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 3 cycle starts with a low level 1 = PWM 3 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 3 value is cleared to 0
2	PWMLDPOL2	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 2 cycle starts with a low level 1 = PWM 2 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 2 value is cleared to 0

1	PWMLDPOL1	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 1 cycle starts with a low level 1 = PWM 1 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 1 value is cleared to 0
0	PWMLDPOL0	**Read:** 0 = Last configuration has been loaded in PWM 1 = Last configuration has not been loaded
		Write **In PWM Mode** 0 = PWM 0 cycle starts with a low level 1 = PWM 0 cycle starts with a high level **In Timer Mode** 0 = No action 1 = PWM timer 0 value is cleared to 0

12.2 PWM Module Clock Configuration Register

One system clock prescaler is associated with PWM modules 0 to 3, while another is associated with PWM modules 4 to 7. The PWM clock prescalers enables the PWM output frequency to be adjusted to match specific application needs, if required. The PWM clock prescalers are configured via the PWMCLKCFG register. The four upper bits of this register control the clock for PMM modules 4 to 7, and the four lower bits control the clock source for PWM modules 0 to 3.

The PWM module clock configuration register controls the prescale value applied to the PWM modules' input clock, when the PWM modules are configured to operate as either PWMs or general purpose timers.

TABLE 128: PWM CLOCK PRESCALER CONFIGURATION REGISTER - PWMCLKCFG AFH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:4	U4PWMCLK3[3:0]	PWM Timer 7, 6, 5,:4 Clock Prescaler * see table
3:0	L4PWMCLK3[3:0]	PWM Timer 3, 2, 1,:0 Clock Prescaler * see table

The following table shows the system clock division factor applied to the PWM modules for a given PWMCLKCFG nibble.

TABLE 129: PWM PRESCALER VALUES

U4/L4PWMCLK Value (4 bit)	Clock Prescaler	U4/L4PWMCLK Value (4 bit)	Clock Prescaler
0000	Sys Clk / 1	1000	Sys Clk / 256
0001	Sys Clk / 2	1001	Sys Clk / 512
0010	Sys Clk / 4	1010	Sys Clk / 1024
0011	Sys Clk / 8	1011	Sys Clk / 2048
0100	Sys Clk / 16	1100	Sys Clk / 4096
0101	Sys Clk / 32	1101	Sys Clk / 8192
0110	Sys Clk / 64	1110	Sys Clk/ 16384
0111	Sys Clk / 128	1111	Sys Clk/ 16384

12.3 PWM Alternate Mapping

On the VRS51L30xxx (QFP-64) it is possible to output the PWM on the I/O port 5. the I/O port 5 has a greater current drive capability than I/O port 2. The Bit 6 of the DEVIOMAP register (SFR E1h) controls the mapping of the PWM module outputs, as shown in the following table:

TABLE 130: PWM MODULES OUTPUT MAPPING

DEVIOMAP.6 Bit Value	PWM 7-0
0 (Reset)	P2.7 – P2.0
1	P5.7 – P5.0**

**On VRS51L30xx devices only

Note that the PWM5 and PWM6 outputs have priority over the T0EX and T1EX inputs.

12.4 PWM Example Programs

Code Example:
PWM Basic Configuration

The following example program shows the basic configuration of PWM modules #0, 1,2, 4 & 5

```
//-------------------------------------------------------------------------------------------------//
// VRS51L3074-PWM_basic_SDCC.c  //
//-------------------------------------------------//
//
// DESCRIPTION:    VRS51L3074 PWMs Basic initialization Demonstration Program.
//              Configure PWM0 as 8 bit resolution  (25% duty)
//              Configure PWM1 as 12 bit resolution  (50% duty)
//              Configure PWM2 as 16 bit resolution   (75% duty)
//              Configure PWM4 as 8 bit resolution and prescaler = 4 (25% duty)
//              Configure PWM5 as 16 bit resolution and prescaler = 4 (75% duty)
//-------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

// --- function prototypes

void delay(unsigned int);

void main (void) {
    PERIPHEN2 = 0x02;               //Enable PWM SFR

    //CLEAR All PWM Channels
    PWMCFG = 0x20;

    // Configure the PWM prescaler
    PWMCLKCFG = 0x20;               // Apply a clock prescaler (div / 4) on PWM 7:4

    // Configure PWM Polarity
    PWMPOL = 0x00;                  //Set all PWM in normal polarity
                                    //PWM output = 0 until
                                    //PWMMID Value is reached

    //---------------------------------//
    //Configure PWM0 END value = 0x00FF (8bit)
    PWMCFG = 0x58;                  //Point to PWM0 END MSB
    PWMDATA = 0x00;                 //Set Max Count MSB = 0xFF
    PWMCFG = 0x48;                  //Point to PWM0 END LSB
    PWMDATA = 0xFF;                 //Set PWM MID MSB = 0x00 (8bit)


    //Configure PWM0 MID value (Duty = 25%)
    PWMCFG = 0x50;                  //Point to PWM0 MID MSB
    PWMDATA = 0x00;                 //Set PWM MID MSB = 0x00
    PWMCFG = 0x40;                  //Point to PWM0 MID LSB
    PWMDATA = 0xBF;                 //Set PWM MID LSB = 0xBF

    //---------------------------------//
    //Configure PWM1 END value = 0x0FFF (12bit)
    PWMCFG = 0x59;                  //Point to PWM1 END MSB
    PWMDATA = 0x0F;                 //Set Max Count MSB = 0x0F
    PWMCFG = 0x49;                  //Point to PWM1 END LSB
    PWMDATA = 0xFF;                 //Set Max Count = 0xFF

    //Configure PWM1 MID value (Duty = 50%)
    PWMCFG = 0x51;                  //Point to PWM0 MID MSB
    PWMDATA = 0x08;                 //Set PWM MID MSB = 0x08
    PWMCFG = 0x41;                  //Point to PWM0 MID LSB
    PWMDATA = 0x00;                 //Set PWM MID LSB = 0x00

    //---------------------------------//
    //Configure PWM2 END value = 0xFFFF (16bit)
    PWMCFG = 0x5A;                  //Point to PWM2 END MSB
    PWMDATA = 0xFF;                 //Set Max Count MSB = 0xFF
    PWMCFG = 0x4A;                  //Point to PWM2 END LSB
    PWMDATA = 0xFF;                 //Set Max Count = 0xFF

    //Configure PWM2 MID value (duty = 75%)
    PWMCFG = 0x52;                  //Point to PWM2 MID MSB
    PWMDATA = 0x40;                 //Set PWM MID MSB = 0x04
    PWMCFG = 0x42;                  //Point to PWM2 MID LSB
    PWMDATA = 0x00;                 //Set PWM MID LSB = 0x00


    //---------------------------------//
    //Configure PWM4 END value = 0x00FF (8 bit) (Clock Prescaler = 4)
    PWMCFG = 0x5C;                  //Point to PWM4 END MSB
    PWMDATA = 0x00;                 //Set Max Count MSB = 0xFF
    PWMCFG = 0x4C;                  //Point to PWM4 END LSB
    PWMDATA = 0xFF;                 //Set Max Count LSB = 0xFF

    //Configure PWM4 MID value (duty = 25%)
    PWMCFG = 0x54;                  //Point to PWM4 MID MSB
    PWMDATA = 0x00;                 //Set PWM MID MSB = 0x00
    PWMCFG = 0x44;                  //Point to PWM4 MID LSB
    PWMDATA = 0xBF;                 //Set PWM MID LSB = 0xBF

    //---------------------------------//
    //Configure PWM5 END value = 0xFFFF (16bit)   (Clock Prescaler = 4)
    PWMCFG = 0x5D;                  //Point to PWM5 END MSB
    PWMDATA = 0xFF;                 //Set Max Count MSB = 0xFF
    PWMCFG = 0x4D;                  //Point to PWM5 END LSB
    PWMDATA = 0xFF;                 //Set Max Count = 0xFF

    //Configure PWM5 MID value (duty = 75%)
    PWMCFG = 0x55;                  //Point to PWM5 MID MSB
    PWMDATA = 0x40;                 //Set PWM MID MSB = 0x04
    PWMCFG = 0x45;                  //Point to PWM5 MID LSB
    PWMDATA = 0x00;                 //Set PWM MID LSB = 0x00

    //Enable PWM0, PWM1, PWM2, PWM4 & PWM5 Modules
    PWMEN = 0x37;


    PWMCFG &= 0x1F;                 //Clear the PWMWAIT bit to initiate
                                    //the PWMs operation
    while(1);

}// End of main
```

Code Example:
PWM Configuration and Control Functions

```
//-------------------------------------------------------------------------------------------------//
// VRS51L3074-PWM_CFG_function_SDCC.c                                                              //
//-------------------------------------------------------------------------------------------------//
// DESCRIPTION:    PWM configuration and control Functions
//-------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

// --- functions prototypes
void PWMConfig(char channel,int endval,int midval);
void PWMdata8bit(char,char);
void PWMdata16bit(char,int);
void delay(unsigned int);

void delay(unsigned int);

void main (void) {
     int cptr = 0x00;

  //  PERIPHEN2 = 0x02;               //Enable PWM SFR

    //CLEAR All PWM Channels
    PWMCFG = 0x20;

    // Configure the PWM prescaler
    PWMCLKCFG = 0x00;               // Apply a clock prescaler (div / 1) on all PWM

    // Configure PWM Polarity
    PWMLDPOL = 0x00;                //Set all PWM in normal polarity
                                    //PWM output = 0 until

    //--Configure PWM5 as 8bit resolution,  END = 0xFF, PWM MID = 0x000
   PWMConfig(0x05, 0x0FF,0x000);

   //--Configure PWM0 as 8bit resolution,  END = 0xFFF, PWM MID = 0x0000
    PWMConfig(0x02, 0xFFF,0x000);

    //Continuously vary the PWM2 and PWM5 values

    do{
          for(cptr = 0xFF0; cptr > 0x00; cptr--)
             {
             PWMdata16bit(0x02,cptr);
             PWMdata8bit(0x05,cptr>>4);
             delay(1);
             }
    }while(1);
}// End of main
```

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

//-------------------------------------------------------------------------------------//
// -- PWMConfig
// -------------------------------------------------------------------------------------//
// Description: configure PWM channel
//-------------------------------------------------------------------------------------//
void PWMConfig(char channel,int endval,int midval)
    {
    char pwmch;
    char pwmready = 0x00;

    channel &= 0x07;                  //Make sure PWM ch number <= 7

    //Wait Last configuration to be loaded
    do{
        pwmready = PWMLDPOL;
        }while(pwmready != 0x00);

    //Define PWM Enable section

    PERIPHEN2 |= 0x02;                //Enable PWM SFR
    //--Define the value to put into the PWMEN register
    switch(channel)
        {
        case 0x00 : pwmch = 0x01;
                break;
        case 0x01 : pwmch = 0x02;
                break;
        case 0x02 : pwmch = 0x04;
                break;
        case 0x03 : pwmch = 0x08;
                break;
        case 0x04 : pwmch = 0x10;
                break;
        case 0x05 : pwmch = 0x20;
                break;
        case 0x06 : pwmch = 0x40;
                break;
        case 0x07 : pwmch = 0x80;
                break;
        }//end of switch

     PWMEN |= pwmch;                    //Enable the Selected channel

    //Configure PWM END point

    PWMCFG = (channel + 0x58);          //Set PWM configuration register to point to
                                        //the MSB of End value and set the PWMWAIT bit
                                        //to prevent the PWM configuration to be loaded
                                        //before the configure sequence is completed
    PWMDATA = endval >> 8;

    PWMCFG &= 0xEF;                     //Set PWM configuration register to point to
                                        //the LSB of End value
    PWMDATA = endval;

    //Configure PWM MID point

    PWMCFG = (channel + 0x50);          //Set PWM configuration register to point to
                                        //the MSB of MID value and set the PWMWAIT bit
                                        //to prevent the PWM configuration to be loaded
                                        //before the configure sequence is completed
    PWMDATA = midval >> 8;

    PWMCFG &= 0xEF;                     //Set PWM configuration register to point to
                                        //the LSB of End value
    PWMDATA = midval;

    PWMCFG &= 0x3F;                     //Allows PWM update upon end of next PWM cycle

    }//end of PWMData16bit()
```

```
//-------------------------------------------------------------------------------------//
// -- PWMdata8bit
// -------------------------------------------------------------------------------------//
// Description: Allow PWM channel data update
//                  ( 8bit data )l
//-------------------------------------------------------------------------------------//
void PWMdata8bit(char channel,char pwmdata)
    {
    channel &= 0x07;                    //Make sure PWM ch number <= 7

    //--check that te last configuration has been loaded


    PWMCFG = (channel + 0x40);          //Write new value in PWM Config
                                        //prevent PWM configuration to be loaded
                                        //before the configure sequence is completed
    PWMDATA = pwmdata;                  //Write new Data into the PWM registers

    PWMCFG &= 0x3F;                     //Allows PWM update upon end of next PWM cycle

    }//end of PWMData8bit()


//-------------------------------------------------------------------------------------//
// -- PWMdata16bit
// -------------------------------------------------------------------------------------//
// Description: Allow PWM channel data update
//                  ( 16bit data )l
//-------------------------------------------------------------------------------------//
void PWMdata16bit(char channel,int pwmdata)
    {


    channel &= 0x07;                    //Make sure PWM ch number <= 7
    PWMCFG = (channel + 0x50);          //Set PWM configuration register to point to
                                        //the MSB of Data value and set the PWMWAIT bit
                                        //and set the PWMWAIT bit to prevent the
                                        //PWM configuration to be loaded
                                        //before the configure sequence is completed
    PWMDATA = pwmdata >>8;

    PWMCFG &= 0xEF;                     //Set PWM configuration register to point to
                                        //the LSB of Data value
    PWMDATA = pwmdata;

    PWMCFG &= 0x3F;                     //Allows PWM update upon end of next PWM cycle

    }//end of PWMData16bit()


//;-------------------------------------------------------------------------------------//
//;- DELAY1MSTO : 1MS DELAY USING TIMER0                                                //
//;                                                                                    //
//; CALIBRATED FOR 40MHZ                                                               //
//;-------------------------------------------------------------------------------------//
void delay(unsigned int dlais){

idata unsigned char x=0;
idata unsigned int dlaisloop;

    x = PERIPHEN1;                  //LOAD PERIPHEN1 REG
    x |= 0x01;                      //ENABLE TIMER 0
    PERIPHEN1 = x;

    dlaisloop = dlais;
    while ( dlaisloop > 0)
      {
       TH0 = 0x63;                  //TIMER0 RELOAD VALUE FOR 1MS AT 40MHZ
       TL0 = 0xC0;

       T0T1CLKCFG = 0x00;           //NO PRESCALER FOR TIMER 0 CLOCK
       T0CON = 0x04;                //START TIMER 0,  COUNT UP

        do{
            x=T0CON;
            x= x & 0x80;
            }while(x==0);

        T0CON = 0x00;               //Stop Timer 0
        dlaisloop = dlaisloop-1;
        }//end of while dlais...

       x = PERIPHEN1;               //LOAD PERIPHEN1 REG
       x = x & 0xFE;                //DISABLEBLE TIMER 0
       PERIPHEN1 = x;
    }//End of function delais
```

12.5 Using PWM Modules as Timers

By appropriately configuring the PWMTMREN SFR, the PWM modules can also operate as general purpose 16-bit timers. The following table describes the PWMTMREN register:

TABLE 131: PWM TIMER MODE ENABLE REGISTER - PWMTMREN SFR ADH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PWM7TMREN	PWM 7 Module Operating Mode 0 = PWM 7 module is configured as PWM 1 = PWM 7 module is configured as timer
6	PWM6TMREN	PWM 6 Module Operating Mode 0 = PWM 6 module is configured as PWM 1 = PWM 6 module is configured as timer
5	PWM5TMREN	PWM 5 Module Operating Mode 0 = PWM 5 module is configured as PWM 1 = PWM 5 module is configured as timer
4	PWM4TMREN	PWM 4 Module Operating Mode 0 = PWM 4 module is configured as PWM 1 = PWM 4 module is configured as timer
3	PWM3TMREN	PWM 3 Module Operating Mode 0 = PWM 3 module is configured as PWM 1 = PWM 3 module is configured as timer
2	PWM2TMREN	PWM 2 Module Operating Mode 0 = PWM 2 module is configured as PWM 1 = PWM 2 module is configured as timer
1	PWM1TMREN	PWM 1 Module Operating Mode 0 = PWM 1 module is configured as PWM 1 = PWM 1 module is configured as timer
0	PWM0TMREN	PWM 0 Module Operating Mode 0 = PWM 0 module is configured as PWM 1 = PWM 0 module is configured as timer

When operating in timer mode, the PWM module timer will count from 0000h up to the maximum PWM timer value defined by the PWM MID sub registers, which are accessible through the PWMCFG register.

TABLE 132: SUMMARY OF PWM MID SUB REGISTERS ACCESS

	PWMCFG bit PWMLSBMSB	PWMCFG bit PWMMIDEND
PWM timer MSB max count value	0	1
PWM timer MSB max count value	1	1

Once the PWM MID value is reached, the PWM timer overflow is set and the PWM timer rolls over to 0000h.

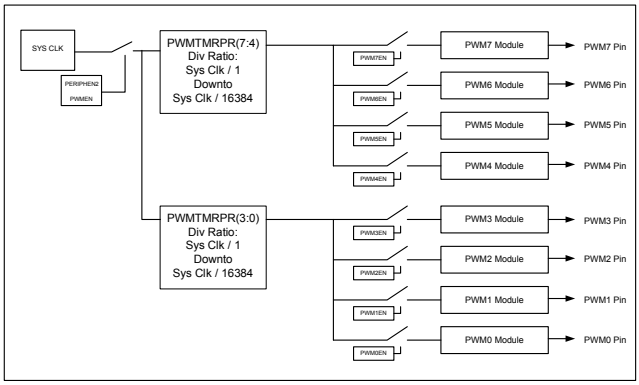
The PWM timer flags are raised when the timer reaches the maximum value set by PWMMIDH and PWMMIDL. The PWMxTMRF bit must be cleared manually by the interrupt service routine.

TABLE 133: PWM TIMER FLAGS REGISTER - PWMTMRF SFR AEH

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PWM7TMRF	PWM 7 Module Timer Flag 0 = No Overflow 1 = PWM Timer 7 Overflow
6	PWM6TMRF	PWM 6 Module Timer Flag 0 = No overflow 1 = PWM Timer 6 Overflow
5	PWM5TMRF	PWM 5 Module Timer Flag 0 = No Overflow 1 = PWM Timer 5 Overflow
4	PWM4TMRF	PWM 4 Module Timer Flag 0 = No Overflow 1 = PWM Timer 4 Overflow
3	PWM3TMRF	PWM 3 Module Timer Flag 0 = No Overflow 1 = PWM Timer 3 Overflow
2	PWM2TMRF	PWM 2 Module Timer Flag 0 = No Overflow 1 = PWM Timer 2 Overflow
1	PWM1TMRF	PWM 1 Module Timer Flag 0 = No Overflow 1 = PWM Timer 1 Overflow
0	PWM0TMRF	PWM 0 Module Timer Flag 0 = No Overflow 1 = PWM Timer 0 Overflow

FIGURE 40: PWM AS TIMERS OVERVIEW



12.6 Configuring the PWM Timers

Configuring the PWM modules to operate in PWM timer mode requires the following steps:

1. Activate the PWMSFR register
2. Configure the PWM clock prescaler (if required)
3. Set the PWMLDPOL register to 00h
4. Configure the PWM timer maximum count value by setting the PWM MID sub-registers
5. Configure the PWM timer interrupts (if required)
6. Configure the PWM modules as timers
7. Enable the PWM modules

Follow the code example below to perform these seven steps :

```
(…)
    PERIPHEN2 |= 0x02;              //Enable PWM SFR

    //--Configure the PWM prescaler
    PWMCLKCFG = 0x03;               //Apply a clock prescaler (div / 8) on PWM 3:0

    //--Configure PWM Polarity
    PWMLDPOL = 0x00;                //Set all PWM in normal polarity
                                    //PWM output = 0 until

    //--Configure PWM5 as timer
    //  PWM Timer 5 counts from 0000 to F000h
    PWMCFG =  0x15;                 //Point to MSB MID
    PWMDATA = 0xF0;                 //Set PWM as Timer Max MSB

    PWMCFG =  0x05;                 //Point to LSB MID
    PWMDATA = 0x00;                 //Set PWM as Timer Max LSB

    //--Configure and Enable PWM as timer Interrupt  to monitor PWM5 only
    INTSRC2 &= 0xDF;                //PWM7:4 Timer module interrupt
    INTPINSENS1 = 0xDF;             // sensitive on high level(0)
    INTPININV1 = 0xDF;              //Set INT0 Pin sensitivity on normal level(0)
    INTEN2 |= 0x20;                 //Enable PWM7:4 Timer module interrupt

    //--Activate the PWM module and cofigure the PWM modules 5 as timer
    PWMEN |= 0x20;                  //Enable PWM 5
    PWMTMREN |= 0x20;               //Enable PWM 5 as Timer
    GENINTEN = 0x03;                //Enable Global interrupt
```

12.7 PWMs as Timers Example Programs

Code Example:

Configuring PWM0 and PWM5 as Timers

The following example program demonstrates how to initialize PWM0 and PWM5 as general purpose timers, and how to monitor the PWM timer's overflow flags by pooling or via an interrupt.

```
//-------------------------------------------------------------------------------------------------------//
// VRS51L3074-PWM_as_Timer1_SDCC.c.c
//-------------------------------------------------------------------------------------------------------//
// DESCRIPTION:  PWM as Timer Example Program
//               Enable and configure PWM Timer 0
//               Apply a clock prescaler on PWM Timer 0 (div/8)
//               Enable and configure PWM Timer 5
//               Monitor PWM Timer 0 OV Flag by pooling
//               When PWM Timer 0 Overflow, toggle P1.0 pin
//               Monitor PWM Timer 5 OV Flag by interrupt
//               When PWM Timer 5 Overflow interrupt occurs toggle P1.5 pin
///------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>
void main (void) {
    int cptr = 0x00;
    char flagread;

    PERIPHEN2 |= 0x02;              //Enable PWM SFR

    //Configure Port1 as output

    P1PINCFG = 0x00;
    //Clear All PWM Channels
   // PWMCFG = 0x20;

    // Configure the PWM prescaler
    PWMCLKCFG = 0x03;               // Apply a clock prescaler (div / 8) on PWM 3:0

    // Configure PWM Polarity
    PWMLDPOL = 0x00;                //Set all PWM in normal polarity
                                    //PWM output = 0 until

    //--Configure PWM0 as Timer  (will be monitored by pooling)
    //  PWM Timer 0 counts from 0000 to 01F0h

    PWMCFG =  0x10;                 //Point to MSB MID
    PWMDATA = 0x01;

    PWMCFG =  0x00;                 //Point to LSB MID
    PWMDATA = 0xF0;

    //--Activate the PWM modules and configure the PWM modules as timers
    PWMEN |= 0x01;
    PWMTMREN |= 0x01;               //Enable PWM 0 as Timer

    //--Configure PWM5 as Timer  (will be monitored by interrupt)
    //  PWM Timer 5 counts from 0000 to F000h
    PWMCFG =  0x15;                 //Point to MSB MID
    PWMDATA = 0xF0;                 //

    PWMCFG =  0x05;                 //Point to LSB MID
    PWMDATA = 0x00;

    //--Configure and enable PWM as timer interrupt  to monitor PWM5 only
    INTSRC2 &= 0xDF;                //PWM7:4 Timer module interrupt
    INTPINSENS1 = 0xDF;             // sensitive on high level(0)
    INTPININV1 = 0xDF;              //Set INT0 Pin sensitivity on normal level(0)
    INTEN2 |= 0x20;                 //Enable PWM7:4 timer module interrupt

    //--Activate the PWM modules and configure the PWM modules as timers
    PWMEN |= 0x20;                  //Enable PWM 5
    PWMTMREN |= 0x20;               //Enable PWM 5 as Timer

    GENINTEN = 0x03;                //Enable global interrupt

   while(1){

   //Wait for PWM0 as timer overflow Flag  PWM0 timer flag pooled
   do
       {
       flagread = PWMTMRF;
       flagread &=0x01;
       }while(flagread == 0);

   PWMTMRF &= 0xFE;                 //Clear the PWM0 Timer Flag
   P1 = P1^0x01;                    //Toggle P1.0
   }//end of while(1)
}// End of main

//-------------------------------------------------------------- //
//----- Interrupt INT13 - PWM7:4 as Timer  //
//--------------------------------------------------------------//
void INT13Interrupt(void) interrupt 13
    {
    char flagread;

    INTEN2 = 0x00;                  //Disable PWM7:4 Timer module interrupt

    flagread = PWMTMRF;             //Read PWM Timer OV Flags
    flagread &= 0x20;               //Check if PWM Timer 5 OV Flag is active
    if(flagread != 0x00)
      P1 = P1^0x20;                 //Toggle P1.5

    PWMTMRF &= 0xDF;                //Clear the PWM Timer 5 OV Flag

    INTEN2 |= 0x20;         //Enable PWM7:4 Timer module interrupt
    }//end of INT0 interrupt
```

13 Enhanced Arithmetic Unit

All members of the VRS51L3xxx device family include a hardware-based, calculation engine that executes very fast arithmetic operations. With the exception of 16-bit division, which requires 5 cycles, the enhanced arithmetic unit performs multiplication, addition and data shifting in 1 system clock cycle.

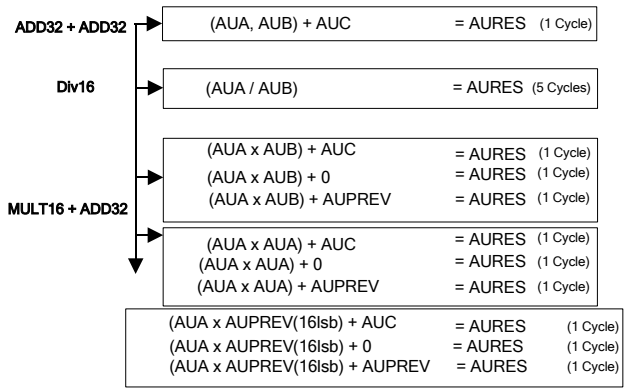
This enables a tremendous performance gain of approximately 30% to 50% for multiplication and accumulation and 700% faster for 16-bit division compared to a standard C compiler when implementing mathematical and digital signal processing (DSP) operations.

The enhanced arithmetic 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 length
- Auto/manual reload of AU result register
- Easy implementation of complex mathematical operations
- 16-bit and 32-bit overflow flag
- 32-bit overflow can set an interrupt
- Arithmetic unit operand registers can be cleared individually or simultaneously
- Overflow flags can be configured to stay active until manually cleared
- Can store and use results from previous operations
- Hardware arithmetic unit features a 32-bit barrel shifter in front of the AURES register, which can be employed to scale up/down the result of the operation being performed
- Data shifting operation is performed within the 1 cycle required for multiplication/addition

The arithmetic unit can be configured to perform the operations in the following figure. It can also perform data shifting.

FIGURE 41: VRS51L3XXX ARITHMETIC UNIT OPERATION



Where AUA (multiplier), AUB (multiplicand), AUC (accumulator), AURES (result) and AUPREV (previous result) are 16-, 16-, 32-, 32- and 32-bits wide, respectively.

Applications that require arithmetic and DSP operations will benefit from the execution of such calculations on the enhanced arithmetic unit. These include digital filtering, data encryption, sensor output data processing, lookup table replacement, etc. More specifically, applications like FIR filtering that require the repeated execution of 16-bit multiplication and accumulation will benefit tremendously from the arithmetic unit.

13.1 Using the Enhanced Arithmetic Unit

The enhanced arithmetic unit operates in signed binary. Access to its registers is executed via the SFR registers, located on SFR Page 1. This page is accessed by setting the SFRPAGE bit of DEVMEMCFG register to 0x01. The DEVMEMCFG register is located at address F6h on both SFR pages.

Before accessing the enhanced arithmetic unit SFR registers, the module must be enabled. This is done by setting AUEN bit 5 of the PERIPHEN2 register to 1. AUEN bit 5 is located at address F5h on both SFR pages.

13.2 Arithmetic Unit Control Registers

With the exception of the barrel shifter, the arithmetic unit's operation is controlled by two SFR registers:

- AUCONFIG1
- AUCONFIG2

The following tables describe these control registers:

TABLE 134: ARITHMETIC CONFIG REGISTER 1 – AUCONFIG1 SFR C2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	CAPPREV	Read: Always Read as 0
		Capture Previous Result Enable 0 = Previous result capture is disabled 1 = Capture the previous result if CAPMODE bit is set to 1
6	CAPMODE	0 = The capture of previous result is automatic each time a write operation is done to the AU0 1 = The capture of the previous result is manual and occurs when the CAPPREV bit is set to 1
5	OVCAPEN	Capture Result on 32-Bit Overflow 0 = No result capture is performed 1 = The AU result is captured and stored when a 32-bit overflow condition occurs
4	READCAP	Read Stored Result 0 = AURES contains current operation result 1 = AURES contains previous result
3:2	ADDSRC[1:0]	AU Adder Input n 32-bit Addition Source **B Input** 00 = 0 (No Add) **01 = C (std 32-bit reg)** 10 = AUPREV 11 = AUC (std 32-bit reg) **A Input** 00=Multiplication 01=Multiplication 10=Multiplication 11= Concatenation of {A, B} + C for 32-bit addition
1:0	MULCMD[1:0]	AU Multiplication Command 00 = AUA x AUB 01 = AUA x AUA 10 = AUA x AUPREV (16 LSB) 11 = AUA x AUB Notes In Divider Mode MULTA_IN = MULT_IN = 0x0000 In Multiplier Mode DIVA_IN = 0x0000 and DIVB_IN = 0x0001

TABLE 135: ARITHMETIC CONFIG REGISTER 2 – AUCONFIG2 SFR C3H

7	6	5	4	3	2	1	0
W	W	W	R/W	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:5	AUREGCLR [2:0]	Read: Always read as 0 Arithmetic Unit Operand Registers Clear 000 = No clear 001 = Clear AUA 010 = Clear AUB 011 = Clear AUC 100 = Clear AUPREV 101 = Clear all AU module registers and overflow flags 110 = Clear overflow flags only
4	AUINTEN	Arithmetic Unit Interrupt Enable 0 = Arithmetic unit interrupt is disabled 1 =-Arithmetic unit interrupt is enabled in divider mode
3	-	Not used, Read as 0
2	DIVOUTRG	AU division is out of range flag This flag is set if AUB = 0x0000 or (AUA = 0x8000 and AUB = 0xFFFF)
1	AUOV16	Arithmetic Unit 16-Bit Overflow Flag 0 = No 16 bit overflow condition detected 1 = a 16-bit overflow occurred Will occur if there is a carry on from bit 15 to bit 1,6 but also from bit 31 to bit 32
0	AUOV32	Arithmetic Unit 32-Bit Overflow Flag 0 = No 16 bit overflow condition detected 1 = Operation result is larger than 32 bits

13.3 Arithmetic Unit Data Registers

The arithmetic unit 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 AUC), while others can be used for all operations. The use of the arithmetic unit operation registers is described in the following sections.

13.4 AUA and AUB Multiplication (Addition) Input Registers

The AUA and AUB registers serve as 16-bit input operands when performing multiplication.

When the arithmetic unit is configured to perform 32-bit addition, the AUA and the AUB registers are concatenated. In this case, the AUA register contains the upper 16 bits of the 32-bit operand and the AUB contains the lower 16 bits.

TABLE 136: ARITHMETIC UNIT A REGISTER BIT [7:0] - AUA0 SFR A2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUA[7:0]	LSB of the A Operand Register

TABLE 137: ARITHMETIC UNIT A REGISTER BIT [15:8]- AUA1 SFR A3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUA[15:8]	MSB of the A Operand Register

TABLE 138:ARITHMETIC UNIT B REGISTER BIT [7:0] - AUB0 SFR B2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUB[7:0]	LSB of the B Operand Register for Multiplication and Addition Operations

TABLE 139:ARITHMETIC UNIT DIVISION MODE REGISTER – AUB0DIV SFR B1H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUB0DIV[7:0]	Writing to this byte instead of AUB0 will set the arithmetic unit to divisor mode

TABLE 140: ARITHMETIC UNIT B REGISTER BIT [15:8] - AUB1 SFR B3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUB[15:8]	MSB of the B Operand Register

13.5 AUC Input Register

The AUC register is a 32-bit register used to perform 32-bit addition. The AUPREV register can be substituted with the AUC register or by 0 in the 32-bit addition.

TABLE 141:ARITHMETIC UNIT C REGISTER BIT [7:0] - AUC0 SFR A4H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUC[7:0]	Bit [7:0]of the C Operand Register

TABLE 142: ARITHMETIC UNIT C REGISTER BIT [15:8] - AUC1 SFR A5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUC[15:8]	Bit [15:8] of the C Operand Register

TABLE 143:ARITHMETIC UNIT C REGISTER BIT [23:16] - AUC2 SFR A6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUC[23:16]	Bit [23:16] of the C Operand Register

TABLE 144:ARITHMETIC UNIT C REGISTER BIT [31:24] – AUC3 SFR A7H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:0	AUC[31:24]	Bit [31:24] of the C Operand Register

13.6 The Arithmetic Unit AURES Register

The AURES register, which is 32 bits wide, is read-only and contains the result of the last arithmetic unit operation. The AURES register is located at the output of the barrel shifter.

When the arithmetic unit is configured to perform multiplication and/or addition, the AURES operates as a 32-bit register that contains the result of the previous operation(s).

However when the arithmetic unit has performed a 16-bit division, the upper 16 bits of the AURES register contain the quotient of the operation, while the lower 16 bits contain the remainder of the division operation.

The barrel shifter is deactivated when the arithmetic unit is performing 16-bit division.

Four SFR registers located in SFR Page 1 provide access to the arithmetic unit AURES register.

TABLE 145: ARITHMETIC UNIT RESULT REGISTER BIT [7:0] - AURES0 SFR B4H

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

Bit	Mnemonic	Description
7:0	AURES[7:0]	Bit [7:0]of the RESULT Register

TABLE 146: ARITHMETIC UNIT RESULT REGISTER BIT [15:8] - AURES1 SFR 5H

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

Bit	Mnemonic	Description
7:0	AURES[15:8]	Bit [15:8] of the RESULT Register

TABLE 147: ARITHMETIC UNIT RESULT REGISTER BIT [23:16] – AURES2 SFR B6H

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

Bit	Mnemonic	Description
7:0	AURES[23:16]	Bit [23:16] of the RESULT Register

TABLE 148: ARITHMETIC UNIT RESULT REGISTER BIT [31:24] – AURES3 SFR B7H

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

Bit	Mnemonic	Description
7:0	AURES[31:24]	Bit [31:24] of the RESULT Register

13.7 AUPREV Register

The AUPREV register can automatically or manually save the contents of the AURES 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 for the next one.

As previously mentioned, there are two ways to load the AUPREV register. This is controlled by the CAPMODE bit value as follows:

CAPMODE = 0:
Auto AUPREV load, by writing into the AUA0 register. Selected when CAPPREV = 0.

CAPMODE = 1:
Manual load of AUPREV when the CAPPREV bit is set to 1.

Auto loading of the AUPREV register is useful in FIR filter calculations. For example, it is possible to save a total of eight MOV operations per tap calculation.

TABLE 149: ARITHMETIC UNIT PREVIOUS RESULT BIT [7:0] - AUPREV0 SFR C4H

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

Bit	Mnemonic	Description
7:0	AUPREV[7:0]	Bit [7:0]of the Previous Result Register

TABLE 150:ARITHMETIC UNIT PREVIOUS RESULT BIT [15:8] - AUPREV1 SFR C5H

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

Bit	Mnemonic	Description
7:0	AUPREV[15:8]	Bit [15:8] of the Previous Result Register

TABLE 151:ARITHMETIC UNIT PREVIOUS RESULT BIT [23:16] – AUPREV2 SFR C6H

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

Bit	Mnemonic	Description
7:0	AUPREV[23:16]	Bit [23:16] of the Previous Result Register

TABLE 152:ARITHMETIC UNIT PREVIOUS RESULT BIT [31:24] – AUPREV3 SFR C7H

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

Bit	Mnemonic	Description
7:0	AUPREV[31:24]	Bit [31:24] of the Previous Result Register

13.8 Multiplication and Accumulate Operations

The multiplication and accumulate operations of the arithmetic unit are defined by the MULCMD[1:0] and ADDSRC[1:0] bits of the AUCONFIG1 register.

TABLE 153: MULTIPLICATION OPERATIONS VS. MULCMD BIT OF THE AUCONFIG1

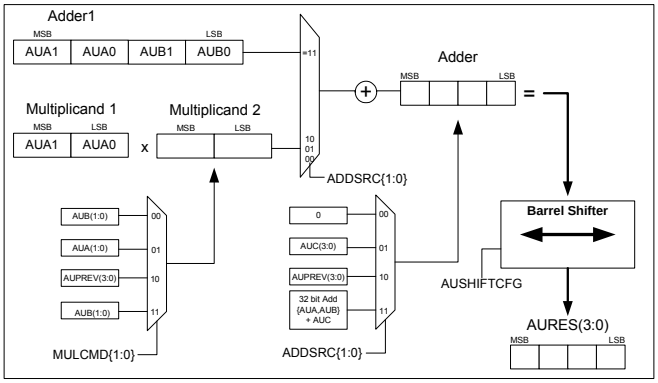
MULCMD[1:0]	Multiplication Operation
00	AUA x AUB
01	AUA x AUA
10	AUA x AUPREV (16LSB)
11	AUA x AUA

TABLE 154: ADDITION OPERATIONS VS. ADDSRC BIT OF THE AUCONFIG1

ADDSRC[1:0]	Addition operation
00	No addition
01	AUC
10	AUPREV[31:0]
11	32-bit addition of [AUA,AUB] + AUC

The following figure provides a block diagram representation of the arithmetic unit operation for multiplication and addition.

FIGURE 42: ARITHMETIC UNIT MULTIPLICATION AND ADDITION OVERVIEW



The following table provides examples of the AUCONFIG and AUSHIFTCFG register values and the corresponding math operations performed by the arithmetic unit. It also provides the value that would be present in the AURES register if the arithmetic unit input registers were initialized to the following values:

- AUA = 3322h
- AUB = 4411h
- AUC = 11111111h
- AUPREV = 12345678h

TABLE 155:CONFIGURATION OF THE ARITHMETIC UNIT, OPERATION AND OUTPUT RESULT

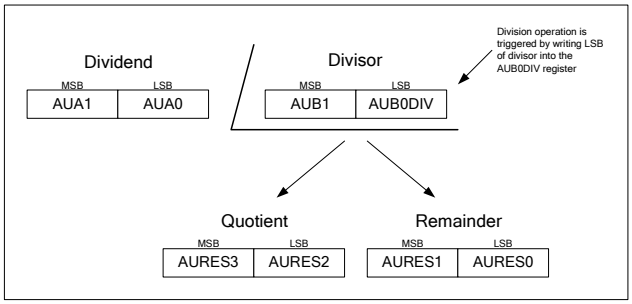
AUCONFIG1	AUSHIFTCFG	Operation	AURES
01h	00h	AUA x AUA	0A369084h
00h	00h	AUA x AUB	0D986D42h
03h	00h	AUA x AUB	0D986D42h
02h	00h	AUA x AUPREV15:0	114563F0h
0Ch,0Dh,0Eh ,0Fh	00h	(AUA,AUB) +AUC] 32 bit addition	44335522h
04h, 07h	00h	(AUA x AUB)+ AUC	1EA97E53h
04h, 07h	01h	((AUA x AUB)+ AUC) x 2 (shift 1 left)	3D52FCA6h
04h, 07h	3Eh	((AUA x AUB)+ AUC) / 4 (shift 2 right)	7AA5F94h

Multiplication and accumulate operations take place within one system clock cycle.

13.9 Division Operation (AUA / AUB1:AUB0DIV)

The VRS51L3xxx arithmetic unit can be configured to perform 16-bit division operations: the division of AUA by AUB1,AUB0DIV. The quotient of this operation is stored in the AURES3, AURES2 registers, with the remainder stored in the AURES1, AURES0 registers The following figure represents a 16-bit division.

FIGURE 43: ARITHMETIC UNIT DIVISION OVERVIEW



Writing the LSB of the divisor into the AUB0DIV register will trigger a division operation. Once the division starts, the value written in the AUB0DIV register will be automatically transferred into the AUB0 register.

This operation is neither affected by the barrel shifter nor the multiplication/addition operation, defined by the AUCONFIG register.

The division operation takes five system clock cycles to be complete.

13.10 Barrel Shifter

The arithmetic unit includes a 32-bit barrel shifter at the output of the 32-bit addition unit. The barrel shifter is used to perform right/left shift operations on the arithmetic unit output. The shift operation takes only one cycle.

The barrel shifter can be used to scale the output result of the arithmetic unit.

The shifting range is adjustable from 0 to 16 in both directions. The “shifted” value can be routed to:

- AURES
- AUPREV
- AUOV32

Moreover, the shift left operation can be configured as an arithmetic or logical shift, in which the sign bit is discarded.

TABLE 156: ARITHMETIC UNIT SHIFT REGISTER CONFIG - AUSHIFTCFG SFR C1H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	SHIFTMODE	AU Barrel SHIFTER Shift Mode 0 = Shift value is unsigned 1 = Shift value is signed
6	ARITHSHIFT	AU Arithmetic Shift Enable 0 = Left shift is considered as logical shift (sign bit is lost) 1 = Left shift is arithmetic shift where sign bit is kept
5:0	SHIFT[5:0]	The value of SHIFT[5:0] equals the amplitude of the shift performed on the arithmetic unit result register AURES Positive value represent shift to the left Negative value represent shift to the right

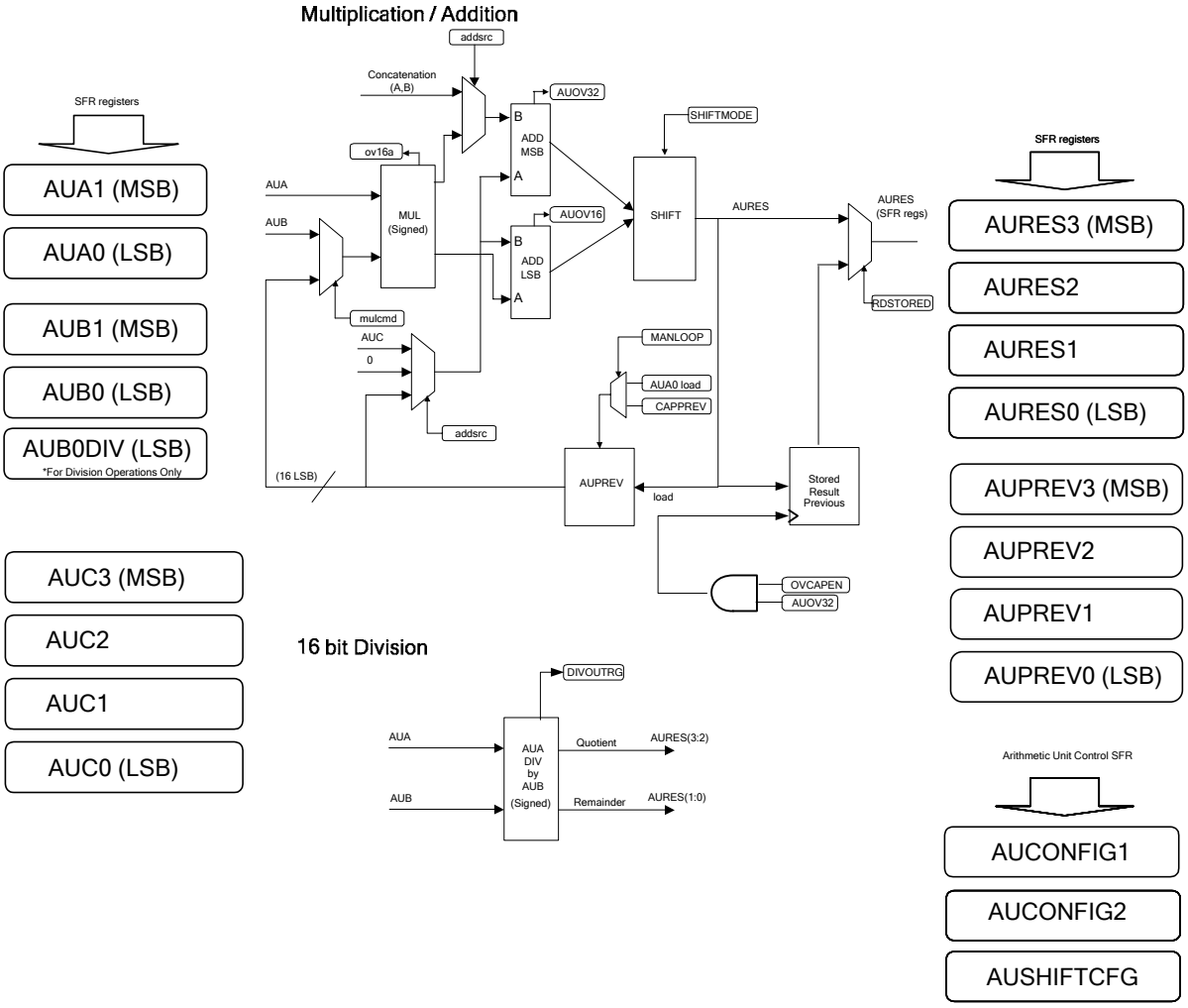
The barrel shifter section operates independently of the multiply and accumulate sections on the arithmetic unit. As such, if the AUSHIFTCFG register bits 5:0 are set to a value other than 0, the value of AUPREV, if derived from the AURES register either automatically or manually, will be affected by the barrel shifter.

When the arithmetic unit is configured to perform multiplication and addition operations, the barrel shifter is active and the shift operation performed depends on the current value of the AUSHIFTCFG register. When the arithmetic unit is configured to perform 16-bit division, the barrel shifter is deactivated.

13.11 Arithmetic Unit Block Diagram

The following block diagram provides a hardware description of the registers and the other components that comprise the enhanced arithmetic unit.

FIGURE 44: ARITHMETIC UNIT FUNCTIONAL DIAGRAM



13.12 Arithmetic Unit Example Programs

Code Example:
Basic Arithmetic Operations

The following example program demonstrates the required arithmetic unit configuration to perform mathematical operations

```
//-------------------------------------------------------------------------------------------/
// VRS51L3074_MULTACCU1_SDCC.c  //
//-------------------------------------------------------------------------------------------/
//
// DESCRIPTION:    VRS51L3074 Arithmetic Unit Demonstration Program
//
//-------------------------------------------------------------------------------------------/
#include <VRS51L3074_SDCC.h>

//-----------------------------------------------------------/
//                MAIN FUNCTION
//-----------------------------------------------------------/
 void main (void) {
    PERIPHEN2 = 0x20;              //Enable Arithmetic Unit

    DEVMEMCFG = 0x01;           //SELECT SFR PAGE 1


    //Configure Arithmetic Unit to perform math operations

    //Place Value in AUA

    AUA1 = 0x33;
    AUA0 = 0x22;

    //Place Value in AUB
    AUB1 = 0x44;
    AUB0 = 0x11;

    //Place Value in AUC

    AUC3 = 0x11;
    AUC2 = 0x11;
    AUC1 = 0x11;
    AUC0 = 0x11;

    //Place Value in AUPREV
    AUPREV3 = 0x12;
    AUPREV2 = 0x34;
    AUPREV1 = 0x56;
    AUPREV0 = 0x78;

    //--Some operation examples--

    // To perform:  [(AUAxAUA)+0]
    AUCONFIG1  = 0x01;          //Set operation (AUA x AUA) + 0
                                //AURES = 0A369084h

    // To perform:  [(AUAxAUB)+0]
    AUCONFIG1  = 0x00;          //Set operation (AUA x AUB) + 0
                                //AURES = 0D986D42h
    // or

    AUCONFIG1  = 0x03;          //Set operation (AUA x AUB) + 0
                                //AURES = 0D986D42h

    // To perform:  [(AUA x AUPREV[15:0]))+0]
    AUCONFIG1  = 0x02;          //Set operation (AUAxAUPREV)+0
                                //AURES = 114563F0h

    // To perform:  [ (AUA,AUB) + AUC ]   32 bit addition
    AUCONFIG1  = 0x0C;          //Set operation (AUA,AUB)+ AUC
                                //AURES = 44335522h

    //or...
    AUCONFIG1  = 0x0D;          //Set operation (AUA,AUB)+ AUC
                                //AURES = 44335522h
    //or...
    AUCONFIG1  = 0x0E;          //Set operation (AUA,AUB)+ AUC
                                //AURES = 44335522h
    //or...
    AUCONFIG1  = 0x0F;          //Set operation (AUA,AUB)+ AUC
                                //AURES = 44335522h

    // To perform:  [ (AUA x AUB)+ AUC ]  No shift
    AUCONFIG1  = 0x04;          //Set operation (AUA x AUB)+ AUC
    AUSHIFTCFG = 0x00;          //No Shift
                                //AURES = 1EA97E53h
       // To perform:  [ (AUA x AUB)+ AUC ] x 2 (Shift one LEFT)
    AUCONFIG1  = 0x04;          //Set operation (AUA x AUB)+ AUC
    AUSHIFTCFG = 0x01;          //Set barrel shifter to perform one SHIFT LEFT (logical)
                                //No need to preset the AUSHIFTCFG register for every
                                //operations
                                //AURES = 3D52FCA6h

    // To perform:  [ (AUA x AUB)+ AUC ] / 2 (Shift one Right)
    AUCONFIG1  = 0x04;          //Set operation (AUA x AUB)+ AUC
    AUSHIFTCFG = 0x3F;           //Set barrel shifter to perform one SHIFT right
                                 //No need to preset the AUSHIFTCFG register for every
                                //operations
                                //AURES = F54BF29h


    DEVMEMCFG = 0x00;       //SELECT SFR PAGE 0

   while(1);


}// End of main
```

Code Example:
FIR Filter Function

The following example program shows the implementation of a FIR filter computation function for one iteration; a data shifting operation; and the definition of the FIR filter coefficient table. The FIR computation algorithm is simple to implement, but requires a lot of processing power. For each new data point, multiplication with the associated coefficients and addition operations must be performed N times (N=number of filter taps).

Since it is hardware-based, the arithmetic unit is very efficient in performing operations such as FIR filter computation. In the example below, the COMPUTEFIR loop is the “heart” of the FIR computation. Note that because of the arithmetic unit’s features, very few instructions are needed to perform mathematical operations and the calculation results are ready at the next instruction. This provides a dramatic performance improvement when compared to having to perform all math operations manually, using general processor instructions.

```
//-----------------------------------------------------------//
// VRS51L3074_AU_FIR_asm_c_-SDCC.c    //
//-------------------------------------------------------------------------------------------------------------//
//
// DESCRIPTION:  FIR filter demonstration program - mixed ASM and C coding to optimize
//                    the FIR loop speed.
//
//        This program demonstrates the configuration and use of the SPI interface
//        for interface to serial 12-bit A/D and D/A converters.
//        The program reads the A/D and outputs the read value on a D/A converter
//
//        At 40MHzm the 16-tap FIR loop + data shifting of the VRS51L3074 provide the
//        following performances:
//
//        FIR computation using AU module (asm) = 10.4 uSeconds
//        Data shifting (asm) =                   17.2 useconds
//        FIR Computation + Datashift =           27.6 uSeconds  (1/T = 36.2 KHz)
//
//  Rev   1.0
//  Date: August 2005
//-------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

//--FIR Filter Coefficient Tables
//;FSAMPLE 480HZ, N=16, LOW PASS 0.1HZ -78DB @ 60HZ

const int flashfircoef[] =
{0x023D,0x049D,0x086A,0x0D2D,0x1263,0x1752,0x1B30,0x1D51,
0x1D51,0x1B30,0x1752,0x1263,0x0D2D,0x086A,0x049D,0x023D};
//-- Global variables definition
int at 0x30 fircoef[16];
int at 0x50 datastack[16];
unsiged int at 0x75 dacdata;
```

```
//---- Functions Declaration ----//
//-- FIR Filter computation function
void FIRCompute(void);
void CopyFIRCoef(void);

//--Gen_ADC
void ReadGen_ADC(void);                 //

//- Gen_DAC
void WriteGen_DAC(unsigned int );

//---Generic functions prototype
void V2KDelay1ms(unsigned int);   //Standard delay function

// Global variables definitions
idata unsigned char cptr = 0x00;

unsigned int adcdata = 0x00;

//-------------------------------------------------------------//
//---------        MAIN FUNCTION   ----------------//
//-------------------------------------------------------------//
void main (void) {
   PERIPHEN2 |= 0x02;               //Enable PWM SFR
   P2PINCFG = 0xF0;                 //P2[3:0] is output
   PWMCLKCFG = 0x10;                //PWM Timer 7 Prescaler = Sys Clock / 2
   //--Configure PWM7 as timer  (will be monitored by interrupt)

   //  PWM Timer 7 counts from 0000 to A2C2h
   PWMCFG =  0x17;                  //Point to MSB MID
   PWMDATA = 0xA2

   PWMCFG =  0x07;                  //Point to LSB MID
   PWMDATA = 0xC2;

   //--Configure and enable PWM as timer Interrupt  to monitor PWM5 only
   INTSRC2 &= 0xDF;                 //PWM7:4 Timer module Interrupt
   INTPINSENS1 = 0xDF;              // sensitive on high level(0)
   INTPININV1 = 0xDF;               //Set INT0 Pin sensitivity on normal level(0)
   INTEN2 |= 0x20;                  //Enable PWM7:4 Timer module interrupt

   //-- Copy FIR filter coefficients to IRAM
   CopyFIRCoef();

   //--Activate the PWM modules and configure the PWM modules as timers
   PWMEN |= 0x80;                   //Enable PWM 7
   PWMTMREN |= 0x80;                //Enable PWM 7 as Timer
   GENINTEN = 0x01;                 //Enable global interrupt
   while(1);

}// End of main

//-------------------------------------------------------------------------//
//---------------------- Interrupt Function-----    ---------------------//
//-------------------------------------------------------------------------//

//-------------------------------------------------------------------------
// NAME:                INT13Interrupt PWMTMR7:4  as Timer
//-------------------------------------------------------------------------
void INT13Interrupt(void) interrupt 13
   {
   char flagread;

   INTEN2 = 0x00;                   //Disable PWM7:4 Timer module interrupt

   flagread = PWMTMRF;              //read PWM Timer OV Flags
   flagread &= 0x80;                //check if PWM Timer 7 OV Flag is Active
   if(flagread != 0x00)
      {
      P2 = P2^0x01;                 //Toggle P2.0 (test)
      ReadGen_ADC();                //Read the A/D Converter
      FIRCompute();                 //Perform the FIR filter computation and write into DAC
      }
   PWMTMRF &= 0x7F;                 //Clear the PWM Timer 7 OV Flag
   INTEN2 |= 0x20;                  //Enable PWM7:4 Timer module interrupt
   }//end of PWM as timer interrupt

//-------------------------------------------------------------------------//
//---------------------- Individual Functions ----------------------------//
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------
// NAME:                FIRCompute
//-------------------------------------------------------------------------
void FIRCompute()
   {
   char *coef = &fircoef;
   char *ydata = &datastack;
   char fircptr = 0x00;

   PERIPHEN2 |= 0x20;               //Enable the Arithmetic Unit
   P2 = 0xFF;                       //Set P2 = 0xFF to monitor duration for FIR Loop
   *ydata = adcdata & 0x0FF;        //Store the LSB of adc read data
   ydata += 1;
   *ydata = (adcdata >> 8)&0x00FF;  //Store the MSB of adc read data
   DEVMEMCFG = 0x01;                //Switch to SFR Page 1
   AUCONFIG1 = 0x08;                //CAPREV  = 0  : Previous Res capture is automatic
                                    //CAPMODE = 1  : Capture of previous Result
                                    //occurs when AUA0 is written into
                                    //OVCAPEN = 0  : Capture on OV32 disabled
                                    //READCAP = 0  : AURES contains current result
                                    //ADDSRC  = 10 : Add SCR = AUC
                                    //MULCMD  = 00 : Mul cmd = AUA x AUB

   AUCONFIG2 = 0xA0;                //Clear the Arithmetic Unit registers

   _asm
   MOV R0,#0x30;                    //Copy Start address of FIR Coefficient Table into R0
   MOV R1,#0x50;                    //Copy Start address of FIR Data Table into R1
   _endasm;

   // Yn Computation mostly in assembler -- Faster...
   for(fircptr = 0; fircptr < 16; fircptr++)
         {
         _asm
         MOV  0xA2,@R0;             //copy LSB of pointed coefficient to AUA0
         INC  R0;
         MOV  0xA3,@R0;             //copy MSB of pointed coefficient to AUA1
         INC  R0;
         MOV  0xB2,@R1;             //copy LSB of pointed coefficient to AUB0
         INC  R1;
         MOV  0xB3,@R1;             //copy MSB of pointed coefficient to AUB1
         INC  R1;
         _endasm;
         }//end of For cptr

   //-- Performing the data stack shifting allows to save 8.8uS @ 40MHz
   _asm
   MOV R0,#0x6F;
   MOV R1,#0x71;
   _endasm;

   for(fircptr = 16; fircptr > 0; fircptr--)
   {
      _asm
      mov  A,@R0;
      mov  @R1,A;
      dec  R0;
      dec  R1;
      mov  A,@R0;
      mov  @R1,A;
      dec  R0;
      dec  R1;
      _endasm;
      }//end of shift for loop

      //-Scale down the AURES output by 16 using the barrel shifter
      // the coefficient had been scaled up by a factor of 65536
      AUSHIFTCFG = 0x30;
      _asm
      NOP;
      _endasm;
      P2 = 0x00;                    //Set P2 = 0x00 to signal the end of the FIR Loop

      dacdata = (AURES1 << 8) + AURES0;

   //Reset the Barrel shifter
   AUSHIFTCFG = 0x00;
   // Note:
   //      In this case, 6 System clock cycles could be saved
   //      by reading AURES3 and AURES2 directly
   DEVMEMCFG = 0x00;                //Switch to SFR Page 0
   WriteGen_DAC(dacdata);           //Write data to SPI DAC
   }//End of FIRCompute
```

```
//-------------------------------------------------------------------------
// NAME:                CopyFIRCoef
//-------------------------------------------------------------------------
// DESCRIPTION:  Copy the FIR Filter Coefficient into
//               SRAM variable which is faster access
//               than Flash
//-------------------------------------------------------------------------
void CopyFIRCoef(void)
   {
   char cptr = 0x00;
         for(cptr = 0x00; cptr < 16; cptr++)
      fircoef[cptr]= flashfircoef[cptr];
   }//End of CopyFIRCoef

//-------------------------------------------------------------------------
// NAME:              ReadGen_ADC
//-------------------------------------------------------------------------
// DESCRIPTION:  Read the Gen_ADC A/D
//      ADC is connected to SPI interface using CS0
//      Max clk speed is 3.2MHz, Fosc = 40MHz assumed
//-------------------------------------------------------------------------
void ReadGen_ADC()
   {
   int cptr = 0x00;
   char readflag = 0x00;

   //SPI Configuration Section
   /(Can be moved to Main function if only one device is connected to the SPI interface)

   PERIPHEN1 |= 0xC0;                  //Make sure the SPI interface is activated

   //--Wait activity stops on the SPI interface (Monitor SPINOCS)
   while(!(SPISTATUS &= 0x08));

   SPICTRL = 0x65;                     //SPICLK = /16 (2.5MHz)
                                       //CS0 Active
                                       //SPI Mode 1 Phase = 1, POL = 0
                                       //SPI Master Mode

   SPICONFIG = 0x40;                   //SPI Chip select is automatic
                                       //Clear SPIUNDEFC flag
                                       //SPILOAD = 0 -> Manual CS3 behaviour
                                       //No SPI interrupt used
   SPISTATUS = 0x00;                   //SPI transactions are in MSB first format
   SPISIZE = 0x0E;                     //SPI transaction size are 15-bit

   //-Dummy Read the SPI RX buffer to clear the RXAV flag
   readflag = SPIRXTX0;

   //-Perform the SPI read
    SPIRXTX0 = 0x00;                   //Writing to the SPIRXTX0 will trigger the SPI
                                       //Transaction
   //Wait for the SPI RX AV Flag being set
   while(!(SPISTATUS &= 0x02));
   /*
   // -- It is possible to monitor the SPINOCS flag instead of the SPIRXAV flag
   //The code piece below shows how to do it.  However in that case,
   //No that the reading of the SPISTATUS register must be done at
   //least 4 system clock cycles after the write operation to the SPIRXTX0 register

   //-Wait for SPINOCS Flag have time to be updated
   _asm
   NOP;
   _endasm;

   //--Wait activity stops on the SPI interface
   while(!(SPISTATUS &= 0x08));
   */

   //Read SPI data
   adcdata= (SPIRXTX1 << 8);
   adcdata+= SPIRXTX0;
   adcdata&= 0x0FFF;                   //isolate the 12 lsb of the read value
   }//end of ReadGen_ADC


//-------------------------------------------------------------------------------------//
// NAME:              WriteGen_DAC
//-------------------------------------------------------------------------------------//
// DESCRIPTION:  Write 12bit Data into the Gen_DAC device
//               ADC is connected to SPI interface using CS1
//               Max clk speed is 12.5MHz, Fosc = 40MHz assumed
//               We will set the SPI prescaler to sysclk / 8
//
void WriteGen_DAC(unsigned int dacdata)
   {
   char subdata = 0x00;
   char readflag = 0x00;

   PERIPHEN1 |= 0xC0;          //Make sure the SPI interface is activated

   //--Wait activity stops on the SPI interface (Monitor SPINOCS)
   while(!(SPISTATUS &= 0x08));

   //SPI Configuration Section
   //Can be moved to main function if only one device is connected to the SPI interface

   SPICTRL = 0x4D;             //SPICLK = /8 (MHz)
                               //CS1 Active
                               //SPI Mode 1 Phase = 1, POL = 0
                               //SPI Master Mode

   SPICONFIG = 0x40;           //SPI Chip select is automatic
                               //Clear SPIUNDEFC Flag
                               //SPILOAD = 0 -> Manual CS3 behaviour
                               //No SPI interrupt used

   SPISTATUS = 0x00;           //SPI transactions are in MSB first format
   SPISIZE = 0x0B;             //SPI transaction size are 12 bit

   //-Format the 12 bit data so data bit 11 is positioned on bit 7 of SPIRXTX0
   // and data bit 0 is positioned on bit 4 of SPIRXTX1 and perform the SPI write operation

   dacdata &= 0x0FFF;          //Make sure dacdata is <= 0FFFh  (12 bit)
   SPIRXTX3 = 0x00;
   SPIRXTX2 = 0x00;
   SPIRXTX1  = (dacdata << 4)& 0xF0;

   //-Dummy read the SPI RX buffer to clear the RXAV Flag  (facultative if SPINOCS is
monitored)
   readflag = SPIRXTX0;

   SPIRXTX0 = (dacdata >> 4);  //Writing to SPIRXTX0 will trigger the transmission

   //--Wait the SPI transaction completes
   //  This section can be omitted if a check of activity on the SPI interface
   //  is made before each access to it in master mode

   //Wait for the SPI RX AV flag being set
   while(!(SPISTATUS &= 0x02));
   // -- It is possible to monitor the SPINOCS flag instead of the SPIRXAV flag
   //The code piece below shows how to do it.  However in that case,
   //No that the reading of the SPISTATUS register must be done at
   //least 4 system clock cycles after the write operation to the SPIRXTX0 register
   /*
   //-Wait for SPINOCS flag have time to be updated
   _asm
      NOP;
   _endasm;
   //--Wait activity stops on the SPI interface  (monitor SPINOCS Flag)
   while(!(SPISTATUS &= 0x08));
   */
   }//end of WriteGen_DAC

//-------------------------------------------------------------------------------------//
// NAME:                V2KDelay1ms
//-------------------------------------------------------------------------------------//
// DESCRIPTION:  VRS3074 specific 1 millisecond delay function
//               Using Timer 0 and calibrated for 40MHz oscillator
//-------------------------------------------------------------------------------------//
void V2KDelay1ms(unsigned int dlais){
idata unsigned char x=0;
idata unsigned int dlaisloop;

   PERIPHEN1 |= 0x01;                  //LOAD PERIPHEN1 REG

   dlaisloop = dlais;
   while ( dlaisloop > 0)
      {
      TH0 = 0x63;              //TIMER0 RELOAD VALUE FOR 1MS AT 40MHZ
      TL0 = 0xC0;
      T0T1CLKCFG = 0x00;           //NO PRESCALER FOR TIMER 0 CLOCK
      T0CON = 0x04;              //START TIMER 0,  COUNT UP
   do{
   x=T0CON;
   x= x & 0x80;
   }while(x==0);

   T0CON = 0x00;                       //Stop Timer 0
   dlaisloop = dlaisloop-1;
   }//end of while dlais...
      PERIPHEN1 &= 0xFE;               //Disable Timer 0
   }//End of function V2KDelay1ms
```

14 Watchdog Timer

The VRS51L3xxx devices include a watchdog timer . The watchdog timer is composed of a 14-bit prescaler, which derives its source from the active system clock. An overflow of the watchdog timer resets the processor. The WDTCFG SFR register controls the watchdog timer operations.

TABLE 157: THE WATCHDOG TIMER REGISTER - WDTCFG 91H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:4	WDTPERIOD	Watchdog Timer Period Configuration *see table below
3	WTIMEROVF	WDT as Timer Overflow Flag 0 = WDT as timer as not expired 1 = WDT as timer has overflow
2	ASTIMER	Watchdog as Timer 0 = WDT mode 1 = WDT operate as a regular timer (no reset) Writing to this bit will clear the timer
1	WDTOVF	Read: 0 = Watchdog is counting 1 = Watchdog timer period has expired
		Write: 0 = No action 1 = Clear the watchdog timer flag
0	WDTRESET	Read: No Action
		Watchdog Timer Reset To reset the watchdog timer, two consecutive writes to the WDTRESET bit must be made: First clear the WDTRESET bit and second, set it to 1

14.1 WDT Timeout Period

The watchdog timer timeout period is controlled by adjusting bit 7:4 of the WDTCFG register. The following table provides the approximate timeout vs. the selected WDTPERIOD. The WDT timeout period is not affected by the clock divider.

TABLE 158: WATCHDOG TIMER REGISTER TIMEOUT PERIOD

WDTPERIOD Value (4 bit)	Actual WDT Period**	Approx Timeout** (40MHz)
0000	0x3FFF*	819-1000 us
0001	0x3FFE	1.23 – 1.36 ms
0010	0x3FFD	2.05 – 2.2 ms
0011	0x3FFB	4.92 ms
0100	0x3FF4	9.83 ms
0101	0x3FE8	20.07 ms
0110	0x3FCF	49.97 ms
0111	0x3F86	74.96 ms
1000	0x3F49	99.94 ms
1001	0x3F0C	249.86 ms
1010	0x3E9E	500.12 ms
1011	0x3B3B	749.98 ms
1100	0x38D9	999.83 ms
1101	0x3677	2.99 s
1110	0x2364	6.71s
1111	0x0000	13.4 s

*Not available in timer mode

The watchdog timer timeout period is calculated as follows:

$$\text{WDT Period*} = 2*\frac{16384*(0x4000 - \text{WDTPERIOD}[3:0])}{\text{Fosc}}$$

**For a given configuration, the timeout period of the watchdog timer may vary by about 200us. This delay is caused by internal timing of the watchdog timer module.*

14.2 Resetting the Watchdog Timer

To reset the watchdog timer, two consecutive write operations to the WDTCFG register must be performed. During the first write operation, the WDTRESET bit must be cleared. During the second write operation, the WDTRESET should be set to 1.

This sequence is also required to set a new value for WDTPERIOD. For example, if the watchdog period is set to 100ms, the following sequence of operations will reset the watchdog timer:

```
MOV     WDTCFG,#92h
MOV     WDTCFG,#93h
```

Errata: Resetting the Watchdog Timer when running from an external crystal

There is an issue in the clock setting of the WDT that prevent the program to reset the WDT properly when the Clock Divider setting of the Versa Ware JTAG software Device Option is Set to OFF and the VRS51L3xxx operates from an external crystal or oscillator.

Whenever the VRS51L3xxx is running from an external crystal or oscillator and the WDT module is to be used, you must set the Clock Divider setting of the Device Option in Versa Ware to either Fosc/2, Fosc/4 or Fosc/8.

In order to run the program at "full" speed, add the instruction: DEVCLKCFG1 &= 0xF0; somewhere at the beginning of the code to force the system clock speed back to Fosc/1

This work around is only required for application using the WDT and running from an external crystal or oscillator and it does not apply if the internal 40MHz oscillator is used

14.3 WDT Example Programs

Code Example:

WDT configuration and reset example when using the internal 40MHz oscillator

The following demo program shows how to configure and use the WDT module of the VRS51L3xxx when running from the internal oscillator

```
//-------------------------------------------------------------------------------------------------------------//
// V3K_WDT_Demo_Int_40MHzOsc_SDCC.c  //
//-------------------------------------------------------------------------------------------------------------//
// DESCRIPTION:
//         VRS51L2070/3074 Watchdog timer Demonstration Program
//         when using the Internal Oscillator.
//               Program operation:
//               *This Program Set P1 as Output
//               *P1 is set to 0xFF for 100ms
//               *Initialize the watchdog Timer with a Timeout period of 20ms
//               *Clear P1
//               *Start a Delay function
//               *If the Delay parameter of the Delay function is larger than the
//               Timeout period of the watchdog Timer, the WDT will reset the VRS2000
//               which will bring back P1 to high level
//   Note/Errata:
//         Please refer to the "V2K_WDT_Demo_w_CY_osc_SDCC.c" demo program if you
want
//       to use the WDT in a application running from the external crystal
//       or an oscillator module.
//-------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

// --- function prototypes
void delay40(unsigned int);
```

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

    PERIPHEN1 = 0x01;                   //Enable Timer 0
    P1PINCFG = 0x00;                    //Config port 1 as output

    DEVCLKCFG1 &= 0xF0;                 //Force Maximum clock speed

    //-- Enable the Watchdog Timer
    PERIPHEN2 |= 0x04;
    P1 = 0xFF;                          //Set P1 to output 0xFF
    delay(10);                          //Keep P1 high for 10ms

    //-- Configure the watchdog Timer
    WDTCFG = 0x62;              //Configure and Reset the Watchdog Timer
    WDTCFG = 0x63;              //Bit 7:4  = WDTPERIOD : Define the timeout
                                //period (~40ms @ 40MHz)
                                //Bit 3= WTIMEROVF: WDT as Timer
                                //Overflow Flag
                                //Bit 2 = ASTIMER: WDT mode
                                //(0=WDT, 1=Timer)
                                //Bit 1    = WDTOVF    : WDT Overflow (Timeout) Flag
                                //Bit 0    = WDTRESET  : WDT Reset. To reset WDT
                                //this bit must be cleared, then Set

    P1 = 0x00;                  //Clear P1
   do{
     delay(50);                //If delay > 40ms then the WDT will reset the processor
                                //and P1 will return to High

     WDTCFG = 0x62;             //Reset the Watchdog Timer
     WDTCFG = 0x63;
   }while(1);                   //Infinite Loop

}// End of main

//-----------------------------------------------------------------//
//                INDIVIDUALS FUNCTIONS                //
//-----------------------------------------------------------------//
//;-----------------------------------------------------------------
//;- DELAY40 : 1MS DELAY USING TIMER0
//;
//; CALIBRATED FOR 40MHZ
//;-----------------------------------------------------------------
void delay40(unsigned int dlais){

idata unsigned char x=0;
idata unsigned int dlaisloop;

    x = PERIPHEN1;              //Load PERIPHEN1 register
    x |= 0x01;                  //Enable Timer 0
    PERIPHEN1 = x;

    dlaisloop = dlais;
    while ( dlaisloop > 0)
      {
       TH0 = 0x63;              //Timer 0 Reload value for 1MS at 40MHZ
       TL0 = 0xC0;
       T0T1CLKCFG = 0x00;       //No prescaler for Timer 0 clock
       T0CON = 0x04;            //Start Timer0 Count-up

        do{
            x=T0CON;
            x= x & 0x80;
            }while(x==0);

        T0CON = 0x00;           //Stop Timer 0
        dlaisloop = dlaisloop-1;

        }//end of while dlais...

       x = PERIPHEN1;           //Load PERIPHEN1 register
       x = x & 0xFE;            //Disable Timer 0
       PERIPHEN1 = x;
    }//End of function delay40
```

Code Example:

WDT configuration and reset example when using an external oscillator

The following demo program show the configuration and use of the WDT module when the VRS51L3xxx is running from an external crystal or an oscillator.

```
//-------------------------------------------------------------------------------------------------------------//
// V3K_WDT_Demo_w_ext_osc_SDCC.c  //
//-------------------------------------------------------------------------------------------------------------//
//
// DESCRIPTION:
//      VRS51L2070/3074 Watchdog Timer Demonstration Program when using an
//      the Crystal oscillator or an external oscillator module.
//
//              Errata:
//                 In order for the WDT to work properly when the VRS51L2070/3074
//                 operates from an external crystal or oscillator module,
//                 the Clock Divider setting in the device Option of the Versa Ware JTAG
//                 programming interface MUST be configured to either Fosc/2, Fosc/4
//                 or Fosc/8
//
//                 In order to run the application at "full speed", the DEVCLKCFG1
//                 SFR register MUST be manually reconfigured to the  Fosc/1
//                 from within the code. )

//  Program operation:
//
//      *This Program Set P1 as Output
//      *Reconfigure the DEVCLKCFG1 register to Fosc/1
//      (if system must operate at "full" speed)
//      *Activate the WDT module
//      *P1 is set to 0xFF for 10ms (still running from 40MHz)
//       *Activate external Crystal oscillator and switch processor operation to it
//      *Initialize the watchdog Timer and configure the Timeout period
//      *Clear P1
//      *Start a Delay function
//      *If the Delay parameter of the Delay function is larger than the
//      Timeout period of the watchdog Timer, the WDT will reset the processor
//      which will bring back P1 to high level
//-------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

// --- function prototypes

void delay40(unsigned int);
void delay22(unsigned int);
//-------------------------------------------------------------------//
//                      MAIN FUNCTION
//-------------------------------------------------------------------//
void main (void) {

    PERIPHEN1 = 0x01;               //Enable Timer 0
    P1PINCFG = 0x00;                //Config port 1 as output

   DEVCLKCFG1 &= 0xF0;              //Force Maximum clock speed (See Errata Note)


    //-- Enable the Watchdog Timer
    PERIPHEN2 |= 0x04;
    P1 = 0xFF;                      //Set P1 to output 0xFF
    delay40(10);                    //Keep P1 high for 10ms

    //--Activate the external crystal oscillator
    DEVCLKCFG2 = 0xC4;              //Activate the external crystal oscillator (22.1184MHz)
    delay40(1);
    DEVCLKCFG1 &= 0xBF;             //Select the external crystal oscillator
    DEVCLKCFG2 &= 0xBF;             //Deactivate the internal oscillator


    //-- Configure the watchdog Timer
    WDTCFG = 0x62;                  //Configure timeout period ~72ms (22.1184MHz)
                                    //and Reset the Watchdog Timer
    WDTCFG = 0x63;                  //Bit 7:4  = WDTPERIOD : The timeout period (20ms)
                                    //Bit 3  = WTIMEROVF: WDT as Timer Overflow Flag
                                    //Bit 2  = ASTIMER   : WDT mode (0=WDT, 1=Timer)
                                    //Bit 1   = WDTOVF    : WDT Overflow (Timeout) Flag
                                    //Bit 0   = WDTRESET  : WDT Reset. To reset WDT
                                    //this bit must be cleared, then Set
   P1 = 0x00;                       //Clear P1
    do{
     delay22(50);                       //If delay > ~72ms then the WDT will reset
                                        //the processor and P1 will return to High

     WDTCFG = 0x62;                 //Reset the Watchdog Timer
     WDTCFG = 0x63;
    }while(1);                      //Infinite Loop

}// End of main

//-------------------------------------------------------------------//
//                  INDIVIDUALS FUNCTIONS
//-------------------------------------------------------------------//
//;-------------------------------------------------------------------
//;- DELAY40 : 1MS DELAY USING TIMER0
//;
//; CALIBRATED FOR 40MHZ
//;-------------------------------------------------------------------
void delay40(unsigned int dlais){

idata unsigned char x=0;
idata unsigned int dlaisloop;

    x = PERIPHEN1;                  //Load PERIPHEN1 register
    x |= 0x01;                      //Enable Timer 0
    PERIPHEN1 = x;

    dlaisloop = dlais;
    while ( dlaisloop > 0)
      {
       TH0 = 0x63;                  //Timer 0 Reload value for 1MS at 40MHZ
       TL0 = 0xC0;
       T0T1CLKCFG = 0x00;           //No prescaler for Timer 0 clock
       T0CON = 0x04;                //Start Timer0 Count-up

        do{
            x=T0CON;
            x= x & 0x80;
            }while(x==0);

        T0CON = 0x00;               //Stop Timer 0
        dlaisloop = dlaisloop-1;

        }//end of while dlais...

       x = PERIPHEN1;               //Load PERIPHEN1 register
       x = x & 0xFE;                //Disable Timer 0
       PERIPHEN1 = x;
    }//End of function delay40


//;-------------------------------------------------------------------
//;- DELAY22 : 1MS DELAY USING TIMER0
//;
//; CALIBRATED FOR 22.1184MHZ
//;-------------------------------------------------------------------
void delay22(unsigned int dlais){

idata unsigned char x=0;
idata unsigned int dlaisloop;

    x = PERIPHEN1;                  //Load PERIPHEN1 register
    x |= 0x01;                      //Enable Timer 0
    PERIPHEN1 = x;

    dlaisloop = dlais;
    while ( dlaisloop > 0)
      {
       TH0 = 0xA9;                  //TIMER0 RELOAD VALUE FOR 1MS AT 22.1184MHZ
       TL0 = 0x9A;
       T0T1CLKCFG = 0x00;           //No prescaler for Timer 0 clock
       T0CON = 0x04;                //Start Timer0 Count-up

        do{
            x=T0CON;
            x= x & 0x80;
            }while(x==0);

        T0CON = 0x00;               //Stop Timer 0
        dlaisloop = dlaisloop-1;

        }//end of while dlais...

       x = PERIPHEN1;               //Load PERIPHEN1 register
       x = x & 0xFE;                //Disable Timer 0
       PERIPHEN1 = x;
    }//End of function delay22
```

14.4 Using the Watchdog as a Timer

The watchdog timer module can also be used as a timer. In this case, the timeout period is defined by the watchdog timer period value. Due to the presence of the 14-bit prescaler, long timeout periods can be achieved.

Configuring the watchdog timer operation as a general purpose timer is achieved by:

- Setting the ASTIMER bit of the WDTCFG register to 1
- Selecting the timer maximum time value of WDTPeriod
- Performing a watchdog timer reset sequence to clear the timer and apply the timer configuration

The WTIMERFLAG bit of the WDTCFG register is used to monitor the timer overflow. When configured in timer mode, the watchdog timer does not reset the processor and cannot trigger an interrupt.

14.5 Watchdog as Timer Example Program

Code Example:

Initialization and Reset of the Watchdog Timer

```
//-------------------------------------------------------------------------------------------------------------------//
// VRS51L3074-WDT_Demo_SDCC.c  //
//-------------------------------------------------------------------------------------------------------------------//
// DESCRIPTION:     VRS51L3074 Watchdog Timer Demonstration Program
//              *This Program Set P1 as output
//              *P1 is set to 0xFF for 100ms
//              *Initialize the watchdog timer with a timeout period of 20ms
//              *Clear P1
//              *Start a delay function
//              *If the Delay parameter of the delay function is larger than the
//              Timeout period of the watchdog timer, the WDT will reset the VRS51L3074
//              which will bring back P1 to high level
//-------------------------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>

// --- function prototypes
void delay(unsigned int);

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

void main (void) {

    PERIPHEN1 = 0x01;               //Enable Timer 0
    PERIPHEN2 = 0x08;               //Enable IOPORT

    P1PINCFG = 0x00;                //Config port 1 as output

    //-- Enable the Watchdog Timer
    PERIPHEN2 |= 0x04;
    P1 = 0xFF;                        //Set P1 to output 0xFF
    delay(100);                      //Keep P1 high for 100ms

    //-- Configure the watchdog timer

    WDTCFG = 0x62;          //Configure and Reset the Watchdog Timer
    WDTCFG = 0x63;          //Bit 7:4  = WDTPERIOD : Define the timeout period (20ms)
                              //Bit 3    = WTIMEROVF : WDT as timer overflow flag
                              //Bit 2    = ASTIMER   : WDT mode (0=WDT, 1=Timer)
                              //Bit 1    = WDTOVF    : WDT overflow (Timeout) Flag
                              //Bit 0    = WDTRESET  : WDT reset. To reset WDT
                              //this bit must be cleared, then set

    P1 = 0x00;                  //Clear P1

    do{
      delay(10);                //If delay > 20ms then the WDT will reset the VRS51L3074
                                  //and P1 will return to high

      WDTCFG = 0x62;        //Reset the watchdog timer
      WDTCFG = 0x63;
    }while(1);                  //Loop Forever

}// End of main

//;-------------------------------------------------------------------------//
//;- DELAY1MSTO : 1MS DELAY USING TIMER0
//;
//; CALIBRATED FOR 40MHZ
//;-------------------------------------------------------------------------//
void delay(unsigned int dlais){

idata unsigned char x=0;
idata unsigned int dlaisloop;

    x = PERIPHEN1;                  //LOAD PERIPHEN1 REG
    x |= 0x01;                      //ENABLE TIMER 0
    PERIPHEN1 = x;

    dlaisloop = dlais;
    while ( dlaisloop > 0)
      {
       TH0 = 0x63;                  //TIMER0 RELOAD VALUE FOR 1MS AT 40MHZ
       TL0 = 0xC0;

       T0T1CLKCFG = 0x00;           //NO PRESCALER FOR TIMER 0 CLOCK
       T0CON = 0x04;                //START TIMER 0,  COUNT UP

        do{
            x=T0CON;
            x= x & 0x80;
            }while(x==0);

        T0CON = 0x00;                //Stop Timer 0

        dlaisloop = dlaisloop-1;

        }//end of while dlais...

       x = PERIPHEN1;               //LOAD PERIPHEN1 REG
       x = x & 0xFE;                //DISABLEBLE TIMER 0
       PERIPHEN1 = x;
    }//End of function delais
```

15 Interrupts

The VRS51L3xxx devices feature a comprehensive set of 49 interrupt sources and have 16 interrupt vectors to handle them. The interrupts are categorized in two distinct groups:

- Module interrupt
- Pin change interrupts

The module interrupts include interrupts that are generated by the peripherals such as the UARTs, SPI, I²C , PWC and port change monitoring modules.

As their name implies, the pin change interrupts are interrupts that are generated by predefined conditions at the physical pin level: The pin change interrupts can be caused by a level or an edge (rising or falling) on a given pin. Standard 8051 INT0 and INT1 interrupts are considered pin change interrupts. The VRS51L3xxx includes INT0 and INT1, as well as 14 other pin interrupts distributed on ports 0 and 3.

The interrupt sources share 16 interrupt vectors from 00h to 7Bh. Each interrupt vector can be configured to respond to either a pin change interrupt or a module interrupt. The two following diagrams provide an overview of the VRS51L3xxx modules/pin interrupt structure, the associated SFR registers and the interaction among the interrupt management SFRs.

FIGURE 45: INTERRUPT SOURCES DETAILED VIEW

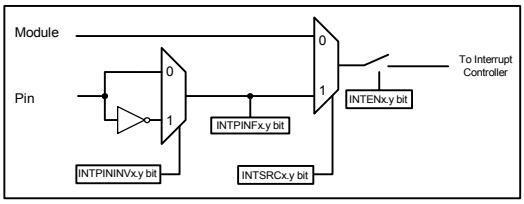
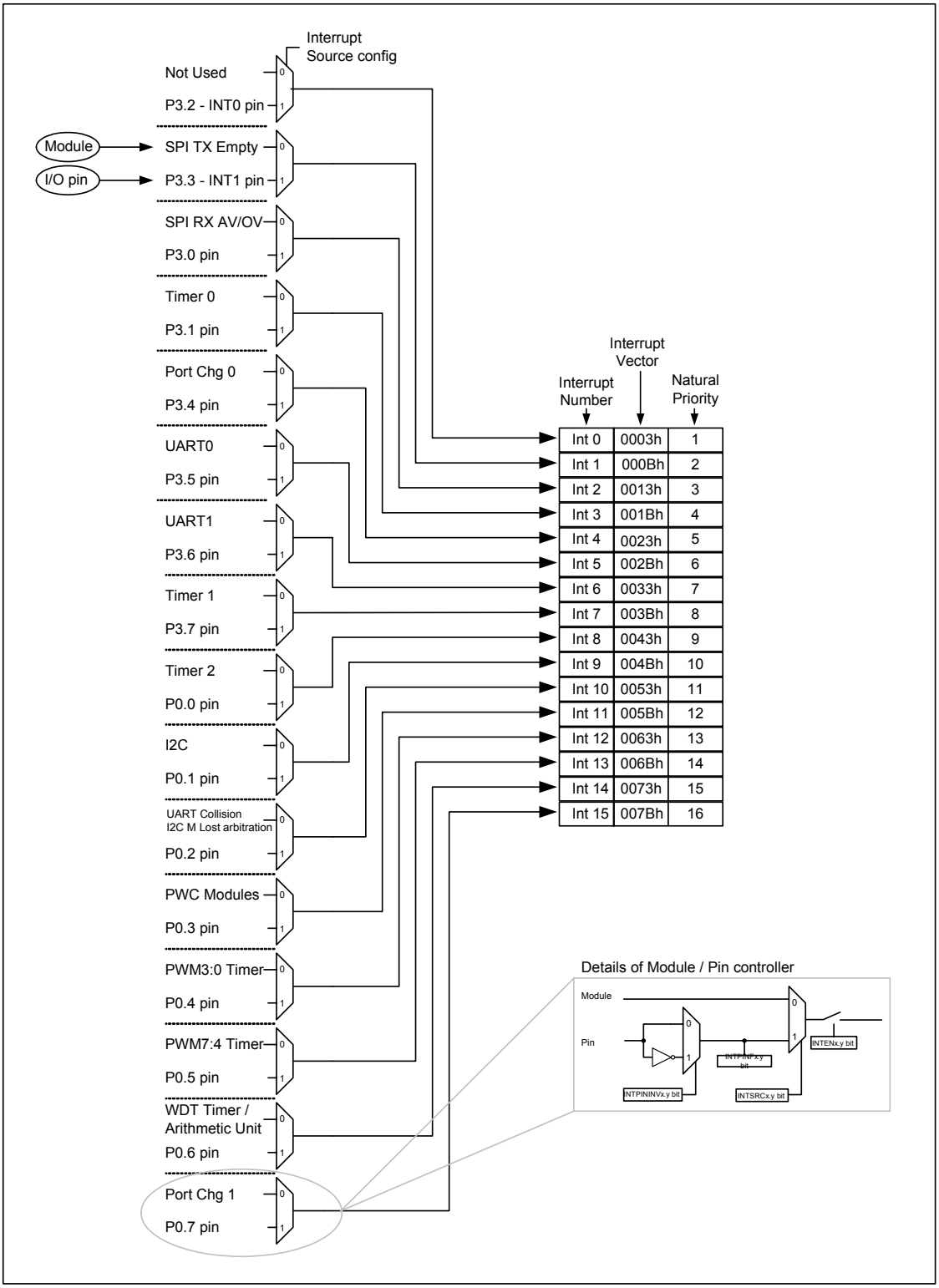


FIGURE 46: INTERRUPT SOURCES OVERVIEW



The interaction between the interrupt management configuration registers is summarized in the following table. The paragraphs below describe each one of these registers in detail.

TABLE 159:VRS51L3XXX INTERRUPT CONFIGURATION SUMMARY

Int #	Priority	Interrupt Vector	Interrupt Enable	Interrupt Priority	Interrupt Source	Connected Modules	Connected Pin	Pin Inversion	Pin Sensitivity	Pin Interrupt Flag
INT 0	1	0003h	INTEN1.0	INTPRI1.0	INTSRC1.0	None	P3.2-INT0	IPINTINV1.0	IPINSENS1.0	IPINFLAG1.0
Int 1	2	000Bh	INTEN1.1	INTPRI1.1	INTSRC1.1	SPI TX Empty	P3.3-INT1	IPINTINV1.1	IPINSENS1.1	IPINFLAG1.1
Int 2	3	0013h	INTEN1.2	INTPRI1.2	INTSRC1.2	SPI RX Available SPI RX Overrun	P3.0	IPINTINV1.2	IPINSENS1.2	IPINFLAG1.2
Int 3	4	001Bh	INTEN1.3	INTPRI1.3	INTSRC1.3	Timer 0	P3.1	IPINTINV1.3	IPINSENS1.3	IPINFLAG1.3
Int 4	5	0023h	INTEN1.4	INTPRI1.4	INTSRC1.4	Port Change 0	P3.4	IPINTINV1.4	IPINSENS1.4	IPINFLAG1.4
Int 5	6	002Bh	INTEN1.5	INTPRI1.5	INTSRC1.5	UART0 Tx Empty UART0 RX Available UART0 RX Overrun UART0 Timer OV	P3.5	IPINTINV1.5	IPINSENS1.5	IPINFLAG1.5
Int 6	7	0033h	INTEN1.6	INTPRI1.6	INTSRC1.6	UART1 Tx Empty UART1 RX Available UART1 RX Overrun UART1 Timer OV	P3.6	IPINTINV1.6	IPINSENS1.6	IPINFLAG1.6
Int 7	8	003Bh	INTEN1.7	INTPRI1.7	INTSRC1.7	Timer 1	P3.7	IPINTINV1.7	IPINSENS1.7	IPINFLAG1.7
Int 8	9	0043h	INTEN2.0	INTPRI2.0	INTSRC2.0	Timer 2	P0.0	IPINTINV2.0	IPINSENS2.0	IPINFLAG2.0
Int 9	10	004Bh	INTEN2.1	INTPRI2.1	INTSRC2.1	I²C Tx Empty I²C RX Available I²C RX Overrun	P0.1	IPINTINV2.1	IPINSENS2.1	IPINFLAG2.1
Int 10	11	0053h	INTEN2.2	INTPRI2.2	INTSRC2.2	UART0 Collision UART1 Collision I²C Master Lost Arbitration	P0.2	IPINTINV2.2	IPINSENS2.2	IPINFLAG2.2
Int 11	12	005Bh	INTEN2.3	INTPRI2.3	INTSRC2.3	PWC 0 End Condition PWC 1 End Condition	P0.3	IPINTINV2.3	IPINSENS2.3	IPINFLAG2.3
Int 12	13	0063h	INTEN2.4	INTPRI2.4	INTSRC2.4	PWM3 as Timer OV PWM2 as Timer OV PWM1 as Timer OV PWM0 as Timer OV	P0.4	IPINTINV2.4	IPINSENS2.4	IPINFLAG2.4
Int 13	14	006Bh	INTEN2.5	INTPRI2.5	INTSRC2.5	PWM7as Timer OV PWM6as Timer OV PWM5as Timer OV PWM4as Timer OV	P0.5	IPINTINV2.5	IPINSENS2.5	IPINFLAG2.5
Int 14	15	0073h	INTEN2.6	INTPRI2.6	INTSRC2.6	Watchdog as Timer OV Arithmetic Unit OV	P0.6	IPINTINV2.6	IPINSENS2.6	IPINFLAG2.6
Int 15	16	007Bh	INTEN2.7	INTPRI2.7	INTSRC2.7	Port Change 1	P0.7	IPINTINV2.7	IPINSENS2.7	IPINFLAG2.7

15.1 Interrupt Enable Registers

The interrupt enable and the general interrupt enable registers establish the link between the peripheral module/pin interrupt signals and the processor interrupt system.

The GENINTEN register controls activation of the global interrupt. The GENINTEN register is similar to the standard 8051 EA bit. When the GENINTEN bit is set to 1, all the enabled interrupts emanating from the modules/pins will reach the interrupt controller.

The CLRPININT bit is used to initialize the Pin interrupt subsystem. It is recommended to set the CLRPININT bit before enabling the pin interrupt. This will avoid receiving a pin interrupt right after the pin interrupt is enabled.

TABLE 160:GENINTEN SFR REGISTER - NAME SFR E8H

7	6	5	4	3	2	1	0
-	-	-	-	-	-	W	R/W
							0

Bit	Mnemonic	Description
7:2	Unused	
1	CLRPININT	It is recommended to set this bit to 1 before enabling a pin interrupt to avoid receiving an interrupt right after GENINTEN bit is set
0	GENINTEN	General Interrupt Enable 0 = All enabled interrupts are masked (deactivated) 1 = All enabled interrupt can raise an interrupt

When a given interrupt bit is set to 1, the corresponding interrupt path is activated.

TABLE 161: INT ENABLE 1 REGISTER - INTEN1 (MODULES /PIN/INT VECTOR) SFR 88H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	T1IEN	Timer 1 Interrupt Enable
	P3.7 pin	P3.7 pin if interrupt source is set to pin
	Int 7	Interrupt vector 7 at address 003Bh
6	U1IEN	UART1 Interrupt Enable o UART1 Tx Empty o UART1 Rx Available o UART1 Rx Overrun o UART1 Baud Rate Generator as Timer Overflow
	P3.6 pin	P3.6 pin if interrupt source is set to pin
	Int 6	Interrupt vector 6 at address 0033h
5	U0IEN	UART0 Interrupt Enable o UART0 Tx Empty o UART0 Rx Available o UART0 Rx Overrun o UART0 Baud Rate Generator as Timer Overflow
	P3.5 pin	P3.5 pin if interrupt source set to pin
	Int 5	Interrupt vector 5 at address 0002Bh
4	PCHGIEN0	Port Change Interrupt Module 0 Enable
	P3.4 pin	P3.4 pin if interrupt source is set to pin
	Int 4	Interrupt vector 4 at address 0023h
3	T0IEN	Timer 0 Interrupt Enable
	P3.3 pin	P3.3 pin if interrupt source is set to pin
	Int 3	Interrupt vector 3 at address 001Bh
2	SPIRXOVIEN	SPI Interrupt Enable SPI Rx Available SPI Rx Overrun
	P3.0	P3.0 pin if interrupt source is set to pin
	Int 2	Interrupt vector 2 at address 0013h
1	SPITXEIEN	SPI Tx Empty Interrupt Enable
	P3.3 pin	P3.3 pin if interrupt source is set to pin
	Int 1	Interrupt vector 0 at address 000Bh
0	No Module	Unused
	P3.2 pin	P3.2 pin if interrupt source is set to pin
	Int 0	Interrupt vector 0 at address 0003h

TABLE 162: INT ENABLE 2 REGISTER INTEN2 (MODULES /PIN/INT VECTOR) SFR A8H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PCHGIEN1	Port Change Interrupt Module 1 Enable
	P0.7 pin	P0.7 pin if interrupt source is set to pin
	Int 15	Interrupt vector 15 at address 007Bh
6	AUWDTIEN	Watchdog Timer and Arithmetic Unit Interrupt Enable o Watchdog as Timer Overflow o Arithmetic Unit 32-bit Overflow
	P0.6 pin	P0.6 pin if interrupt source is set to pin
	Int14	Interrupt vector 14 at address 0073h
5	PWMT74IEN	PWM as Timer 7 to 4 Overflow Interrupt Enable o PWM as Timer Module 7 Overflow o PWM as Timer Module 6 Overflow o PWM as Timer Module 5 Overflow o PWM as Timer Module 4 Overflow
	P0.5 pin	P0.5 pin if interrupt source set to pin
	Int 13	Interrupt vector 13 at address 006Bh
4	PWMT30IEN	PWM as Timer 3 to 0 Overflow Interrupt Enable o PWM as Timer Module 3 Overflow o PWM as Timer Module 2 Overflow o PWM as Timer Module 1 Overflow o PWM as Timer Module 0 Overflow
	P0.4 pin	P0.4 pin if interrupt source is set to pin
	Int 12	Interrupt vector 12 at address 0063h
3	PWCIEN	Pulse Width Counter Interrupt Enable o PWC0 END condition occurred o PWC1 END condition occurred
	P0.3 pin	P0.3 pin if interrupt source set to pin
	Int 11	Interrupt vector 11 at address 005Bh
2	I2CUCOLIEN	I²C and UARTs Interrupts Enable o I²C Master Lost Arbitration o UART0 Collision Interrupt o UART1 Collision Interrupt
	P0.2 pin	P0.2 pin if interrupt source is set to pin
	Int 10	Interrupt vector 10 at address 0053h
1	I2CIEN	I²C Interrupts Enable o TX Empty o RX Available o RX Overrun
	P0.1 pin	P0.1 pin if interrupt source set to pin
	Int 9	Interrupt vector 9 at address 004Bh
0	T2IEN	Timer 2 Interrupt Enable (INTSCR
	P0.0 pin	P0.0 pin if interrupt source is set to pin
	Int 8	Interrupt vector 8 at address 0043h

15.2 Interrupt Source

Each one of the 16 interrupt vectors on the VRS51L3xxx can be configured to function as either a peripheral module or a pin change interrupt. The selection of the interrupt source is handled by the INTSRC1 and the INTSRC2 registers.

By default, the interrupt source is set to peripheral module. However, setting the INTSRC bit to 1 will "associate" the corresponding interrupt vector to the corresponding pin interrupt.

When a given interrupt vector is associated with a module, the corresponding bit of the IPINSENSx must be set to 0, so it is level sensitive (reset value).

TABLE 163:INTERRUPT SOURCE 1 REGISTER - INTSRC1 SFR E4H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	INTSRC1.7	Interrupt 7 Source 0 = Timer 1 1 = P3.7
6	INTSRC1.6	Interrupt 6 Source 0 = UART1 1 = P3.6
5	INTSRC1.5	Interrupt 5 Source 0 = UART0 1 = P3.5
4	INTSRC1.4	Interrupt 4 Source 0 = Port Change 0 1 = P3.4
3	INTSRC1.3	Interrupt 3 Source 0 = Timer 0 1 = P3.1
2	INTSRC1.2	Interrupt 2 Source 0 = SPI RXAV, SPI RXOV 1 = P3.0
1	INTSRC1.1	Interrupt 1 Source 0 = SPI Tx EMPTY 1 = INT1 (P3.3)
0	INTSRC1.0	Interrupt 0 Source 0 = - 1 = INT0 (P3.2)

TABLE 164:INTERRUPT SOURCE 2 REGISTER - INTSRC2 SFR E5H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	INTSRC2.7	Interrupt 15 Source 0 = Port Change 1 1 = P0.7
6	INTSRC2.6	Interrupt 14 0 = WDT Timer OV, AU OV 1 = P0.6
5	INTSRC2.5	Interrupt 13 Source 0 = PWM7:4 Timer 1 = P0.5
4	INTSRC2.4	Interrupt 12 Source 0 = PWM3:0 Timer OV 1 = P0.4
3	INTSRC2.3	Interrupt 11 Source 0 = PWC0, PWC1 1 = P0.3
2	INTSRC2.2	Interrupt 10 Source 0 = UARTs Coll, I²C Lost Arbitration 1 = P0.2
1	INTSRC2.1	Interrupt 9 Source 0 = I²C 1 = P0.1
0	INTSRC2.0	Interrupt 8 Source 0 = Timer 1 = P0.0

15.3 Interrupt Priority

The INTPRIx registers enable the user to modify the interrupt priority of either the module or the pin interrupts. When the INTPRIx is set to 0, the natural priority of module/pin interrupts prevails. Setting the INTPRIx register bit to 1 will set the corresponding module/pin priority to high.

If more than two module/pin interrupts are simultaneously set to high priority, the natural priority order will apply: Priority will be give to the module/pin interrupts with high priority, over normal priority.

TABLE 165:INTERRUPT PRIORITY 1 REGISTER - INTPRI1 SFR E2H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	T1P37PRI	Interrupt 7 Priority Level (Timer 1 / P3.7) 0 = Normal Priority 1 = High Priority
6	U1P36PRI	Interrupt 6 Priority Level (UART1 / P3.6) 0 = Normal Priority 1 = High Priority
5	U0P35PRI	Interrupt 5 Priority Level (UART0 / P3.5) 0 = Normal Priority 1 = High Priority
4	PC0P34PRI	Interrupt 4 Priority Level (Port Chg 0 / P3.4) 0 = Normal Priority 1 = High Priority
3	T0P31PRI	Interrupt 3 Priority Level (Timer 0 / P3.1) 0 = Normal Priority 1 = High Priority
2	SRP30PRI	Interrupt 2 Priority Level (SPI RX / P3.0) 0 = Normal Priority 1 = High Priority
1	STP33PRI	Interrupt 1 Priority Level (SPI TX / P3.3) 0 = Normal Priority 1 = High Priority
0	INT0P32PRI	Interrupt 0 Priority Level (INT0 / P3.2) 0 = Normal Priority 1 = High Priority

TABLE 166:INTERRUPT PRIORITY 2 REGISTER - INTPRI2 SFR E3H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	PC1P07PRI	Interrupt 15 Priority Level (Port Chg 1 / P0.0) 0 = Normal Priority 1 = High Priority
6	AIP06PRI	Interrupt 14 Priority Level (WDT, AU / P0.6) 0 = Normal Priority 1 = High Priority
5	PWHP05PRI	Interrupt 13 Priority Level (PWM7:4 timer / P0.5) 0 = Normal Priority 1 = High Priority
4	PWLP04PRI	Interrupt 12 Priority Level (PWM3:0 timer / P0.4) 0 = Normal Priority 1 = High Priority
3	PWCP02PRI	Interrupt 11 Priority Level (PWC0, PWC1 / P0.3) 0 = Normal Priority 1 = High Priority
2	INT10P01PRI	Interrupt 10 Priority Level (UARTs Coll, I²C Lost Arbitration / P0.2) 0 = Normal Priority 1 = High Priority
1	I2CP01PRI	Interrupt 9 Priority Level (I²C / P0.1) 0 = Normal Priority 1 = High Priority
0	T2P00PRI	Interrupt 8 Priority Level (Timer 2 / P0.0) 0 = Normal Priority 1 = High Priority

15.4 Pin Inversion Setting

TABLE 167: IMPACT OF PIN INVERSION SETTING ON PIN INTERRUPT SENSITIVITY

Pin Inversion	Interrupt Condition
0	Normal Interrupt Polarity Sensitivity
1	Inverted Interrupt Polarity Sensitivity

TABLE 168:INTERRUPT PIN INVERSION 1 REGISTER - IPININV1 SFR D6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	P37IINV	Interrupt 7 Pin Polarity 0 = P3.7 1 = P3.7 Inverted
6	P36IINV	Interrupt 6 Pin Polarity 0 = P3.6 1 = P3.6 Inverted
5	P35IINV	Interrupt 5 Pin Polarity 0 = P3.5 1 = P3.5 Inverted
4	P34IINV	Interrupt 4 Pin Polarity 0 = P3.4 1 = P3.4 Inverted
3	P31IINV	Interrupt 3 Pin Polarity 0 = P3.1 1 = P3.1 Inverted
2	P30IINV	Interrupt 2 Pin Polarity 0 = P3.0 1 = P3.0 Inverted
1	INT1IINV	Interrupt 1 Pin Polarity 0 = INT1 (P3.3) 1 = INT1 (P3.3) Inverted
0	INT0IINV	Interrupt 0 Pin Polarity 0 = INT0 (P3.2) 1 = INT0 (P3.2) Inverted

TABLE 169: INTERRUPT PIN INVERSION 2 REGISTER - IPININV2 SFR D7H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	P07IINV	Interrupt 15 Pin Polarity 0 = P0.7 1 = P0.7 Inverted
6	P06IINV	Interrupt 14 Pin Polarity 0 = P0.6 1 = P0.6 Inverted
5	P05IINV	Interrupt 13 Pin Polarity 0 = P0.5 1 = P0.5 Inverted
4	P04IINV	Interrupt 12 Pin Polarity 0 = P0.4 1 = P0.4 Inverted
3	P03IINV	Interrupt 11 Pin Polarity 0 = P0.3 1 = P0.3 Inverted
2	P02IINV	Interrupt 10 Pin Polarity 0 = P0.2 1 = P0.2 Inverted
1	P01IINV	Interrupt 9 Pin Polarity 0 = P0.1 1 = P0.1 Inverted
0	P00IINV	Interrupt 8 Pin Polarity 0 = P0.0 1 = P0.0 Inverted

15.5 Pin Interrupt Sensitivity Setting

The pin interrupt can be configured as level sensitive or edge triggered. The pin interrupt sensitivity is set via the IPINSENSx and IPININVx registers. The following table summarizes the pin interrupt trigger condition settings for IPINSENx and IPININVx.

TABLE 170:IMPACT OF PIN SENSITIVITY AND PIN INVERSION SETTING ON PIN INTERRUPT

Pin Sensitivity	Pin Inversion	Interrupt Condition
0	0	High level on pin
0	1	Low level on pin
1	0	Rising edge on pin
1	1	Falling edge on pin

The following tables provide the bit definitions for the IPINSENS1 and IPINSENS2 registers. It is assumed that the corresponding IPININVx bit is set to 0. If the corresponding IPININVx bit is set to 1, the corresponding interrupt event will be inverted.

TABLE 171:INTERRUPT PIN SENSITIVITY 1 REGISTER - IPINSENS1 SFR E6H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	P37ISENS	Interrupt 7 Pin Sensitivity (IPININV1.7 = 0) 0 = P3.7 High Level 1 = P3.7 Rising Edge
6	P36ISENS	Interrupt 6 Pin Sensitivity (IPININV1.6 = 0) 0 = P3.6 High Level 1 = P3.6 Rising Edge
5	P35ISENS	Interrupt 5 Pin Sensitivity (IPININV1.5 = 0) 0 = P3.5 High Level 1 = P3.5 Rising Edge
4	P34ISENS	Interrupt 4 Pin Sensitivity (IPININV1.4 = 0) 0 = P3.4 High Level 1 = P3.4 Rising Edge
3	P31ISENS	Interrupt 3 Pin Sensitivity (IPININV1.3 = 0) 0 = P3.1 High Level 1 = P3.1 Rising Edge
2	P30ISENS	Interrupt 2 Pin Sensitivity (IPININV1.2 = 0) 0 = P3.0 High Level 1 = P3.0 Rising Edge
1	INT1ISENS	Interrupt 1 Pin Sensitivity (IPININV1.1 = 0) 0 = INT1 (P3.3) High Level 1 = INT1 (P3.3) Rising Edge
0	INT0ISENS	Interrupt 0 Pin Sensitivity (IPININV1.0 = 0) 0 = INT0 (P3.2) High Level 1 = INT0 (P3.2) Rising Edge

TABLE 172:INTERRUPT PIN SENSITIVITY 2 REGISTER - IPINSENS2 SFR E7H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	P07ISENS	Interrupt 7 Pin Sensitivity (IPININV2.7 = 0) 0 = P0.7 High Level 1 = P0.7 Rising Edge
6	P06ISENS	Interrupt 6 Pin Sensitivity (IPININV2.6 = 0) 0 = P0.6 High Level 1 = P0.6 Rising Edge
5	P05ISENS	Interrupt 5 Pin Sensitivity (IPININV2.5 = 0) 0 = P0.5 High Level 1 = P0.5 Rising Edge
4	P04ISENS	Interrupt 4 Pin Sensitivity (IPININV2.4 = 0) 0 = P0.4 High Level 1 = P0.4 Rising Edge
3	P03ISENS	Interrupt 3 Pin Sensitivity (IPININV2.3 = 0) 0 = P0.3 High Level 1 = P0.3 Rising Edge
2	P02ISENS	Interrupt 2 pin Sensitivity (IPININV2.2 = 0) 0 = P0.2 High Level 1 = P0.2 Rising Edge
1	P01ISENS	Interrupt 1 Pin Sensitivity (IPININV2.1 = 0) 0 = P0.1 High Level 1 = P0.1 Rising Edge
0	P00ISENS	Interrupt 0 Pin Sensitivity (IPININV2.0 = 0) 0 = P0.0 High Level 1 = P0.0 Rising Edge

15.6 Pin Interrupt Flags

For each pin interrupt there is an interrupt flag that can be monitored. When the selected interrupt event is detected on a given pin, the corresponding pin interrupt flag is set to 1 by the system.

The interrupt pin flags should be cleared before any pin interrupt is activated.

The pin interrupt Flag are not automatically cleared by the interrupt service routine RETI instruction. For this reason they must be cleared by the software before exiting the interrupt service routine.

The pin interrupt flags can be monitored via the software, even if the corresponding pin interrupt is not activated. If all the corresponding interrupts are routed to modules and all the interrupts are disabled, the IPINFLAGx registers can be used as general purpose scratchpad registers. However this is not recommended.

TABLE 173:INTERRUPT PIN FLAG 1 REGISTER - IPINFLAG1 SFR B8H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	P37IF	Interrupt 7 Pin Flag Set to 1 if P3.7 pin Interrupt occurs
6	P36IF	Interrupt 6 Pin Flag Set to 1 if P3.6 pin Interrupt occurs
5	P35IF	Interrupt 5 Pin Flag Set to 1 if P3.5 pin Interrupt occurs
4	P34IF	Interrupt 4 Pin Flag Set to 1 if P3.4 pin Interrupt occurs
3	P31IF	Interrupt 3 Pin Flag Set to 1 if P3.1 pin Interrupt occurs
2	P30IF	Interrupt 2 Pin Flag Set to 1 if P3.0 pin Interrupt occurs
1	INT1IF	Interrupt 1 Pin Flag Set to 1 if INT1 (P3.3) pin Interrupt occurs
0	INT0IF	Interrupt 0 Pin Flag Set to 1 if INT0 (P3.2) pin Interrupt occurs

TABLE 174:INTERRUPT PIN FLAG 2 REGISTER - IPINFLAG2 SFR D8H

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7	P07IF	Interrupt 15 Pin Flag Set to 1 if P0.7 pin Interrupt occurs
6	P06IF	Interrupt 14 Pin Flag Set to 1 if P0.6 pin Interrupt occurs
5	P05IF	Interrupt 13 Pin Flag Set to 1 if P0.5 pin Interrupt occurs
4	P04IF	Interrupt 12 Pin Flag Set to 1 if P0.4 pin Interrupt occurs
3	P03IF	Interrupt 11 Pin Flag Set to 1 if P0.3 pin Interrupt occurs
2	P02IF	Interrupt 10 Pin Flag Set to 1 if P0.2 pin Interrupt occurs
1	P01IF	Interrupt 9 Pin Flag Set to 1 if P0.1 pin Interrupt occurs
0	P00IF	Interrupt 8 Pin Flag Set to 1 if P0.0 pin Interrupt occurs

16 JTAG Interface

The VRS51L3xxx devices include a JTAG interface that enables programming of the onboard Flash as well as code debugging. To free up as many I/Os as possible, the JTAG interface pins are shared with regular I/O pins that can be used as general purpose I/Os when the JTAG interface is not being used.

The JTAG interface is mapped into the following pins:

TABLE 175: JTAG INTERFACE PIN MAPPING

JTAG Pin	Function	Corresponding Pin
TDI	JTAG Data Input	P4.3
TDO	JTAG Data Output	P4.2
CM0	Chip Mode 0	ALE
TMS	Test Mode Select	P4.1
TCK	JTAG Clock	P2.7

Activation of the JTAG interface is controlled by the CM0-ALE pin. The CM0-ALE pin includes an internal pull-up resistor and its state is sampled when a power-on reset occurs.

Forcing the CM0-ALE pin to 0 (logic low) during the power-on reset will activate the JTAG interface and put the VRS51L3xxx into JTAG mode.

When the device is in JTAG mode, the code will not execute after a power-on reset or a regular reset: To start the code, a "program run" command must be sent through the JTAG interface. This is handled by the Versa Ware JTAG software developed by Ramtron and available on Ramtron's web site at www.ramtron.com . This software provides an easy-to-use interface for device programming and in-circuit debugging. For more information on the debugger's features and use, please consult the Versa Ware JTAG software Help.

16.1 Impact of JTAG interface activation

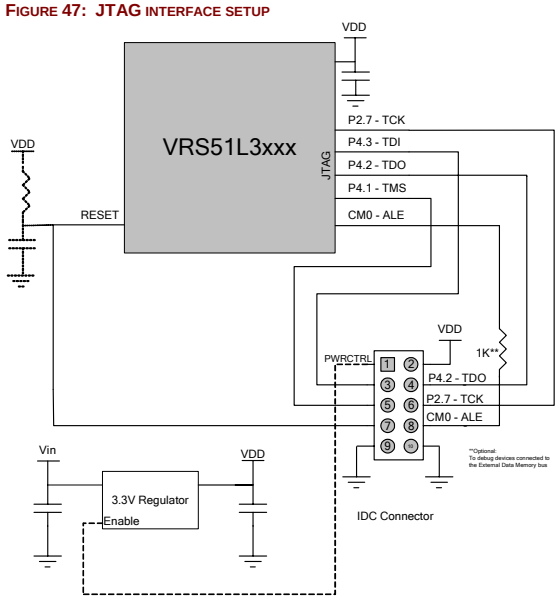
When the JTAG interface is activated, it has the following consequences on the VRS51L3xxx's operation:

- The PWM 7 output is deactivated. The PWM7 module can still be activated and used as a general purpose timer.
- The P2.7, P4.3, P4.2, P4.1 I/O pins become dedicated to the JTAG interface.
- To debug code accessing devices connected to the external data memory bus, a 1K Ohms resistor should be placed in the path of CM0-ALE to the Versa-JTAG interface module.

16.2 Board Level JTAG Interface Implementation

To perform in-circuit programming and debugging of the VRS51L3xxx, access to the device's JTAG port should be provided at the board level. The following figure demonstrates a typical setup for JTAG port access.

FIGURE 47: JTAG INTERFACE SETUP



The configuration of the IDC connector shown in Figure 35 matches that on the Versa-JTAG interface IDC connector.

Please note that if the target PCB's regulator includes a power control feature, the power control line can be routed to the JTAG IDC connector, enabling the Versa-JTAG to control the target board's power during device programming and in-circuit debugging. The other option is to leave the PWRCTRL pin of the IDC connector unconnected.

For the RESET control line, the presence of an external RC reset circuit is optional.

16.3 In-Circuit Debugger

The VRS51L3xxx devices include advanced debugging features that enable real-time, in-circuit debugging and emulation via the JTAG interface. When the debugger is activated, the upper 1024 bytes of the Flash memory are not available for user program.

The Debugger is intended to be used in conjunction with the Versa Ware JTAG software.

17 Flash Programming Interface (FPI)

The FPI module allows the processor to perform in-application management of the Flash memory content. The following operations are supported by the FPI module :

- Mass Erase
- Page Erase
- Byte Write
- Byte Read

Six SFR registers are associated with the FPI module operation, as shown in the table below:

TABLE 176: FLASH PROGRAMMING INTERFACE REGISTERS

SFR	Name	Function	Reset Value
E9h	FPICONFIG	Configures the FPI operations	34h
EAh	FPIADDRL	Address for operation (lower byte)	00h
EBh	FPIADDRH	Address for operation (upper byte)	00h
ECh	FPIDATAL	Data to write	00h
EDh	FPIDATAH	Upper byte of data to write	00h
EEh	FPICLKSPD	Clock speed during FPI operations	00h

The FPI module is activated by setting bit 0 of the PERIPHEN2 register. There are two ways to perform read and write operations to the Flash using the FPI module - standard 8-bit mode, which writes 1 byte at a time, and an extended 16-bit mode, which writes 2 bytes at a time (one word), effectively doubling the writing speed. In addition, whenever a write or read is performed, the address is incremented automatically by the FPI module, saving processor cycles and code space.

17.1 FPI Configuration Register

Flash operations are activated via the FPI configuration register. The following table describes the FPI configuration register:

TABLE 177: FPI CONFIGURATION REGISTER - FPICONFIG SFR E9H

7	6	5	4	3	2	1	0
R	R	R	R	R/W	R/W	R/W	R/W
0	0	1	1	0	1	0	0

Bit	Mnemonic	Description
7:6	FPILOCK[1:0]	These bits indicate the stage of the unlock operation: 00 : IAP protection on (no unlock steps done) 01 : IAP first unlock step done: FPI_DATA_LO received 0xAA 10 : IAP protection off: second step done (FPI_DATA_LO received 0x55) 11 : Disables write/erase operations until the next system reset. This occurs if a wrong sequence is used.
5	FPIIDLE	When set to 1 by the FPI module it indicated that the previous FPI operation completed successfully.
4	FPIRDY	Indicates that the FPI is idle in all modes except "write byte" mode, in which the double buffer is ready for a new value
3	RESERVED	Keep this bit at 0
2	FPI8BIT	0 = FPI operates in 16-bit mode 1 = FPI operates in 8-bit mode
[1:0]	FPITASK[1:0]	Operation: 00: Read Mode 01: Mass Erase 10: Page Erase 11: Write (Writing to FPIDATAL will starts the Write operation) Note that actions are only started if FPIRDY is high, otherwise the action is cancelled

FPIIDLE Bit

This bit indicates whether the previous action is complete and the FPI is idle. The FPIIDLE bit must be checked before performing any FPI operation, to ensure that the module is ready.

FPIRDY Bit

When writing a stream of bytes or words, this bit indicates whether the FPI is ready for the next write. Note that AAh then 55h must first be written in order to unlock the FPI module.

FPI8BIT Bit

The FPI8BIT bit of the FPICONFIG register defines whether the FPI module read and write operations will be performed in 8- or 16-bit format. When the FPI8BIT bit is set to 1, the FPI module will operate in 8-bit mode. The 16-bit address of the Flash memory, where the FPI operation will be performed, is defined by the value of the FPIADDRH and FPIADDRL registers.

When the FPI module is used to write data into the Flash memory, the FPIDATAL register holds the value of the data to be written. When the FPI module is

used to read the Flash, the read value is returned via the FPIDATAL register.

When the FPI8BIT bit is cleared, the FPI module will operate in 16-bit mode. In this case, the address range is defined by a 15-bit address (0000h to 7FFFh) and must be written into the FPIADDRH and FPIADDRL registers.

When a 16-bit FPI write operation is performed, the 16-bit data must be stored in the FPIDATAH and FPIDATAL registers. When a Flash memory read operation is performed, the 16-bit data will be returned to the FPIDATAH and FPIDATAL registers.

17.2 FPI Flash Address and Data Registers

The FPIADDRH and FPIADDRL registers are used to specify the address where the IAP function will be performed.

TABLE 178: FPI ADDRESS HIGH FPIADDRH SFR EBH

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
FPIADDR[15:8]							

The FPIADDRH register contains the MSB of the destination address. For page erase operations, it contains the page number where page erase operations are performed.

TABLE 179:FPI ADDRESS LOW -FPIADDRL SFR EAH

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0
R/W							
FPIADDR[7:0]							

The FPIADDRL register contains the LSB of the destination address where the operation is performed. For page erase operations, it must contain the value 0x00.

The FPIDATAH and FPIDATAL registers contain the data byte(s) required to perform the FPI function.

TABLE 180: FPI DATA HIGH - FPIDATAH SFR EDH

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
FPIDATA[15:8]							

When Read: MSB of last word read[15:8] from Flash
When Write: Byte[15:8] to write in Flash

TABLE 181:FPI DATA LOW - FPIDATAL SFR ECH

7	6	5	4	3	2	1	0
R/W, Reset = 0x00							
FPIDATA[7:0]							

Read: Last read byte[7:0] from Flash

Writing to this byte in 'FPI write mode' triggers the FPI state machine to start the write action.

17.3 FPI Clock Speed Control Register

The FPI clock speed control register sets the FPI module to an optimal speed based on the speed of the system clock.

TABLE 182:FPI CLOCK SPEED CONTROL REGISTER - FPICLKSPD SFR EEH

7	6	5	4	3	2	1	0
R	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Mnemonic	Description
7:4	Unused	
3:0	FPICLKSPD [3:0]	Specifies speed of the system clock entering the FPI module Frequency range: 0000 : 20MHz to 40 MHz 0001 : 10MHz to 20 MHz 0010 : 5MHz to 10 MHz 0011 : 2.5MHz to 5 MHz 0100 : 1.25MHz to 2.5 MHz 0101 : 625kHz to 1.25 MHz 0110 : 312.5kHz to 625 kHz 0111 : 156.25kHz to 312.5 kHz 1000 : 78.12kHz to 156.25 kHz 1001 : 39.06kHz to 78.125 kHz 1010 : 19.53kHz to 39.0625 kHz Others : 9.76kHz to 19.53125 kHz

Use the settings found in the following table when using the FPI at a speed other than the nominal speed of the internal oscillator.

TABLE 183: SETTING THE FPICLKSPD REGISTER

Value	Range	
	Minimum	Maximum
0 (default)	20.000 MHz	40.000 MHz
1	10.000 MHz	20.000 MHz
2	5.000 MHz	10.000 MHz
3	2.500 MHz	5.000 MHz
4	1.250 MHz	2.500 MHz
5	625.000 KHz	1.250 MHz
6	312.500 KHz	625.000 KHz
7	156.250 KHz	312.500 KHz
8	78.125 KHz	156.250 KHz
9	39.063 KHz	78.125 KHz
10	19.531 KHz	39.063 KHz
Other	9.766 KHz	19.531 KHz

The FPICLKSPD register must be set to the corresponding system clock speed for proper operation of the FPI module. For example, a 20.0 MHz clock requires FPICLKSPD to be set to 1, while a 20.1 MHz clock requires FPICLKSPD to be set to 0. If FPICLKSPD is set incorrectly, the Flash write operation may not process correctly, causing data corruption.

17.4 FPI Write Protection

The VRS51L3xxx provides a safety mechanism to prevent accidental writing or erasing of the Flash. The following sequence must be written to the FPIDATAL register to unlock the FPI module each time a write is performed.

FPIDATAL ← AAh
FPIDATAL ← 55h

Not performing the above sequence will lock the FPI module until a reset of the VRS51L3xxx is performed. Bit 7 and 6 of the FPICONFIG provide the status of the FPI write protection circuitry.

17.5 Flash Read Operations

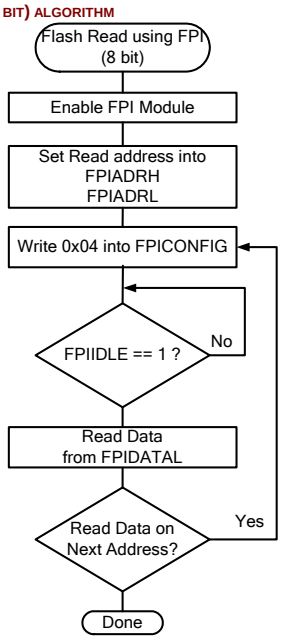
There are three ways to read directly from the VRS51L3xxx Flash memory:

1. Use the MOVC instruction
2. Use the FPI in 8-bit mode
3. Use the FPI in 16-bit mode

It may be preferable to use the FPI over the MOVC instruction, because some compilers will optimize code that repeatedly checks the Flash content. To do a Flash read using the FPI interface, perform the following steps:

- Make sure the FPI module is enabled.
- Set FPIADDRH and FPIADDRL to the appropriate address (see section 1.1.4).
- Write 00000X00 to the FPICONFIG register, where X = 1 if reading in 8-bit mode, and X = 0 if reading in 16-bit mode.
- Loop until FPIIDLE is raised.
- Get the results from FPIDATAH and FPIDATAL if in 16-bit mode, or from FPIDATAL if in 8-bit mode.

48: FPI FLASH READ (8 BIT) ALGORITHM



Example 1:
FPI Flash Read in 8-Bit Mode :

The following code sequence follows the above algorithm to read address ABCDh in 8-bit mode:

```
ORL PERHIPHEN2, #1 ; Enable FPI
MOV FPIADDRH, #0ABh ; Move in upper address
MOV FPIADDRL, #0CDh ; Move in lower address
MOV FPICONFIG, #04h ; Trigger the read in 8-bit mode

Wait:
   MOV A, FPICONFIG ; Get the FPI status
   JNB ACC.4, Wait ; Jump if not ready

; The read is now done. The result in FPIDATAL
```

Example 2:
FPI Flash Read in 16-Bit Mode Example

The following code sequence will read 16 bits from address ABCD:

```
#include <VRS51L3074.h>
unsigned char ucupper;
unsigned char uclower;

void readFPI(int address)
{
   unsigned char result;
PERIPHEN2 |= 1; /* Enable FPI */
FPIADDRH = (unsigned char) (address >> 8); /* Upper address */
FPIADDRL = (unsigned char) address; /* Lower address – automatically truncates */
FPICONFIG = 0; /* Trigger the read */
do
{
   result = FPICONFIG & 0x20; /* Check for the FPI_IDLE bit */
}
while(!result)
ucupper = FPIDATAH;
uclower = FPIDATAL;
}

void main()
{
   /*** SOME CODE***/
   readFPI(0x55e6);   /* This is address ABCD converted to 16 bit addressing */
/*** SOME CODE***/
   while(1);
}
```

17.6 FPI Flash Page Erase

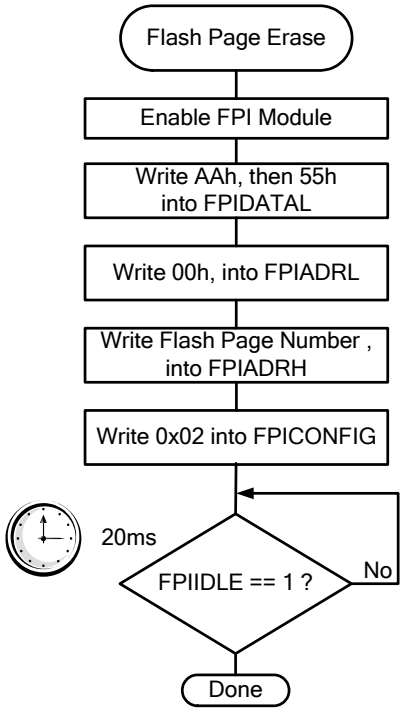
When storing nonvolatile data into Flash memory, it is necessary to erase the Flash before writing to it. Programming is done by byte or word boundary, while erase is done by page boundary. A page is a contiguous block of 512 addresses. Page numbers can be calculated from the following formula:

Page = address / 512

Page 0 contains all the addresses from 0000h to 01FFh, page 1 contains all the addresses from 0200h to 03FFh, and so on. There are 128 Flash pages on the VRS51L3xxx (64KB Flash).To erase a page, follow these steps:

1. Ensure that the FPI module is enabled.
2. Write AAh to the FPIDATAL register.
3. Write 55h to the FPIDATAL register.
4. Write 0 to the FPIADDRL register.
5. Write the page number to the FPIADDRH register.
6. Write 2 to the FPICONFIG register.
7. Wait for FPIIDLE to go high.

FIGURE 49: FPI FLASH PAGE ERASE ALGORITHM



Example1:

FPI Page Erase

This code sequence will erase page 64:

```
ORL PERHIPHEN2, #1 ; Enable FPI
MOV FPIDATAL, #0AAh ; UNLOCK 1
MOV FPIDATAL, #055h ; UNLOCK 2
MOV FPIADDRL, #0 ; Move in 0
MOV FPIADDRH, #64 ; Move in page number
MOV FPICONFIG, #2 ; Trigger the page erase

Wait:
   MOV A, FPICONFIG ; Get the FPI status
   JNB ACC.4, Wait ; Jump if not ready
; The page is now erased
```

17.7 FPI Flash Mass Erase

It is possible to completely erase the Flash memory from within a program. To do so, the following steps must be performed:

1. Make sure that the FPI module is enabled.
2. Write AAh to the FPIDATAL register.
3. Write 55h to the FPIDATAL register.
4. Write 1 to the FPICONFIG register.
5. If still possible, wait for FPIIDLE to go high.

Warning: At this point, the Flash should be totally erased. If running from external memory, ensure that the program is copied back to its locations in Flash with write commands. Step 5 can only be performed if executing code from external SRAM.

17.8 Flash Write Operations

There are two methods to write to the Flash:

- o 8-bit double buffered
- o 16-bit double buffered

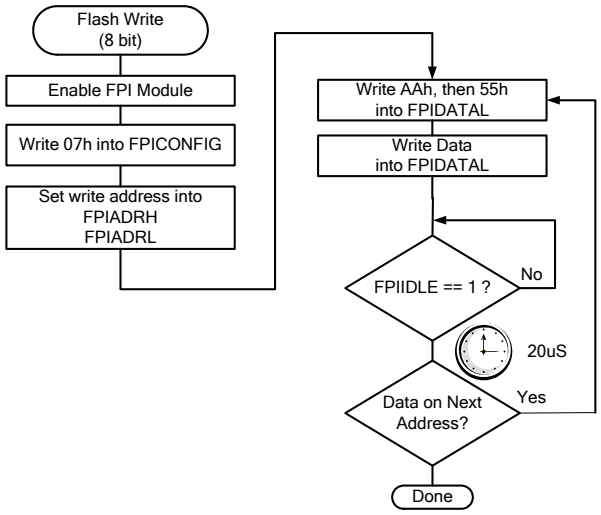
Depending on the complexity and the amount of Flash to be written, one mode may be more efficient than the other: 8-bit mode is more suited to programming a few bytes of data, while 16-bit mode is more suited to memory dumping.

17.9 FPI Flash Write - 8-bit mode

Follow the steps below to write in 8-bit mode:

1. Make sure the FPI module is enabled.
2. Write 7 to the FPICONFIG register.
3. Set FPIADDRH and FPIADDRL to the appropriate addresses.
4. Write AAh to the FPIDATAL register.
5. Write 55h to the FPIDATAL register.
6. Write data to the FPIDATAL register (this triggers the operation).
7. If complete, wait for FPIDLE to go high. If there are more bytes to be written at a different address, return to step 3. If the next address is contiguous, go to step 4 instead.

FIGURE 50: FPI FLASH BYTE WRITE ALGORITHM



Note that the address the data is written to will be automatically incremented for the next byte. As such, the address only needs to be written once per data stream (assuming that a contiguous block is written), as shown in the following example.

Code Example:

FPI Flash Write in 8-Bit Mode Example:

```
//************************************************
//* FPI Flash Write 8bit Mode Example  *
//************************************************
#include <VRS51L3074.h>
/*
This function uses the FPI module to write a null terminated string to flash
*/
void copy_to_Flash(int address, char *str)
{
 unsigned char ready; /* Is the FPI idle? */

 PERIPHEN2 |= 1;     /* Enable FPI */

/* Upper address */
 FPIADRH = (unsigned char) (address >> 8);
/* Lower address - automatically truncates */
 FPIADRL = (unsigned char) address;
 FPICONFIG = 7;  /* Trigger the write
  in 8 bit mode */

 while(*str)   /* while not null */
   {
   FPIDATAH = 0x00;
   FPIDATAL = 0xaa; /* 1st step unlock */
   FPIDATAL = 0x55; /* 2nd step unlock */
   FPIDATAL = (unsigned char)(*str);

   /* Wait for the buffer to be ready */
/* The operation is not finished, check for FPI_READY */
   do
      {
      ready = FPICONFIG & 0x10;
      }while(!ready);
   str++;
   }

/* Null character encountered, write an
  additional 0 to memory */
     FPIDATAL = 0xaa; /* 1st step unlock */
 FPIDATAL = 0x55; /* 2nd step unlock */
 FPIDATAL = 0;    /* End in null - this avoids
          having  to pass the string
          length */

/* The operation is finished, check for FPI_IDLE instead of FPI_READY */
   do
{
   ready = FPICONFIG & 0x10;
   }while(!ready);

 return;
}
void main(void)
{

 /*** CODE ***/
 copy_to_Flash(0x3000, "Ramtron Inc");
 copy_to_Flash(0x4000, "Microsystems connecting two worlds");

 /*** CODE ***/

 while(1);
}
```

17.10 FPI Flash Write - 16-bit mode

Follow the steps below to write in 16-bit mode:

1. Make sure the FPI module is enabled.
2. Write 3 to the FPICONFIG register.
3. Set FPIADDRH and FPIADDRL to the appropriate addresses (remember to convert to 16-bit addressing).
4. Write AAh to the FPIDATAL register.
5. Write 55h to the FPIDATAL register.
6. Write data to the FPIDATAL register (this triggers the operation).
7. If complete, wait for FPI_IDLE to go high. If there are more bytes to be written at a different address, return to step 3. If the next address is contiguous, go to step 4 instead.

Note that the address the data is written to will be automatically incremented for the next byte. As such, the address only needs to be set once per data stream (assuming a contiguous region is written), as shown in the following example.

Code Example:

FPI Flash Write in 16-Bit Mode Example:

This routine copies 512 bytes (1 page) of external SRAM to the Flash memory at address E000h + XRAM. The R0 and R1 registers contain the starting address of the page to copy.

```
//***********************************************
//* FPI Flash Write 16-bit Mode Example  *
//***********************************************
WRITE_PAGE:
  PUSH DPH0             ;PUSH THE DATA POINTER
  PUSH DPL0
  PUSH ACC          ;PUSH THE VAR. TO BE USED
  PUSH B
  MOV ACC, R2
  PUSH ACC

  MOV DPH0, R1      ;LOAD THE DATA POINTER
  MOV DPL0, R0
  MOV R2, #255      ;LOOP COUNTER (511 BYTES)
  ORL PERHIPHEN2, #1    ;ENABLE FPI MODULE
  MOV FPICONFIG, #3     ;ENABLE WRITING IN 16 BIT ;MODE
; SET THE ADDRESS MUST BE 16 BITS (ADDRESS / 2)
  CLR C                         ;CLEAR THE CARRY FLAG
  MOV A, R1
  RRC A                         ;CHECK IF THERE WILL BE A CARRY
  CLR A                         ;DOES NOT AFFECT CARRY BIT
  RRC A                         ;SETS A TO 80h IF R1 WAS ODD, OR
                                ;KEEPS IT 0


MOV FPIADRL, A                  ;SET LOWER ADDRESS
  MOV A, R1
  RR A                          ;DIVIDE ADDRESS BY 2
  ADD A, #7                     ;ADDS E000H TO THE ADDRESS
                                ;(E000 / 2 = 7000)

  MOV FPIADRH, A ;              SET UPPER ADDRESS

  WRITE_PAGE_LOOP:

    MOV FPIDATAL, #0AAh     ;UNLOCK STEP 1
    MOV FPIDATAL, #055h     ;UNLOCK STEP 2
    MOVX A, @DPTR
    MOV B, A
    INC DPTR                ;NEXT BYTE

    MOVX A, @DPTR
    INC DPTR                ;NEXT BYTE
    MOV FPIDATAH, A         ;SET THE UPPER VALUE
    MOV FPIDATAL, B         ;SET THE LOWER VALUE
;AND START THE WRITE

WRITE_PAGE_LOOP_WAIT:
    MOV A, FPICONFIG        ;CHECK TO SEE IF THE
                            ;BUFFER IS READY
                            ;JUMP IF FPI_READY IS NOT HIGH
    JNB ACC.4               ,WRITE_PAGE_LOOP_WAIT

    DJNZ R2                 ,WRITE_PAGE_LOOP

      ;NOW WRITE THE LAST WORD (BYTE 511 AND 512)

    MOV FPIDATAL, #0AAh     ;UNLOCK STEP 1
    MOV FPIDATAL, #055h     ;UNLOCK STEP 2

    MOVX A, @DPTR
    MOV B, A
    INC DPTR                ;NEXT BYTE

    MOVX A, @DPTR
    INC DPTR                ;NEXT BYTE
;(not necessary)
    MOV FPIDATAH, A         ;SET THE UPPER VALUE
    MOV FPIDATAL, B         ;SET THE LOWER VALUE
;AND START THE WRITE
WRITE_PAGE_LAST_WAIT:
    MOV A, FPICONFIG        ;CHECK TO SEE IF THE
                            ;BUFFER IS READY
 JUMP IF FPI_IDLE IS NOT HIGH (LAST WORD)
JNB ACC.4                   , WRITE_PAGE_LAST_WAIT

;RESTORE VARIABLES USED
POP B
POP ACC
    MOV R3, ACC
    POP ACC
    POP DPL0
    POP DPH0
    RET                             ;RETURN TO CALLER
```

Code Example:

FPI Flash Write, Read, Sector Erase Example

```
//-------------------------------------------------------------------------------------------------//
// V2K_FPI_Flash_WR_RD_SErase_8bit_SDCC.c  //
///-----------------------------------------------------------------------//
//
// DESCRIPTION:     VRS51L2070 / VRS51L3074
//                FPI 8 bit Write, 8 bit Read & Sector Erase Demonstration Program.
//-------------------------------------------------------------------------------------------------//
#include <VRS51L3074_SDCC.h>
void main (void) {

    char tablewr[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55}; //Data Table to copy into Flash
    char tablerd[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; //Data Table to copy data from
Flash
    char count = 0x00;                                  //Counter variable

   // Performing FPI 8 bit Write
    PERIPHEN2 |= 0x01;                                  //Enable the FPI Module
    FPICLKSPD = 0x00;                                   //Set FPI module clock speed according to
                                                        //current System clock speed
    //--Configure the FPI module Start address
    FPIADDRH = 0x51;                                    //Set MSB of the address
    FPIADDRL = 0x00;                                    //Set LSB of the address

    //Set FPI for 8 bit Write
    FPICONFIG = 0x07;                                   //Bit 2 = 1:        Set 8 bit mode
                                                        //Bit [1:0] = 11:   Set Write operation

    //--Copy the content of tablewr[] into the Flash memory
    for(count = 0x00;count <= 0x05; count++)
        {
        //Write the pointed data byte into the FLASH
        FPIDATAL = 0xAA;                                //Unlock security 1
        FPIDATAL = 0x55;                                //Unlock security 2
        FPIDATAL = tablewr[count];                      //Write Data into Flash memory

        //-Check that FPI module is ready
        //(This is Required only if the Program executes from 4K SRAM)
        while((FPICONFIG & 0x10) == 0x00);

        }//end of for loop writing data into Flash

//--Use the FPI module to read the data written into the Flash to the tablerd[] table
    //--Reset the FPI Flash start address
    FPIADDRH = 0x51;                                    //Set MSB of the address
    FPIADDRL = 0x00;                                    //Set LSB of the address

    for(count = 0x00;count <= 0x05; count++)
        {
        //Set FPI for 8bit read operations
        FPICONFIG = 0x04;                               //Bit 2 = 1:        Set 8 bit mode
                                                        //Bit [1:0] = 00:   Set Read operation

        //-Check that FPI module is ready
        //This is Required only if the Program executes from 4K SRAM
        while((FPICONFIG & 0x10) == 0x00);

        tablerd[count] = FPIDATAL;                      //Copy data byte present into the
                                                         //FPIDATAL register into tablerd[]
        }//end of for loop comparing data in flash with Table content
    //--Perform a sector Erase of the Flash memory
    //Page address is defined by:
    //        FPIADRH = Round Down(address / 512)

       //Erase Page that covers addresses range 5000h - 51FFh
       FPIDATAL = 0xAA;
       FPIDATAL = 0x55;
       FPIADDRL  = 0x00;
       FPIADDRH  = 0x28;
       FPICONFIG = 0x02;

       //-Check that FPI module is ready
        //This is Required only if the Program executes from 4K SRAM

        while((FPICONFIG & 0x10) == 0x00);

    while(1);

}// End of main
```

17.11 Tips on Using the FPI Interface

The following tips can be used to get the most out of the IAP features on the VRS51L3xxx devices.

- Shorter programming time can be achieved if the FPI Flash write routines are run from the 4KB external SRAM, as the circuitry that reads instructions from the Flash does not interfere with the FPI module.
- The Flash must be erased before reprogramming, and the same value should not be written more than once to the same Flash address, unless an erase cycle is performed in between writes.
- To maximize the endurance of the Flash memory, FPI Flash page erase operations should be done sparingly.
- The FPI mass erase function will erase the entire Flash memory, including code already programmed.
- IAP can be performed even if the Flash protection is enabled. It is the responsibility of the programmer not to reveal the Flash information of a secured device via the IAP.
- When write operations are performed at the boundaries of two contiguous blocks of memory, the address will automatically increment to the next byte/word after a write cycle. This can save processor cycles.
- The FPI read can be used to perform Flash memory reads, however using the MOVC instruction is more efficient.
- Make sure that the location being written to does not interfere with the program running in the Flash.

18 External Crystal

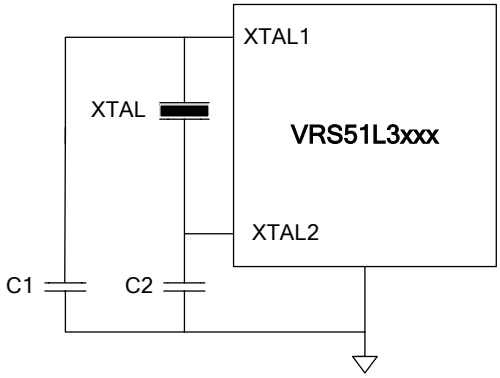
By default, the VRS51L3xxx devices derive its clock from its internal 40MHz oscillator. It is also possible to use external crystal as device's clock source. The crystal connected to the oscillator input should be parallel cut type, operating in fundamental mode.

The addition of 15 to 20pF load capacitors is recommended. See the following figure for a connection diagram.

Note: Oscillator circuits may differ with different crystals or ceramic resonators in higher oscillation frequency. Crystals or ceramic resonator characteristics may also vary from one manufacturer to another.

The user should review the technical literature associated with specific crystal or ceramic resonator or contact the manufacturer to select the appropriate values for the external components.

FIGURE 51: VRS51L3XXX EXTERNAL CRYSTAL OSCILLATOR CONFIGURATION



19 Power-On Reset Time

Upon a power on reset, the start up time of the device is 46 to 52us

20 Operating Conditions

20.1 Absolute Maximum Ratings

Parameter	Min.	Max.	Unit	Notes
Supply voltage input (VDD – VSS)	0	3.6	V	
I/O input voltage all except P4.6 & P4.7	-0.5V	5.5V	V	
I/O input voltage P4.6 & P4.7 only	VSS-0.5	VCC+0.5	V	
Maximum I/O current (sink/source) QFP-64 package		100	mA	Preliminary
Storage temperature	-55	125	ºC	

20.2 Nominal operating conditions

TABLE 184: OPERATING CONDITIONS

Symbol	Description	Min.	Typ.	Max.	Unit	Remarks
TA	Operating temperature	-40		+85	ºC	
VCCV	Supply voltage	3.0	3.3	3.6	V	
BOV	Brown-Out voltage	2.9		3.0	V	
Fextosc 40	Ext. Oscillator Frequency	0	-	40	MHz	
FextCY	Ext. Crystal frequency	4		40	MHz	
		32		100	KHz	
Internal Oscillator Operating frequency		39.7	40	40.3	MHz	At 25C
Internal Oscillator temperature stability			+/-2		%	0 to +70ºC
Internal Oscillator temperature stability			+/-3.25		%	-40 to +85ºC
F-RAM data retention		45	-	-	Years	
F-RAM byte Write		1.1			uS	
F-RAM Byte Read		0.4			uS	
Flash Endurance(Erase / Write cycles)		20K			Cycles	
Flash Data retention		100			Years	At room temperature
Flash Page Erase duration		20			ms	
Flash Byte/Word programming time		20			uS	

20.3 DC Characteristics

VCC = 3.3V, Temp = 25ºC, No load on I/Os

TABLE 185: DC CHARACTERISTICS

Symbol	Parameter	Valid	Min.	Typ	Max.	Unit	Test Conditions
VIL1	Input Low Voltage	Port 0,1,2,3,4,5,6	-0.35		0.80	V	VCC=3.3V
VIL2	Input Low Voltage	RESET, XTAL1	-0.35		0.80	V	VCC=3.3V
VIH1	Input High Voltage	Port 0,1,2,3,4,5,6	2.0		5.5	V	VCC=3.3V
VI H2	Input High Voltage	RES, XTAL1	2.0		5.5	V	VCC=3.3V
VOL1	Output Low Voltage	Port 0,1,2,3,4,5,6,ALE			0.4	V	IOL = Rated I/O max current
VOH2	Output High Voltage	Port 0,1,2,3,4,5,6,ALE	2.4V	Vcc – 0.3V		V	Max Rated I/O Current
ILI	Input Leakage Current	Port 0,1,2,3,4			10	uA	(+/-)
R RES	Reset Equivalent Pull-up Resistance	RES	74	104	177	Kohm	
C˙10	Pin Capacitance				10	pF	Freq=1 MHz, Ta=25°C
ICC	Supply Current	VDD	17		32	mA	Active mode, 40MHz (Int. Oscillator)
			7.9		12	mA	Active mode, 10MHz (Int. Oscillator)
			6.2		8.5	mA	Active mode 5 MHz (Ext. Crystal)
			3.6		11	mA	Idle mode, oscillator running 40MHz

20.4 Timing Requirement of the External Clock

The following diagram shows the timing of an external clock driving the VRS51L3xxx input.

FIGURE 52: TIMING REQUIREMENT OF EXTERNAL CLOCK (VSS= 0.0V IS ASSUMED)

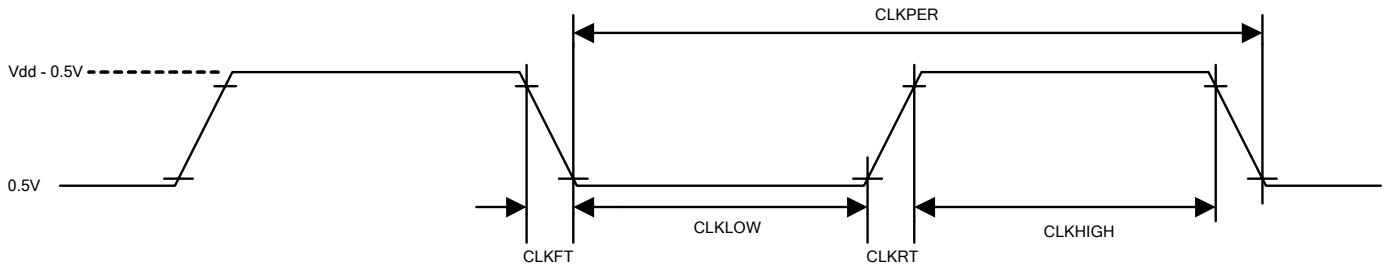


TABLE 186: EXTERNAL CLOCK TIMING REQUIREMENTS

Symbol	Parameter		Variable Fosc			Unit
			Min.	Typ	Max.	
CLKPER	Ext. clock period		25			nS
CLKLOW	Ext. clock low duration		12.5			nS
CLKHIGH	Ext. clock high duration		12.5			nS
CLKFT	Ext. clock fall time					nS
CLKRT	Ext. clock rise time					nS

21 VRS51L30xx QFP-64 Package

FIGURE 53: VRS51L30XX QFP-64 PACKAGE DRAWINGS

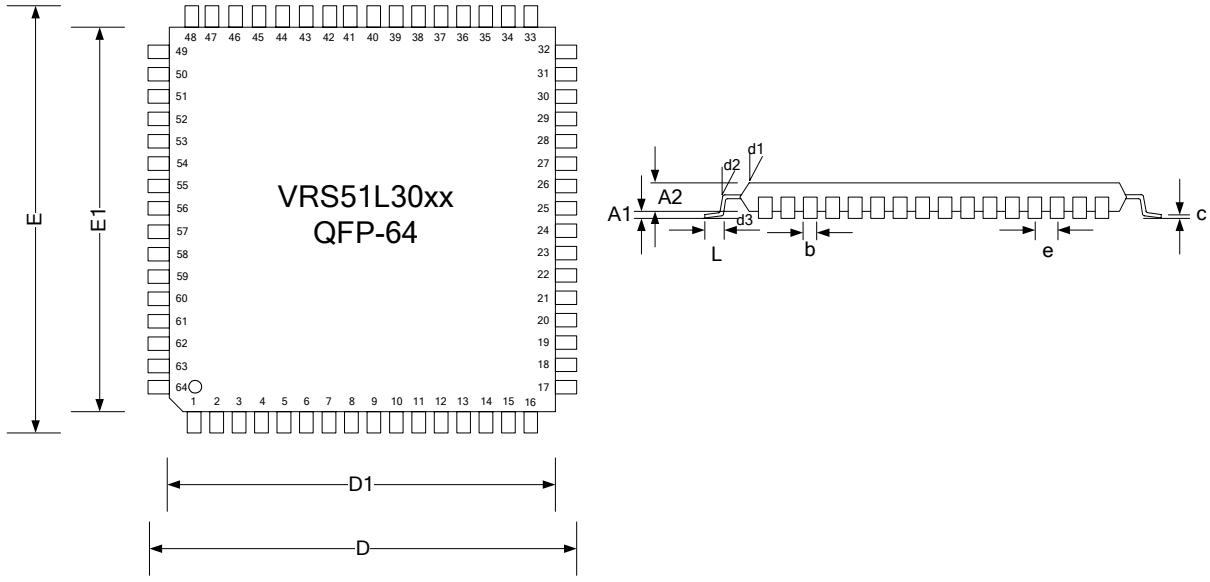


TABLE 187: DIMENSIONS OF QFP-64 PACKAGES

Symbol	Description	Dimension (mm)	Tolerance (mm, º) / Notes
D	Footprint	17.2	+/- 0.25
D1	Body size	14	+/- 0.10
E	Footprint	17.2	+/- 0.25
E1	Body size	14	+/- 0.10
A1	Stand-off	0.25	Max
A2	Body thickness	2.00	
L	Lead Length	0.88	+0.15 / -0.10
b	Lead width	0.35	+/- 0.05
c	L/C thickness	0.17	Max
e	Lead pitch	0.8	
d1	Body edge angle	10º	
d2	Lead angle	6º	+/- 4º
d3	Lead angle	0º to 7º	

22 VRS51L31xx QFP-44 Package

FIGURE 54: VRS51L31XX QFP-44 PACKAGE DRAWINGS

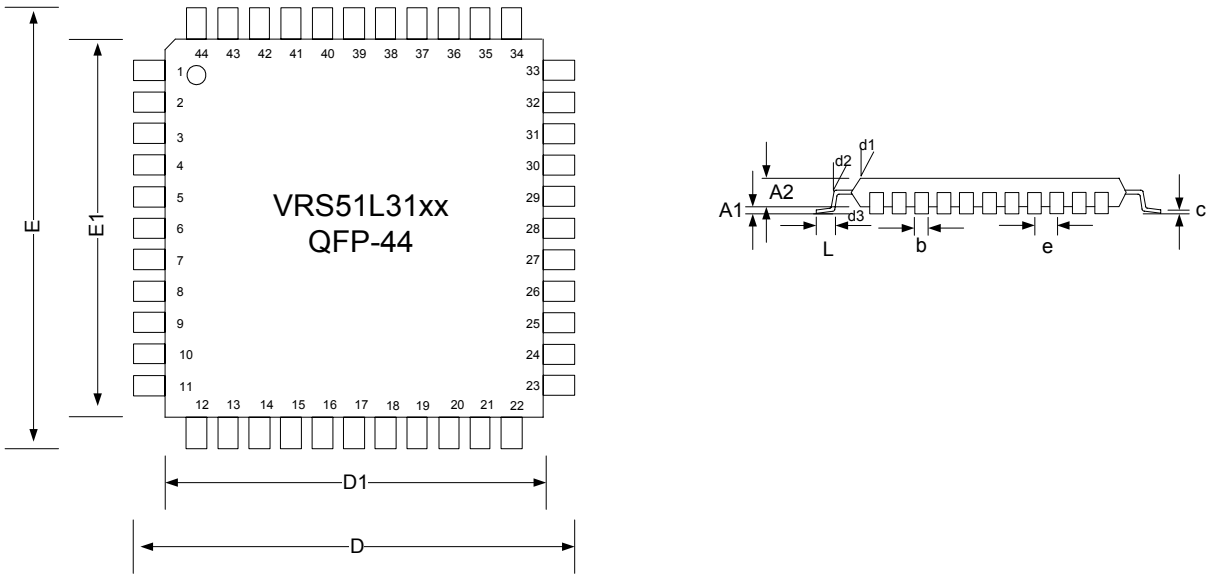
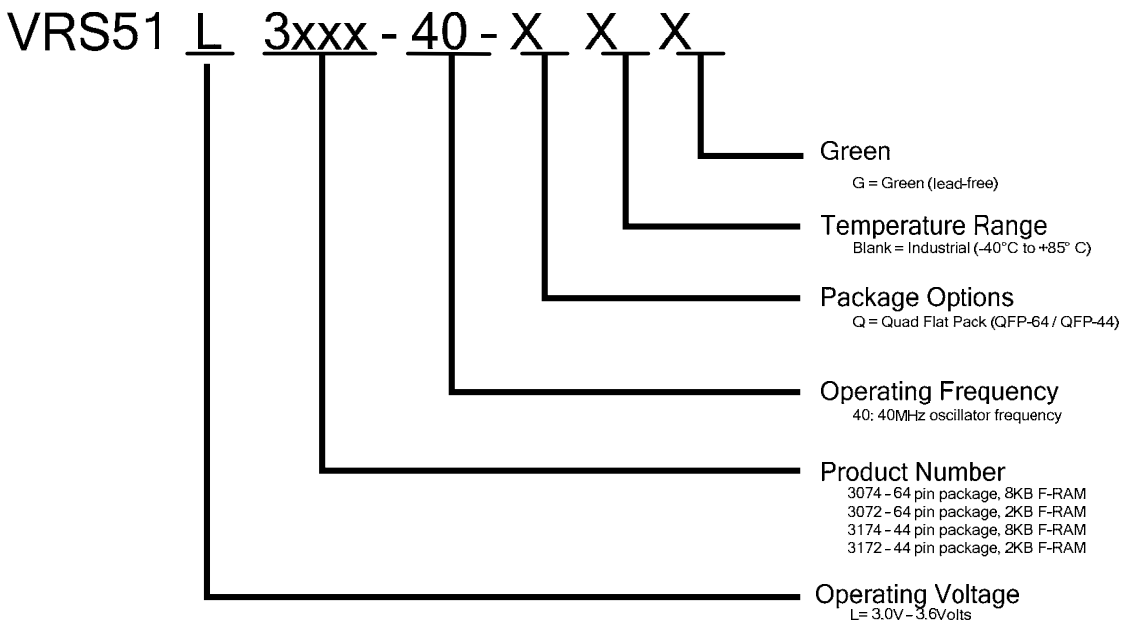


TABLE 188: DIMENSIONS OF QFP-44 PACKAGES

Symbol	Description	Dimension (mm)	Tolerance (mm, º) / Notes
D	Footprint	13.2	+/- 0.25
D1	Body size	10	+/- 0.10
E	Footprint	13.2	+/- 0.25
E1	Body size	10	+/- 0.10
A1	Stand-off	0.25	Max
A2	Body thickness	2.00	
L	Lead Length	0.88	+0.15 / -0.10
b	Lead width	0.35	+/- 0.05
c	L/C thickness	0.17	Max
e	Lead pitch	0.8	
d1	Body edge angle	10º	
d2	Lead angle	6º	+/- 4º
d3	Lead angle	0º to 7º	

23 Ordering Information

23.1 Device Number Structure



23.2 VRS51L3xxx Ordering Options

TABLE 189: VRS51L3XXX4 PART NUMBERING

Device Number	Flash Size	F-RAM Size	SRAM Size	Package Option	Voltage	Temperature	Frequency
VRS51L3074-40-QG	64KB	8KB	4352	QFP-64	3.0V to 3.6V	-40°C to +85°C	40MHz
VRS51L3072-40-QG	64KB	2KB	4352	QFP-64	3.0V to 3.6V	-40°C to +85°C	40MHz
VRS51L3174-40-QG	64KB	8KB	4352	QFP-44	3.0V to 3.6V	-40°C to +85°C	40MHz
VRS51L3172-40-QG	64KB	2KB	4352	QFP-44	3.0V to 3.6V	-40°C to +85°C	40MHz

Errata:

- *Readback of the content in the THx/TLx and RCAPxH/RCAPxL timer registers will return to 0x00 unless the corresponding timer is running or, for the timers 0 and 1, the timer gating bit is set.*
- *There is a problem with the watchdog timer reset that occurs when the VRS51L3xxx is running from an external crystal or an oscillator and the device have been programmed with Clock Divisor set to OFF in the Device Options settings of Versa Ware JTAG. The work around this problem is simple:*
  - *Program the device with Clock divider set to either Fosc/2, Fosc/4 or Fosc/8*
  - *In the beginning of user code, add the line: DEVCLKCFG = 0x00;*

Disclaimers

Right to make change - Ramtron reserves the right to make changes to its products - including circuitry, software and services - without notice at any time. Customers should obtain the most current and relevant information before placing orders.
Use in applications - Ramtron assumes no responsibility or liability for the use of any of its products, and conveys no license or title under any patent, copyright or mask work right to these products and makes no representations or warranties that these products are free from patent, copyright or mask work right infringement unless otherwise specified. Customers are responsible for product design and applications using Ramtron parts. Ramtron assumes no liability for applications assistance or customer product design.
Life support – Ramtron products are not designed for use in life support systems or devices. Ramtron customers using or selling Ramtron's products for use in such applications do so at their own risk and agree to fully indemnify Ramtron for any damages resulting from such applications.