MEGAWIN
MAKE YOU WIN

8051-Based MCU

MPC82E/L52
Data Sheet

Version: A10

2013/04 version A10

Features

- 1-T 80C51 Central Processing Unit
- **MPC82E/L52** with **8K** Bytes flash ROM
  - ISP memory zone could be optioned as 1.0KB, 2.0KB or 3.0KB
  - Two level code protections for flash memory access
  - Flash write/erase cycle: 20,000
  - Flash data retention: 100 years at 25℃
  - MPC82E/L52 Flash space mapping (Default)
    - AP Flash（0000h~17FFh）
    - IAP Flash（1800h~1BFFh）
    - ISP Flash（1C00h~1FFFh）（ISP Boot code）
- On-chip 256 bytes scratch-pad RAM
- Interrupt controller
  - 7 sources, four-level-priority interrupt capability
  - Two external interrupt inputs, INT0 and INT1
- Two 16-bit timer/counters, Timer 0 and Timer 1.
  - X12 mode enabled for Timer 0/1
- Programmable 16-bit counter/timer Array (PCA) with 2 channels PWM
  - Capture mode
  - 16-bit software timer mode
  - High speed output mode
  - 8-bit PWM mode
- Enhanced UART (S0)
  - Framing Error Detection
  - Automatic Address Recognition
- 8-bit ADC
  - Programmable throughput up to 100 ksps
  - 8 channel single-ended inputs
- Master/Slave SPI serial interface
- Programmable Watchdog Timer, one time enabled by CPU or power-on
- Maximum 15 GPIOs in 20-pin package.
  - Can be configured to quasi-bidirectional, push-pull output, open-drain output and input only.
- Multiple power control modes: idle mode and power-down mode
  - All interrupts can wake up IDLE mode
  - **2** sources to wake up Power-Down mode
- Low-Voltage Detector: VDD 3.7V for E-series and VDD 2.4V for L-series
- Operating voltage range:
  - **MPC82E52**: 4.5V~5.5V, minimum 4.5V requirement in flash write operation (ISP/IAP)
  - **MPC82L52**: 2.4V~3.6V, minimum 2.7V requirement in flash write operation (ISP/IAP)
- Operation frequency range: 25MHz(max)
  - **MPC82E52**: 0 – 25MHz @ 4.5V – 5.5V
  - **MPC82L52**: 0 – 12MHz @ 2.4V – 3.6V and 0 – 25MHz @ 2.7V – 3.6V
- Clock Source
  - External crystal mode and Internal RC Oscillator (IRCO, 6MHz)
- Operating Temperature:
  - Industrial (-40℃ to +85℃)*
- Package Types:
  - PDIP20: MPC82E/L52AE
  - SOP20: MPC82E/L52AS

- TSSOP20: MPC82E/L52AT

*: Tested by sampling.

Content

1. General Description

The **MPC82E/L52** is a single-chip microcontroller based on a high performance 1-T architecture 80C51 CPU that executes instructions in 1~6 clock cycles (about 6~7 times the rate of a standard 8051 device), and has an 8051 compatible instruction set. Therefore at the same performance as the standard 8051, the **MPC82E/L52** can operate at a much lower speed and thereby greatly reduce the power consumption.

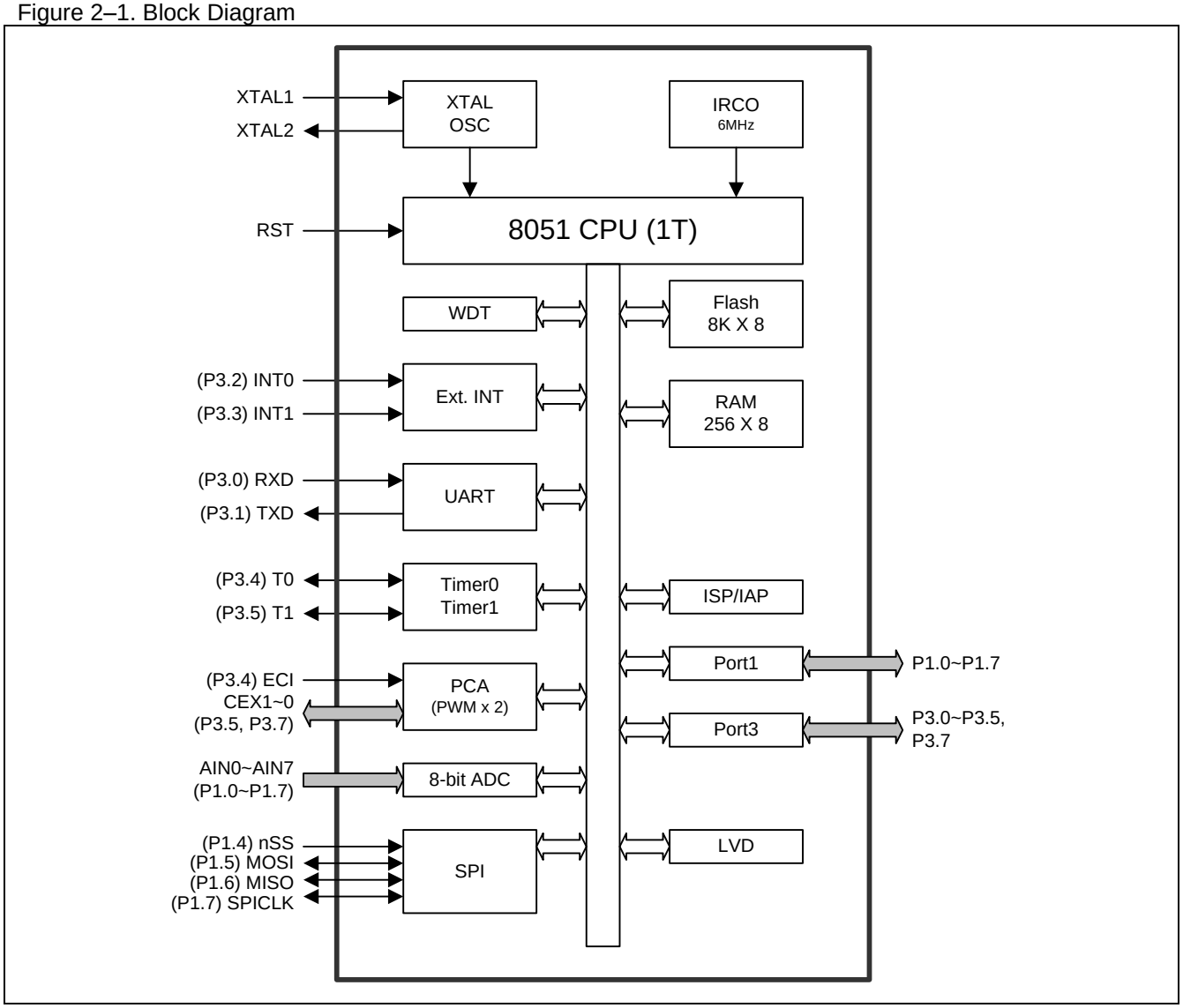
The **MPC82E/L52** has **8K** bytes of embedded Flash memory for code and data. The Flash memory can be programmed either in parallel writer mode or in ISP (In-System Programming) mode. And, it also provides the In-Application Programming (IAP) capability. ISP allows the user to download new code without removing the microcontroller from the actual end product; IAP means that the device can write non-volatile data in the Flash memory while the application program is running. There needs no external high voltage for programming due to its built-in charge-pumping circuitry.

The **MPC82E/L52** retains all features of the standard 80C52 with 256 bytes of scratch-pad RAM, two I/O ports, two external interrupts, a multi-source 4-level interrupt controller and two 16-bits timer/counters. In addition, the **MPC82E/L52** has one 8-btis ADC, a 2-channel PCA, SPI, Watchdog Timer, a Low-Voltage Detector, an on-chip crystal oscillator, an internal oscillator (IRCO, 6MHz), and a more versatile serial channel that facilitates multiprocessor communication (EUART).

The **MPC82E/L52** has multiple operating modes to reduce the power consumption: idle mode and power down mode. In the Idle mode the CPU is frozen while the peripherals and the interrupt system are still operating. In the Power-Down mode the RAM and SFRs' value are saved and all other functions are inoperative; most importantly, in the Power-down mode the device can be waked up by many interrupt or reset sources.

2. Block Diagram

Figure 2–1. Block Diagram



3. Special Function Register

3.1. SFR Map

Table 3–1. SFR Map

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8	--	CH	CCAP0H	CCAP1H	--	--	--	--
F0	B	--	PCAPWM0	PCAPWM1	--	--	--	--
E8	--	CL	CCAP0L	CCAP1L	--	--	--	--
E0	ACC	WDTCR	IFD	IFADRH	IFADRL	IFMT	SCMD	ISPCR
D8	CCON	CMOD	CCAPM0	CCAPM1	--	--	--	--
D0	PSW	--	--	--	--	--	--	--
C8	--	--	--	--	--	--	--	--
C0	--	--	--	--	--	ADCTL	ADCV	PCON2
B8	IP	SADEN	--	--	--	--	--	--
B0	P3	P3M0	P3M1	--	--	--	--	IPH
A8	IE	SADDR	--	--		--	--	--
A0	--	--	--	--	--	--	--	--
98	SCON	SBUF	--	--	--	--	--	--
90	P1	P1M0	P1M1	--	--	--	--	--
88	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	--
80	--	SP	DPL	DPH	SPISDAT	SPICTL	SPIDAT	PCON
	0/8	**1/9**	**2/A**	**3/B**	**4/C**	**5/D**	**6/E**	**7/F**

3.2. SFR Bit Assignment

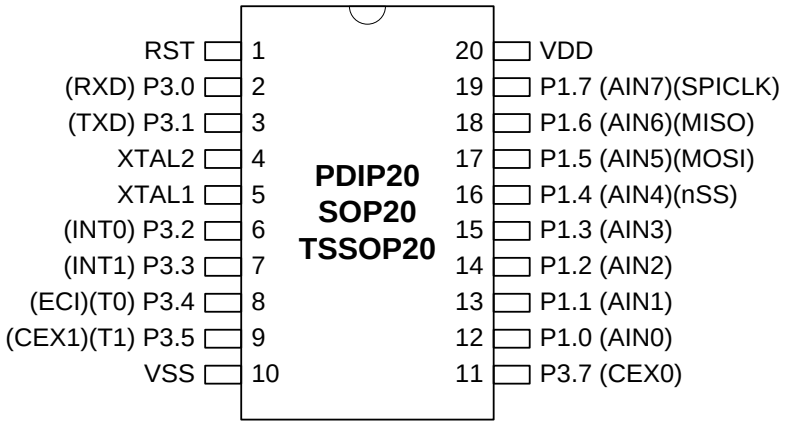
Table 3–2. SFR Bit Assignment

SYMBOL	DESCRIPTION	ADDR	BIT ADDRESS AND SYMBOL								RESET VALUE
			Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	
SP	Stack Pointer	81H									00000111B
DPL	Data Pointer Low	82H									00000000B
DPH	Data Pointer High	83H									00000000B
SPISTAT	SPI Status Register	84H	SPIF	WCOL	--	--	--	--	--	--	00xxxxxxB
SPICON	SPI Control Register	85H	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	00000100B
SPIDAT	SPI Data Register	86H									00000000B
PCON	Power Control Reg.	87H	SMOD	SMOD0	LVF	POF	GF1	GF0	PD	IDL	00110000B
TCON	*Timer Control*	*88H*	*TF1*	*TR1*	*TF0*	*TR0*	*IE1*	*IT1*	*IE0*	*IT0*	*00000000B*
TMOD	Timer Mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00000000B
TL0	Timer Low 0	8AH									00000000B
TL1	Timer Low 1	8BH									00000000B
TH0	Timer High 0	8CH									00000000B
TH1	Timer High 1	8DH									00000000B
AUXR	Auxiliary Register	8EH	T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	--	--	000000xxB
P1	*Port 1*	*90H*	*P1.7*	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*	*11111111B*
P1M0	P1 Mode Register 0	91H	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0	00000000B
P1M1	P1 Mode Register 1	92H	P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0	00000000B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	00000000B
SBUF	Serial Buffer	99H									xxxxxxxxB
IE	Interrupt Enable	A8H	EA	EPCA_ LVD	ESPI_ ADC	ES	ET1	EX1	ET0	EX0	00000000B
SADDR	Slave Address	A9H									00000000B
P3	*Port 3*	*B0H*	*P3.7*	*--*	*P3.5*	*P3.4*	*P3.3*	*P3.2*	*P3.1*	*P3.0*	*1x111111B*
P3M0	P3 Mode Register 0	B1H	P3M0.7	--	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0	0x000000B
P3M1	P3 Mode Register 1	B2H	P3M1.7	--	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0	0x000000B
IPH	Interrupt Priority High	B7H	--	PPCA_ LVD	PSPI_ ADC	PSH	PT1H	PX1H	PT0H	PX0H	x0000000B
IP	*Interrupt Priority Low*	*B8H*	--	PPCA_ LVD	PSPI_ ADC	*PSL*	*PT1L*	*PX1L*	*PT0L*	*PX0L*	*x0000000B*
SADEN	Slave Address Mask	B9H									00000000B
ADCTL	ADC Control	C5H	ADCON	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0	00000000B
ADCV	ADC Result Register	C6H	ADCV.7	ADCV.6	ADCV.5	ADCV.4	ADCV.3	ADCV.2	ADCV.1	ADCV.0	xxxxxxxxB
PCON2	Power Control 2 Reg.	C7H	--	--	--	--	--	CKS2	CKS1	CKS0	xxxxx000B
PSW	*Program Status Word*	*D0H*	*CY*	*AC*	*F0*	*RS1*	*RS0*	*OV*	*F1*	*P*	*00000000B*
CCON	*PCA Control Reg.*	*D8H*	*CF*	*CR*	*--*	*--*	*--*	*--*	*CCF1*	*CCF0*	*00xxxx00B*
CMOD	PCA Mode Reg.	D9H	CIDL	--	--	--	--	CPS1	CPS0	ECF	0xxxx000B
CCAPM0	PCA Module0 Mode	DAH	--	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x0000000B
CCAPM1	PCA Module1 Mode	DBH	--	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000B
ACC	*Accumulator*	*E0H*	*ACC.7*	*ACC.6*	*ACC.5*	*ACC.4*	*ACC.3*	*ACC.2*	*ACC.1*	*ACC.0*	*00000000B*
WDTCR	Watch-dog-timer Control register	E1H	WRF	--	ENW	CLW	WIDL	PS2	PS1	PS0	0x000000B xxx0xxxxB
IFD	ISP Flash data	E2H									11111111B
IFADRH	ISP Flash address High	E3H									00000000B
IFADRL	ISP Flash Address Low	E4H									00000000B
IFMT	ISP Mode Table	E5H	--	--	--	--	--	--	MS1	MS0	xxxxxx00B
SCMD	ISP Serial Command	E6H									xxxxxxxxB
ISPCR	ISP Control Register	E7H	ISPEN	SWBS	SWRST	CFAIL	--	WAIT.2	WAIT.1	WAIT.0	0000x000B
CL	PCA base timer Low	E9H	CL.7	CL.6	CL.5	CL.4	CL.3	CL.2	CL.1	CL.0	00000000B
CCAP0L	PCA module0 capture Low	EAH									00000000B
CCAP1L	PCA module1 capture Low	EBH									00000000B
B	*B Register*	*F0H*	*B.7*	*B.6*	*B.5*	*B.4*	*B.3*	*B.2*	*B.1*	*B.0*	*00000000B*
PCAPWM0	PCA PWM0 Mode	F2H	--	--	--	--	--	--	EPC0H	EPC0L	xxxxxx00B
PCAPWM1	PCA PWM1 Mode	F3H	--	--	--	--	--	--	EPC1H	EPC1L	xxxxxx00B
CH	PCA base timer High	F9H	CH.7	CH.6	CH.5	CH.4	CH.3	CH.2	CH.1	CH.0	00000000B
CCAP0H	PCA Module0 capture High	FAH									00000000B
CCAP1H	PCA Module1 capture High	FBH									00000000B

4. Pin Configurations

4.1. Package Instruction

Figure 4–1. MPC82E/L52 20-Pin Top View



4.2. Pin Description

Table 4–1. Pin Description

MNEMONIC	PIN NUMBER			I/O TYPE	DESCRIPTION
	20-Pin PDIP	20-Pin SOP	20-Pin TSSOP		
P1.0 (AIN0)	12	12	12	I/O	* Port 1.0. * AIN0: ADC channel-0 analog input.
P1.1 (AIN1)	13	13	13	I/O	* Port 1.1. * AIN1: ADC channel-1 analog input.
P1.2 (AIN2)	14	14	14	I/O	* Port 1.2. * AIN2: ADC channel-2 analog input.
P1.3 (AIN3)	15	15	15	I/O	* Port 1.3. * AIN3: ADC channel-3 analog input.
P1.4 (AIN4) (nSS)	16	16	16	I/O	* Port 1.4. * AIN4: ADC channel-4 analog input. * nSS: SPI Slave select.
P1.5 (AIN5) (MOSI)	17	17	17	I/O	* Port 1.5. * AIN5: ADC channel-5 analog input. * MOSI: SPI master out & slave in.
P1.6 (AIN6) (MISO)	18	18	18	I/O	* Port 1.6. * AIN6: ADC channel-6 analog input. * MISO: SPI master in & slave out.
P1.7 (AIN7) (SPICLK)	19	19	19	I/O	* Port 1.7. * AIN7: ADC channel-7 analog input. * SPICLK: SPI clock, output for master and input for slave.
P3.0 (RXD)	2	2	2	I/O	* Port 3.0. * RXD: UART serial input port.
P3.1 (TXD)	3	3	3	I/O	* Port 3.1. * TXD: UART serial output port.
P3.2 (INT0)	6	6	6	I/O	* Port 3.2. * INT0: external interrupt 0 input.
P3.3 (INT1)	7	7	7	I/O	* Port 3.3. * INT1: external interrupt 1 input.
P3.4 (T0) (ECI)	8	8	8	I/O	* Port 3.4. * T0: Timer/Counter 0 external input. * ECI: PCA external clock input.
P3.5 (T1) (CEX1)	9	9	9	I/O	* Port 3.5. * T1: Timer/Counter 1 external input. * CEX1: PCA module-1 external I/O.
P3.7 (CEX0)	11	11	11	I/O	* Port 3 bit-7. * CEX0: PCA module-0 external I/O.
XTAL2	4	4	4	O	* XTAL2: Output of on-chip crystal oscillating circuit.
XTAL1	5	5	5	I	* XTAL1: Input of on-chip crystal oscillating circuit.
RST	1	1	1	I	* RST: External RESET input, high active.
VDD	20	20	20	P	Power supply input. 5V input for E-type. 3.3V input for L-type.
VSS	10	10	10	G	Ground, 0 V reference.

5. 8051 CPU Function Description

5.1. CPU Register

PSW: Program Status Word

SFR Address = 0xD0 RESET = 0000-0000

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	F1	P
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

CY: Carry bit.
AC: Auxiliary carry bit.
F0: General purpose flag 0.
RS1: Register bank select bit 1.
RS0: Register bank select bit 0.
OV: Overflow flag.
F1: General purpose flag 1.
P: Parity bit.

The program status word(PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown above, resides in the SFR space. It contains the Carry bit, the Auxiliary Carry(for BCD operation), the two register bank select bits, the Overflow flag, a Parity bit and two user-definable status flags.

The Carry bit, other than serving the function of a Carry bit in arithmetic operations, also serves as the "Accumulator" for a number of Boolean operations.

The bits RS0 and RS1 are used to select one of the four register banks shown in Section "6.2 On-Chip Data RAM". A number of instructions refer to these RAM locations as R0 through R7.

The Parity bit reflects the number of 1s in the Accumulator. P=1 if the Accumulator contains an odd number of 1s and otherwise P=0.

SP: Stack Pointer

SFR Address = 0x81 RESET = 0000-0111

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

The Stack Pointer holds the location of the top of the stack. The stack pointer is incremented before every PUSH operation. The SP register defaults to 0x07 after reset.

DPL: Data Pointer Low

SFR Address = 0x82 RESET = 0000-0000

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

The DPL register is the low byte of the 16-bit DPTR. DPTR is used to access indirectly addressed XRAM and Flash memory.

DPH: Data Pointer High

SFR Address = 0x83 RESET = 0000-0000

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

The DPH register is the high byte of the 16-bit DPTR. DPTR is used to access indirectly addressed XRAM and Flash memory.

ACC: Accumulator

SFR Address = 0xE0 RESET = 0000-0000

7	6	5	4	3	2	1	0
ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

This register is the accumulator for arithmetic operations.

B: B Register

SFR Address = 0xF0 RESET = 0000-0000

7	6	5	4	3	2	1	0
B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

This register serves as a second accumulator for certain arithmetic operations.

5.2. CPU Timing

The **MPC82E/L52** is a single-chip microcontroller based on a high performance 1-T architecture 80C51 CPU that has an 8051 compatible instruction set, and executes instructions in 1~6 clock cycles (about 6~7 times the rate of a standard 8051 device). It employs a pipelined architecture that greatly increases its instruction throughput over the standard 8051 architecture. The instruction timing is different than that of the standard 8051.

In many 8051 implementations, a distinction is made between machine cycles and clock cycles, with machine cycles varying from 2 to 12 clock cycles in length. However, the 1T-80C51 implementation is based solely on clock cycle timing. All instruction timings are specified in terms of clock cycles. For more detailed information about the 1T-80C51 instructions, please refer Section “24 Instruction Set” which includes the mnemonic, number of bytes, and number of clock cycles for each instruction.

5.3. CPU Addressing Mode

Direct Addressing(DIR)
In direct addressing the operand is specified by an 8-bit address field in the instruction. Only internal data RAM and SFRs can be direct addressed.

Indirect Addressing(IND)
In indirect addressing the instruction specified a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed.

The address register for 8-bit addresses can be R0 or R1 of the selected bank, or the Stack Pointer.
The address register for 16-bit addresses can only be the 16-bit data pointer register – DPTR.

Register Instruction(REG)
The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the op-code of the instruction. Instructions that access the registers this way are code efficient because this mode eliminates the need of an extra address byte. When such instruction is executed, one of the eight registers in the selected bank is accessed.

Register-Specific Instruction
Some instructions are specific to a certain register. For example, some instructions always operate on the accumulator or data pointer, etc. No address byte is needed for such instructions. The op-code itself does it.

Immediate Constant(IMM)
The value of a constant can follow the op-code in the program memory.

Index Addressing
Only program memory can be accessed with indexed addressing and it can only be read. This addressing mode is intended for reading look-up tables in program memory. A 16-bit base register (either DPTR or PC) points to the base of the table, and the accumulator is set up with the table entry number. Another type of indexed addressing is used in the conditional jump instruction.
In conditional jump, the destination address is computed as the sum of the base pointer and the accumulator.

6. Memory Organization

Like all 80C51 devices, the **MPC82E/L52** has separate address spaces for program and data memory. The logical separation of program and data memory allows the data memory to be accessed by 8-bit addresses, which can be quickly stored and manipulated by the 8-bit CPU.

Program memory (ROM) can only be read, not written to. There can be up to **8K** bytes of program memory. In the **MPC82E/L52**, all the program memory are on-chip Flash memory, and without the capability of accessing external program memory because of no External Access Enable (/EA) and Program Store Enable (/PSEN) signals designed.

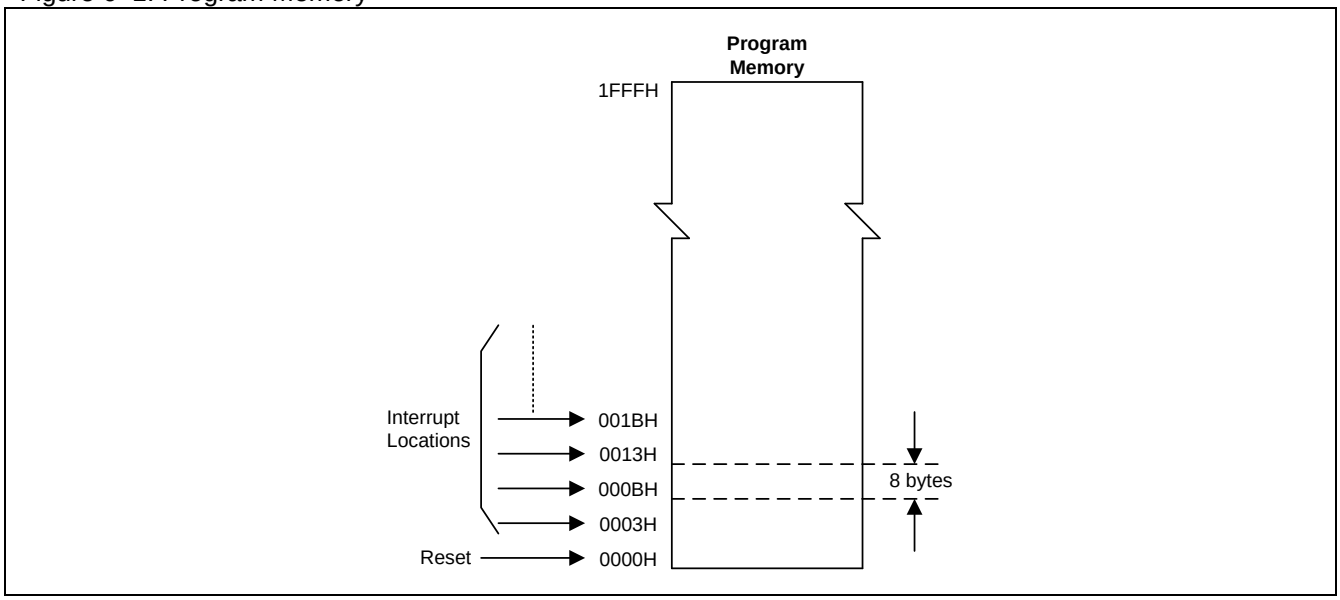
Data memory occupies a separate address space from program memory. In the **MPC82E/L52**, there is only a 256 bytes of internal scratch-pad RAM.

6.1. On-Chip Program Flash

Program memory is the memory which stores the program codes for the CPU to execute, as shown in Figure 6–1. After reset, the CPU begins execution from location 0000H, where should be the starting of the user's application code. To service the interrupts, the interrupt service locations (called interrupt vectors) should be located in the program memory. Each interrupt is assigned a fixed location in the program memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose program memory.

The interrupt service locations are spaced at an interval of 8 bytes: 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

Figure 6–1. Program Memory



6.2. On-Chip Data RAM

Figure 6–2 shows the internal data memory space available to the **MPC82E/L52** user. Internal data memory can be divided into three blocks, which are generally referred to as the lower 128 bytes of RAM, the upper 128 bytes of RAM, and the 128 bytes of SFR space. Internal data memory addresses are always 8-bit wide, which implies an address space of only 256 bytes. Direct addresses higher than 7FH access the SFR space; and indirect addresses higher than 7FH access the upper 128 bytes of RAM. Thus the SFR space and the upper 128 bytes of RAM occupy the same block of addresses, 80H through FFH, although they are physically separate entities.

The lower 128 bytes of RAM are present in all 80C51 devices as mapped in Figure 6–3. The lowest 32 bytes are grouped into 4 banks of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) select which register bank is in use. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing. The next 16 bytes above the register banks form a block of bit-addressable memory space. The 80C51 instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH.

All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing while the Upper 128 can only be accessed by indirect addressing.

Figure 6–4 gives a brief look at the Special Function Register (SFR) space. SFRs include the Port latches, timers, peripheral controls, etc. These registers can only be accessed by direct addressing. Sixteen addresses in SFR space are both byte- and bit-addressable. The bit-addressable SFRs are those whose address ends in 0H or 8H.

Figure 6–2. Data Memory

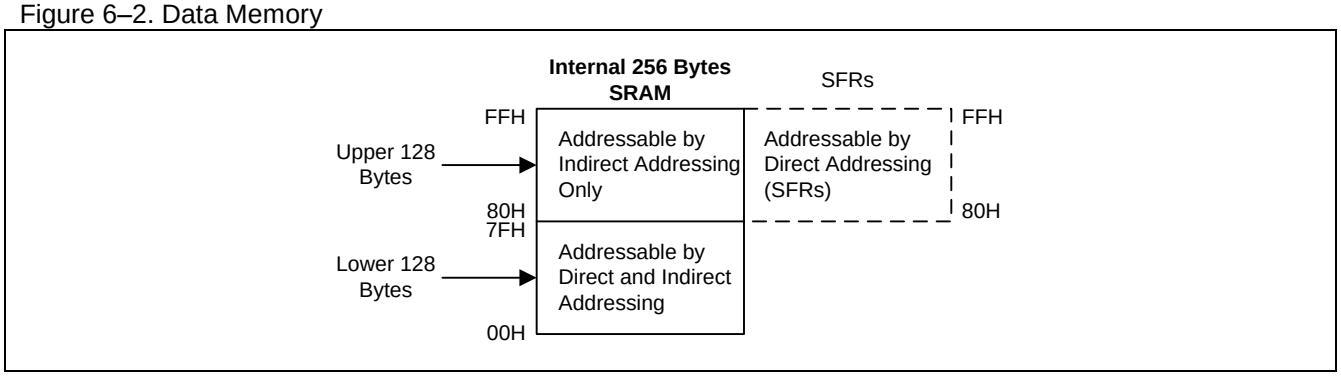


Figure 6–3. Lower 128 Bytes of Internal RAM

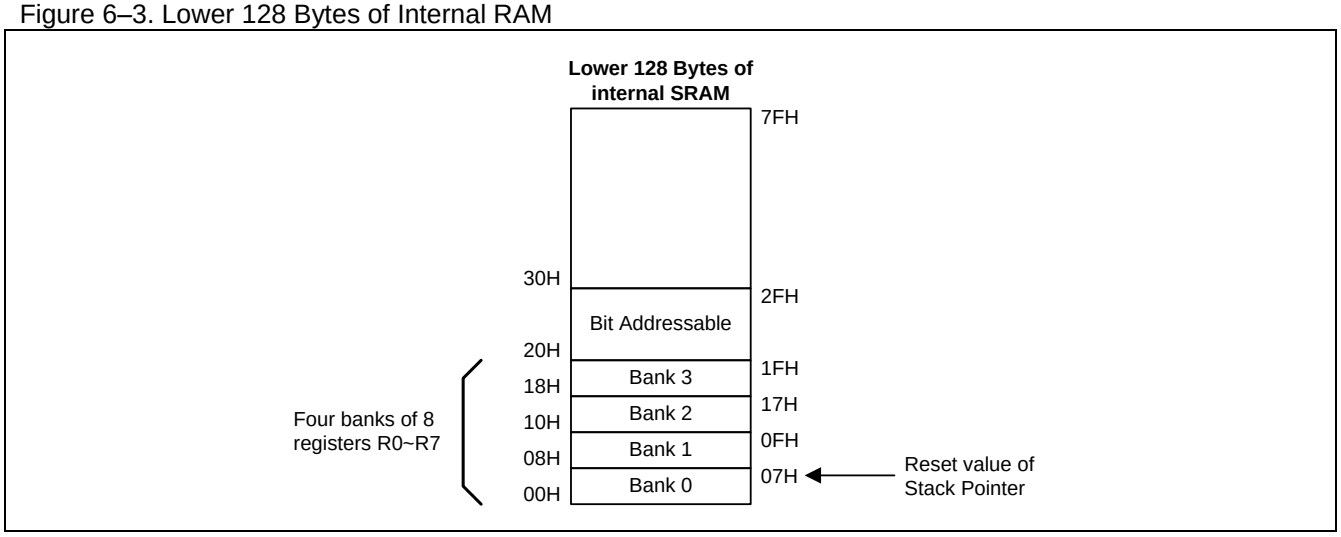
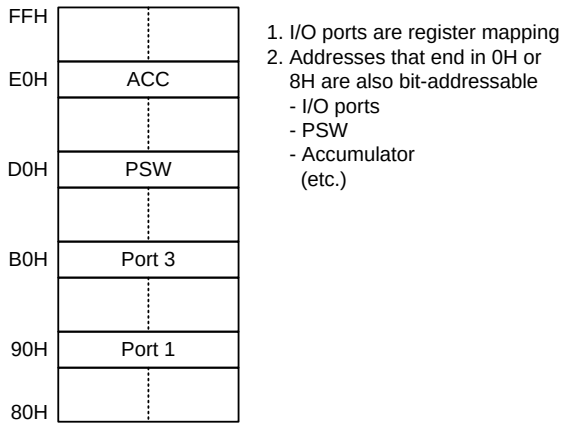


Figure 6–4. SFR Space



6.3. Declaration Identifiers in a C51-Compiler

The declaration identifiers in a C51-compiler for the various **MPC82E/L52** memory spaces are as follows:

data
128 bytes of internal data memory space (00h~7Fh); accessed via direct or indirect addressing, using instructions other than MOVX and MOVC. All or part of the Stack may be in this area.

idata
Indirect data; 256 bytes of internal data memory space (00h~FFh) accessed via indirect addressing using instructions other than MOVX and MOVC. All or part of the Stack may be in this area. This area includes the data area and the 128 bytes immediately above it.

sfr
Special Function Registers; CPU registers and peripheral control/status registers, accessible only via direct addressing.

xdata
There is no on-chip XRAM or XRAM interface for ***xdata*** access.

pdata
There is no on-chip XRAM or XRAM interface for ***pdata*** access.

code
8K bytes of program memory space; accessed as part of program execution and via the “MOVC @A+DTPR” instruction.

7. Data Pointer Register (DPTR)

There is only one set DPTR in **MPC82E/L52**. **MPC82E/L52** does not support external memory access.

8. System Clock

There are two clock sources for the system clock: external crystal oscillator and Internal RC Oscillator (IRCO). Figure 8–1 shows the structure of the system clock in **MPC82E/L52**.

The **MPC82E/L52** can boot from external crystal oscillator or IRCO on 6MHz which is selected by hardware option control bit, ENROSC. The ENROSC is enabled or disabled by general writer or Megawin proprietary writer.

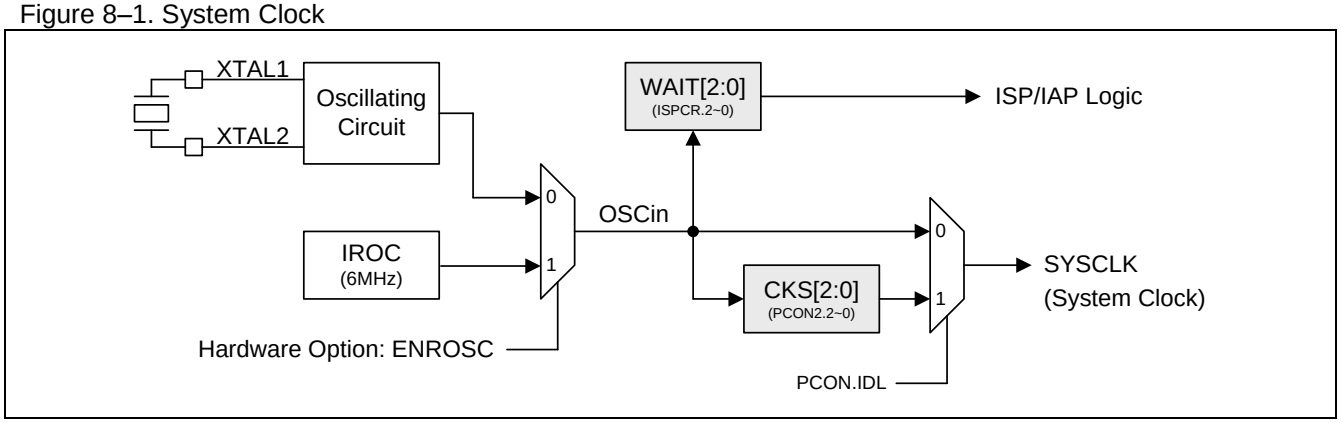
The built-in IRCO provides 6MHz frequency. To find the detailed IRCO performance, please refer Section "23.4 IRCO Characteristics").

The system clock in IDLE mode is obtained from one of these two clock sources through the clock divider, as shown in Figure 8–1. The user can program the divider control bits CKS2~CKS0 (in PCON2 register) to get the desired system clock. That is, the CKS2~CKS0 control bits are only active in **IDLE** mode.

8.1. Clock Structure

Figure 8–1 presents the principal clock systems in the **MPC82E/L52**. The system clock can be sourced by the external oscillator circuit or either internal oscillator.

Figure 8–1. System Clock



8.2. Clock Register

PCON2: Power Control Register 2

SFR Address = 0xC7 RESET = xxxx-x000

7	6	5	4	3	2	1	0
0	0	0	0	0	CKS2	CKS1	CKS0
W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7~3: Reserved. Software must write “0” on these bits when PCON2 is written.

Bit 2~0: CKS2 ~ CKS0, programmable System Clock Selection in **IDLE** mode. The default value of CKS[2:0] is set to “000” to select system clock on OSCin/1.

CKS[2:0]	System Clock in idle mode
0 0 0	OSCin/1
0 0 1	OSCin /2
0 1 0	OSCin /4
0 1 1	OSCin /8
1 0 0	OSCin /16
1 0 1	OSCin /32
1 1 0	OSCin /64
1 1 1	OSCin /128

8.3. Clock Sample Code

*(1). Required Function: Switch SYSCLK to OSCin/128 for **IDLE** mode (default is OSCin/1)*

Assembly Code Example:

```
CKS0                EQU         01h
CKS1                EQU         02h
CKS2                EQU         04h

   ORL     PCON2,#( CKS2 + CKS1 + CKS0)   ; Set CKS[2:0] = “111” to select OSCin/128
```

C Code Example:

```
#define  CKS0                           0x01
#define  CKS1                           0x02
#define  CKS2                           0x04

   PCON2 &= (CKS2 | CKS1 | CKS0);         // System clock divider /128
                                 // CKS[2:0], system clock divider
                                 //   0    | OSCin/1
                                 //   1    | OSCin/2
                                 //   2    | OSCin/4
                                 //   3    | OSCin/8
                                 //   4    | OSCin/16
                                 //   5    | OSCin/32
                                 //   6    | OSCin/64
                                 //   7    | OSCin/128
```

9. Watch Dog Timer (WDT)

9.1. WDT Structure

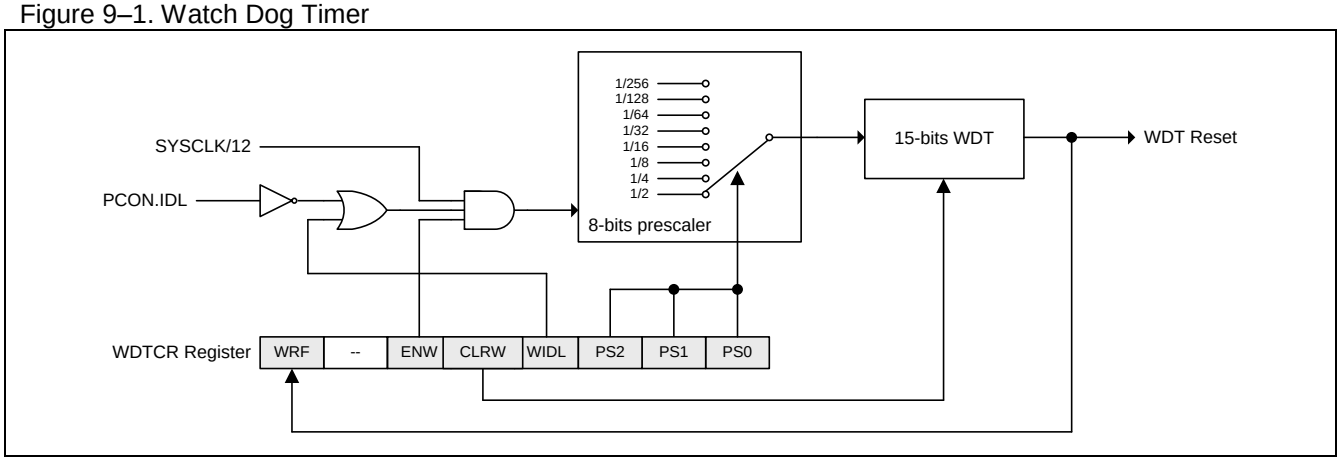
The Watch-dog Timer (WDT) is intended as a recovery method in situations where the CPU may be subjected to software upset. The WDT consists of a 15-bit free-running counter, a 8-bit prescaler and a control register (WDTCR). Figure 9–1 shows the WDT structure in **MPC82E/L52**.

When WDT is enabled, it derives its time base from the SYSCLK/12. The WDT overflow will trigger a system reset and set the WRF on WDTCR.7. To prevent WDT overflow, software needs to clear it by writing “1” to the CLRW bit (WDTCR.4) before WDT overflows.

Once the WDT is enabled by setting ENW bit, there is no way to disable it except through power-on reset. The WDTCR register will keep the previous programmed value unchanged after external reset (RST-pin), software reset and WDT reset.

ENW is implemented to one-time-enabled function, only writing “1” valid. Please refer Section “9.3 WDT Register” for more detail information.

Figure 9–1. Watch Dog Timer



9.2. WDT During Idle and Power Down

In the Idle mode, the WIDL bit (WDTCR.3) determines whether WDT counts or not. Set this bit to let WDT keep counting in the Idle mode. If the hardware option WDTRCO is enabled, the WDT always keeps counting regardless of WIDL bit.

In the Power-Down mode, WDT is frozen which is caused by stopped SYSCLK. After MCU wake-up, WDT continues the counting by SYSCLK resumed.

9.3. WDT Register

WDTCR: Watch-Dog-Timer Control Register

SFR Address = 0xE1 POR = 0000-0111 (xxx0_xxxx by Hardware Option)

7	6	5	4	3	2	1	0
WRF	--	ENW	CLRW	WIDL	PS2	PS1	PS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: WRF, WDT reset flag.
0: This bit should be cleared by software.
1: When WDT overflows, this bit is set by hardware to indicate a WDT reset happened.

Bit 6: Reserved. Software must write “0” on this bit when WDTCR is written.

Bit 5: ENW. Enable WDT.
0: Disable WDT running.
1: Enable WDT while it is set. Once ENW has been set, it can not be cleared by software.

Bit 4: CLRW. Clear WDT counter.
0: Writing “0” to this bit is no operation in WDT.
1: Writing “1” to this bit will clear the 15-bit WDT counter to 000H. Note this bit has no need to be cleared by writing “0”.

Bit 3: WIDL. WDT idle control.
0: WDT stops counting while the MCU is in idle mode.
1: WDT keeps counting while the MCU is in idle mode.

Bit 2~0: PS2 ~ PS0, select prescaler output for WDT time base input.

PS[2:0]	Prescaler Value	WDT period (if SYSCLK= 12MHz)
0 0 0	2	65.5ms
0 0 1	4	131ms
0 1 0	8	262ms
0 1 1	16	524ms
1 0 0	32	1.05S
1 0 1	64	2.10S
1 1 0	128	4.19S
1 1 1	256	8.39S

9.4. WDT Hardware Option

In addition to being initialized by software, the WDTCR register can also be automatically initialized at power-up by the hardware options HWENW, HWWIDL, HWPS[2:0] and WDSFWP, which should be programmed by a universal Writer or Programmer, as described below.

If HWENW is programmed to “enabled”, then hardware will automatically do the following initialization for the WDTCR register at power-up: (1) set ENW bit, (2) load HWWIDL into WIDL bit, and (3) load HWPS[2:0] into PS[2:0] bits.

If both of HWENW and WDSFWP are programmed to “enabled”, hardware still initializes the WDTCR register content by WDT hardware option at power-up. Then, any CPU writing on WDTCR bits will be inhibited except writing “1” on WDTCR.4 (CLRW), clear WDT.

HWENW: Hardware loaded for “ENW” of WDTCR.
☑: Enabled. Enable WDT and load the content of HWWIDL and HWPS2~0 to WDTCR after power-on.
☐: Disabled. WDT is not enabled automatically after power-on.

HWWIDL, HWPS2, HWPS1, HWPS0:
When HWENW is enabled, the content on these four fused bits will be loaded to WDTCR SFR after power-on.

WDSFWP:
☑: Enabled. The WDT SFRs, WREN, NSW, WIDL, PS2, PS1 and PS0 in WDTCR, will be write-protected.
☐: Disabled. The WDT SFRs, WREN, NSW, WIDL, PS2, PS1 and PS0 in WDTCR, are free for writing of software.

9.5. WDT Sample Code

(1) Required function: Enable WDT and select WDT prescalar to 1/32

Assembly Code Example:

```
PS0                 EQU         01h
PS1                 EQU         02h
PS2                 EQU         04h
WIDL                EQU         08h
CLRW                EQU         10h
ENW                 EQU         20h
WRF                 EQU         80h

   ANL    WDTCR,#(0FFh - WRF)          ; Clear WRF flag (write “0”)
   MOV    WDTCR,#(ENW + CLRW + PS2)    ; Enable WDT counter and set WDT prescaler to 1/32
```

C Code Example:

```
#define  PS0          0x01
#define  PS1          0x02
#define  PS2          0x04
#define  WIDL         0x08
#define  CLRW         0x10
#define  ENW          0x20
#define  WRF          0x80

   WDTCR &= ~WRF;                    // Clear WRF flag (write “0”)
   WDTCR = (ENW | CLRW | PS2);           // Enable WDT counter and set WDT prescaler to 1/32
                              // PS[2:0]  | WDT prescaler selection
                              //   0    | 1/2
                              //   1    | 1/4
                              //   2    | 1/8
                              //   3    | 1/16
                              //   4    | 1/32
                              //   5    | 1/64
                              //   6    | 1/128
                              //   7    | 1/256
```

10. System Reset

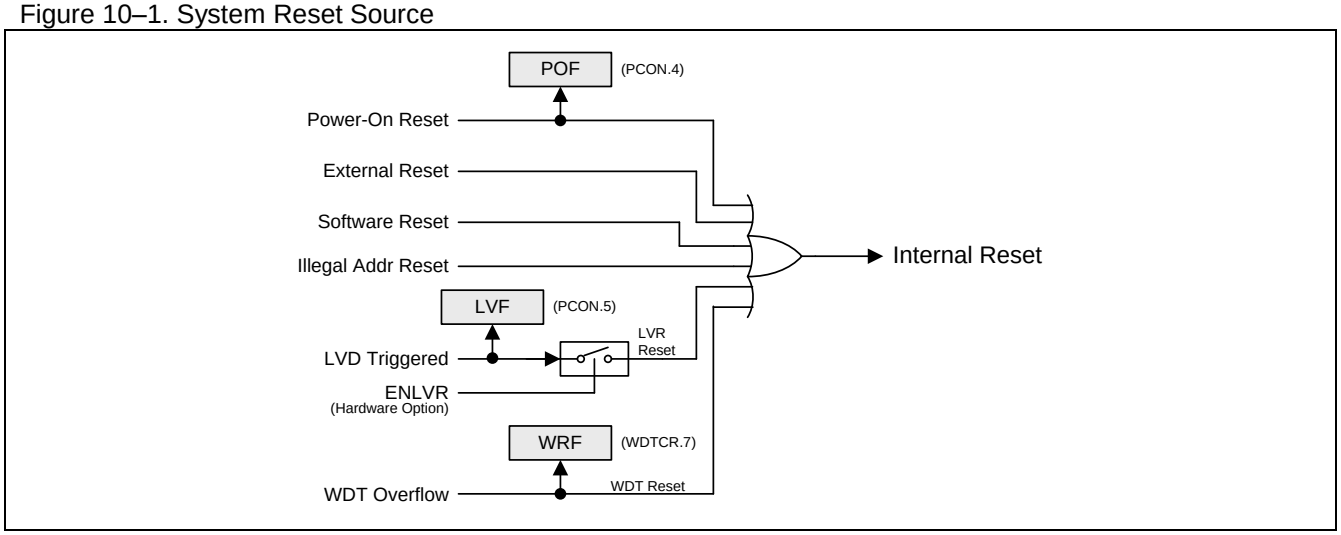
During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector, 0000H, or ISP start address by Hardware Option setting. The **MPC82E/L52** has six sources of reset: power-on reset, external reset, software reset, illegal address reset, WDT reset and low-voltage reset. Figure 10–1 shows the system reset source in **MPC82E/L52**.

The following sections describe the reset happened source and corresponding control registers and indicating flags.

10.1. Reset Source

Figure 10–1 presents the reset systems in the **MPC82E/L52** and all of its reset sources.

Figure 10–1. System Reset Source



10.2. Power-On Reset

Power-on reset (POR) is used to internally reset the CPU during power-up. The CPU will keep in reset state and will not start to work until the VDD power rises above the voltage of Power-On Reset. And, the reset state is activated again whenever the VDD power falls below the POR voltage. During a power cycle, VDD must fall below the POR voltage before power is reapplied in order to ensure a power-on reset

PCON: Power Control Register

SFR Address = 0x87 POR = 0011-0000, RESET = 000X-0000

7	6	5	4	3	2	1	0
SMOD	SMOD0	LVF	**POF**	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: POF. Power-On Flag.
0: The flag must be cleared by software to recognize next reset type.
1: Set by hardware when VDD rises from 0 to its nominal voltage. POF can also be set by software.

The Power-on Flag, POF, is set to “1” by hardware during power up or when VDD power drops below the POR voltage. It can be clear by firmware and is not affected by any warm reset such as external reset, Low-Voltage reset, software reset (ISPCR.5) and WDT reset. It helps users to check if the running of the CPU begins from power up or not. Note that the POF must be cleared by firmware.

10.3. External Reset

A reset is accomplished by holding the RESET pin HIGH for at least 24 oscillator periods while the oscillator is running. To ensure a reliable power-up reset, the hardware reset from RST pin is necessary.

10.4. Software Reset

Software can trigger the CPU to restart by software reset, writing “1” on SWRST (ISPCR.5). SWBS decides the CPU is boot from ISP or AP region after the reset action

ISPCR: ISP Control Register

SFR Address = 0xE5 RESET = 0000-x000

7	6	5	4	3	2	1	0
ISPEN	**SWBS**	**SWRST**	CFAIL	-	WAIT.2	WAIT.1	WAIT.0
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 6: SWBS, software boot selection control.
0: Boot from AP-memory after reset.
1: Boot from ISP memory after reset.

Bit 5: SWRST, software reset trigger control.
0: No operation
1: Generate software system reset. It will be cleared by hardware automatically.

10.5. Low-Voltage Reset

In **MPC82E/L52**, there is a Low-Voltage Detectors (LVD) to monitor VDD power. BOD0 services the fixed detection level at VDD=3.7V for E-series and 2.3V for L-series. If VDD power drops below LVD monitor level. Associated flag, LVF, is set when LVD meets the detection level. If ENLVR (hardware option) is enabled, the LVD will trigger a system reset and a set LVF indicates a LVD Reset occurred.

PCON: Power Control Register

SFR Address = 0x87 POR = 0011-0000, RESET = 000X-0000

7	6	5	4	3	2	1	0
SMOD	SMOD0	**LVF**	POF	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: LVF, Low-Voltage Flag.
0: The flag must be cleared by software after power-up for further detecting.
1: This bit is only set by hardware when VDD meets LVD monitored level.

10.6. WDT Reset

When WDT is enabled to start the counter, WRF will be set by WDT overflow and trigger a system reset that causes CPU to restart. Software can read the WRF to recognize the WDT reset occurred.

WDTCR: Watch-Dog-Timer Control Register

SFR Address = 0xE1 POR = 0000-0111 (xxx0_xxxx by Hardware Option)

7	6	5	4	3	2	1	0
WRF	--	ENW	CLRW	WIDL	PS2	PS1	PS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: WRF, WDT reset flag.
0: This bit should be cleared by software.
1: When WDT overflows, this bit is set by hardware to indicate a WDT reset happened.

10.7. Illegal Address Reset

In **MPC82E/L52**, if software program runs to illegal address such as over program ROM limitation, it triggers a RESET to CPU.

10.8. Reset Sample Code

(1) Required function: Trigger a software reset

Assembly Code Example:

```
SWRST       EQU             20h

   ORL   ISPCR,#SWRST        ; Trigger Software Reset
```

C Code Example:

```
#define  SWRST              0x20

   ISPCR |= SWRST;           // Trigger Software Reset
```

11. Power Management

The **MPC82E/L52** supports one power monitor module, Low-Voltage Detector (LVD), and **2** power-reducing modes: Idle mode and Power-down mode.

LVD reports the chip power status on the flag, LVF, which provides the capability to interrupt CPU by software configured or to reset CPU by hardware option. The three power-reducing modes provide the different power-saving scheme for chip application. These modes are accessed through the PCON and PCON2 register.

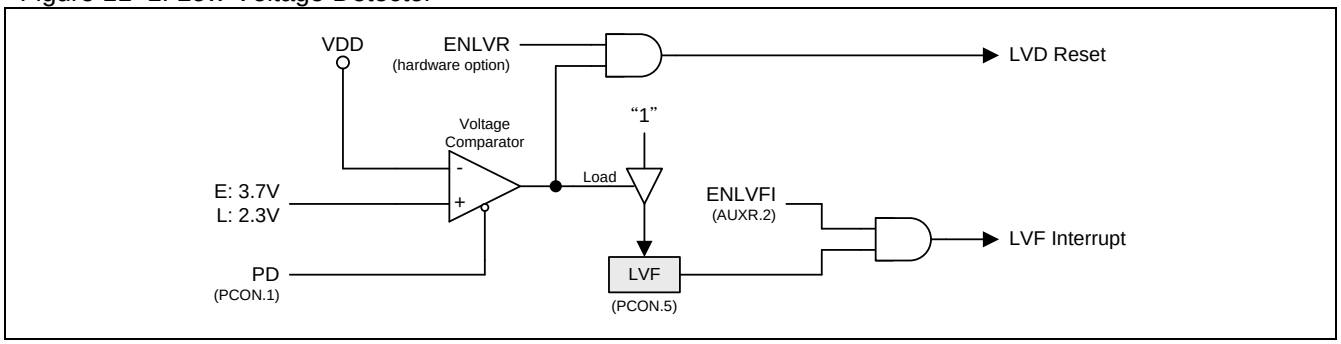
11.1. Low-Voltage Detector

In **MPC82E/L52**, there is a Low-Voltage Detector (LVD) to monitor VDD power. Figure 11–1 shows the functional diagram of LVD. LVD services the fixed detection level at VDD=**3.7V** for **5V** device and **2.3V** on **3.3V** device. Associated flag, LVF (PCON.5), is set when LVD meets the detection level. If ENLVFI (AUXR.2) is enabled, a set LVF will generate a low-voltage detection interrupt. It can interrupt CPU either CPU in normal mode or idle mode.

If ENLVR (hardware option) is enabled, the LVD event will trigger a system reset and a set LVF indicates a LVD Reset occurred. The LVD reset restart the CPU either CPU in normal mode or idle mode.

SPECIAL NOTE ON LOW-VOLTAGE-DETECTOR: The Low-Voltage-Detector is not a precise detector. The threshold voltage depends on temperature change. Lower the temperature goes, higher the threshold voltage rises. During temperature scope (-40℃, 85℃), the threshold voltage falls between scope (2.7V, 1.8V) for MPC82L52, and (4.2V, 3.2V) for MPC82E52. To control the low-voltage detection and reset, also the use must read another document "*Initial Configuration.pdf*" from Megawin which describes the initial option register settings.

Figure 11–1. Low-Voltage Detector



11.2. Power Saving Mode

11.2.1. Idle Mode

Setting the IDL bit in PCON enters idle mode. Idle mode halts the internal CPU clock. The CPU state is preserved in its entirety, including the RAM, stack pointer, program counter, program status word, and accumulator. The Port pins hold the logical states they had at the time that Idle was activated. Idle mode leaves the peripherals running in order to allow them to wake up the CPU when an interrupt is generated. Timer 0, Timer 1, UART, SPI and the LVD will continue to function during Idle mode. The PCA Timer and WDT are conditional enabled during Idle mode to wake up CPU. Any enabled interrupt source or reset may terminate Idle mode. When exiting Idle mode with an interrupt, the interrupt will immediately be serviced, and following RETI, the next instruction to be executed will be the one following the instruction that put the device into Idle.

It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

The alternative to save the Idle mode power is to slow the MCU's operating speed by programming CKS2~CKS0 bits (in PCON2 register, see Section "8 System Clock") to a non-0/0/0 value. The user should examine which program segments are suitable for lower operating speed. In principle, the lower operating speed should not affect the system's normal function. Then, restore its normal speed by hardware in the normal operating mode.

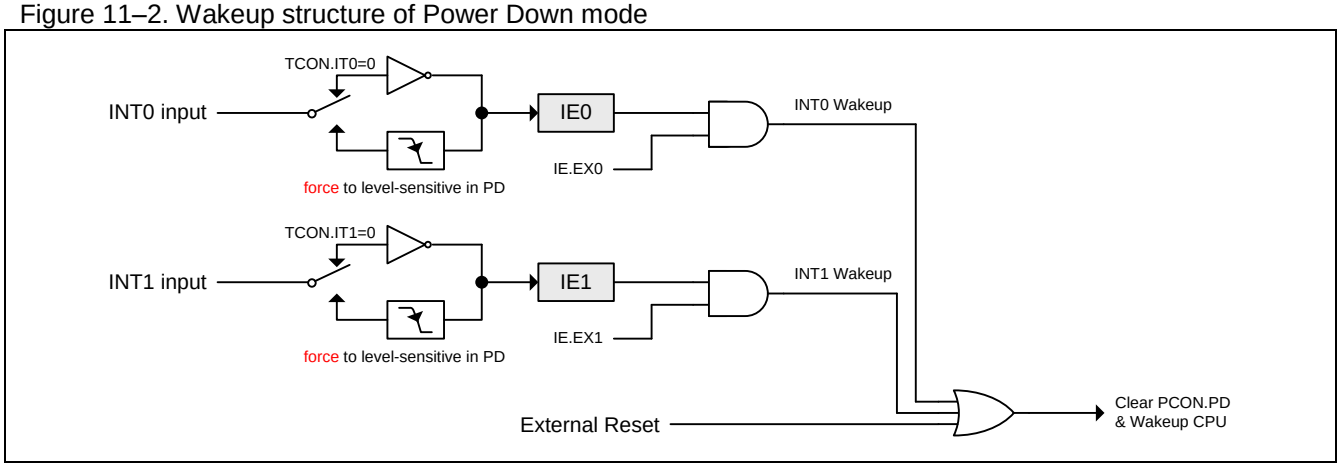
11.2.2. Power-Down Mode

Setting the PD bit in PCON enters Power-down mode. Power-down mode stops the oscillator and powers down the Flash memory in order to minimize power consumption. Only the power-on circuitry will continue to draw power during Power-down. During Power-down the power supply voltage may be reduced to the RAM keep-alive voltage. The RAM contents will be retained; however, the SFR contents are not guaranteed once VDD has been reduced. Power-down may be exited by external reset, power-on reset and enabled external interrupts.

The user should not attempt to enter (or re-enter) the power-down mode for a minimum of 4 µs until after one of the following conditions has occurred: Start of code execution (after any type of reset), or Exit from power-down mode.

Figure 11–2 shows the wakeup mechanism of power-down mode in **MPC82E/L52**.

Figure 11–2. Wakeup structure of Power Down mode



11.2.3. Interrupt Recovery from Power-down

Two external interrupts may be configured to terminate Power-down mode. External interrupts INT0 (P3.2), INT1 (P3.3) may be used to exit Power-down. To wake up by external interrupt INT0, INT1, the interrupt must be enabled and configured for level-sensitive operation. If the enabled external interrupts are configured to edge-sensitive operation (Falling), they will be forced to level-sensitive operation (Low level) by hardware in power-down mode.

When terminating Power-down by an interrupt, the wake up period is internally timed. At the falling edge on the interrupt pin, Power-down is exited, the oscillator is restarted, and an internal timer begins counting. The internal clock will not be allowed to propagate and the CPU will not resume execution until after the timer has reached internal counter full. After the timeout period, the interrupt service routine will begin. To prevent the interrupt from re-triggering, the ISR should disable the interrupt before returning. The interrupt pin should be held low until the device has timed out and begun executing.

11.2.4. Reset Recovery from Power-down

For RST input pin, wakeup from Power-down through an external reset is similar to the interrupt. At the rising edge of RST, Power-down is exited, the oscillator is restarted, and an internal timer begins counting. The internal clock will not be allowed to propagate to the CPU until after the timer has reached internal counter full. The RST pin must be held high for longer than the timeout period to ensure that the device is reset properly. The device will begin executing once RST is brought low.

11.3. Power Control Register

PCON: Power Control Register

SFR Address = 0x87 POR = 0011-0000, RESET = 000x-0000

7	6	5	4	3	2	1	0
SMOD	SMOD0	LVF	POF	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1: PD, Power-Down control bit.
0: This bit could be cleared by CPU or any exited power-down event.
1: Setting this bit activates power down operation.

Bit 0: IDL, Idle mode control bit.
0: This bit could be cleared by CPU or any exited Idle mode event.
1: Setting this bit activates idle mode operation.

PCON2: Power Control Register 2

SFR Address = 0xC7 RESET = xxxx-x000

7	6	5	4	3	2	1	0
0	0	0	0	0	CKS2	CKS1	CKS0
W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7~3: Reserved. Software must write “0” on these bits when PCON2 is written.

Bit 2~0: CKS2 ~ CKS0, programmable System Clock Selection in **IDLE** mode. The default value of CKS[2:0] is set to “000” to select system clock on OSCin/1.

CKS[2:0]	System Clock in idle mode
0 0 0	OSCin/1
0 0 1	OSCin /2
0 1 0	OSCin /4
0 1 1	OSCin /8
1 0 0	OSCin /16
1 0 1	OSCin /32
1 1 0	OSCin /64
1 1 1	OSCin /128

11.4. Power Control Sample Code

*(1) Required function: Switch SYSCLK to OSCin/128 for **IDLE** mode (default is OSCin/1)*

Assembly Code Example:

```
CKS0                EQU         01h
CKS1                EQU         02h
CKS2                EQU         04h

   ORL     PCON2,#( CKS2 + CKS1 + CKS0)   ; Set CKS[2:0] = “111” to select OSCin/128
```

C Code Example:

```
#define  CKS0                           0x01
#define  CKS1                           0x02
#define  CKS2                           0x04

   PCON2 |= (CKS2 | CKS1 | CKS0);         // System clock divider /128
                                 // CKS[2:0], system clock divider
                                 //   0    | OSCin/1
                                 //   1    | OSCin/2
                                 //   2    | OSCin/4
                                 //   3    | OSCin/8
                                 //   4    | OSCin/16
                                 //   5    | OSCin/32
                                 //   6    | OSCin/64
                                 //   7    | OSCin/128
```

12. Configurable I/O Ports

The **MPC82E/L52** has following I/O ports: P1.0~P1.7, P3.0~P3.5 and P3.7. The exact number of I/O pins available depends upon the package types. See Table 12–1.

Table 12–1. Number of I/O Pins Available

Package Type	I/O Pins	Number of I/O ports
20-pin PDIP/SOP/TSSOP	P1.0~P1.7, P3.0~P3.5, P3.7	15

12.1. IO Structure

The I/O operating modes in **MPC82E/L52** support four configurations on I/O operating. These are: quasi-bidirectional (standard 8051 I/O port), push-pull output, input-only (high-impedance input) and open-drain output.

Followings describe the configuration of the all types I/O mode.

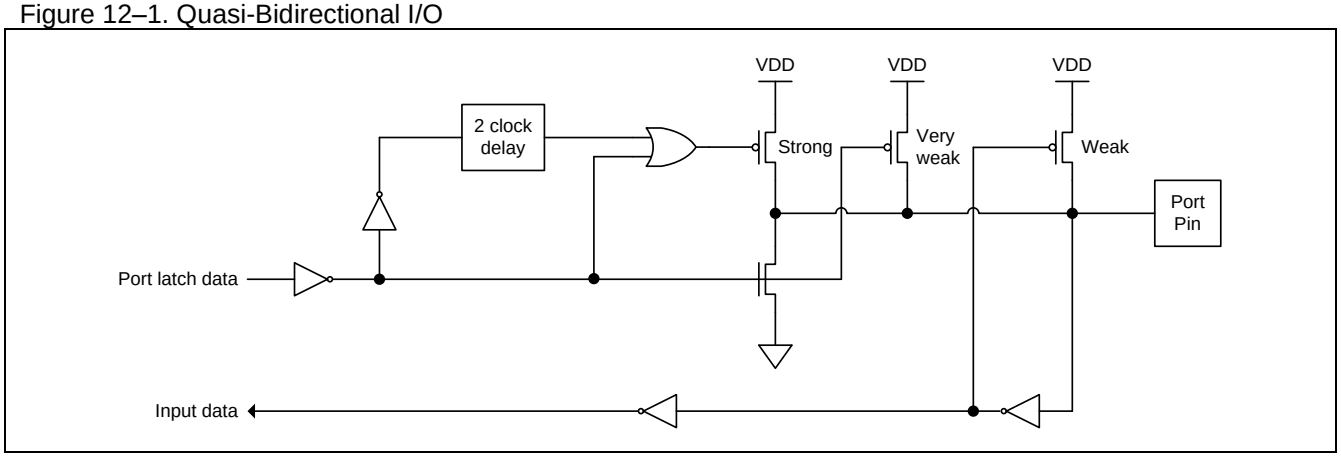
12.1.1. Quasi-Bidirectional IO Structure

All port pins in quasi-bidirectional mode are similar to the standard 8051 port pins. A quasi-bidirectional port can be used as an input and output without the need to reconfigure the port. This is possible because when the port outputs a logic high, it is weakly driven, allowing an external device to pull the pin low. When the pin outputs low, it is driven strongly and able to sink a large current. There are three pull-up transistors in the quasi-bidirectional output that serve different purposes.

One of these pull-ups, called the “very weak” pull-up, is turned on whenever the port register for the pin contains a logic “1”. This very weak pull-up sources a very small current that will pull the pin high if it is left floating. A second pull-up, called the “weak” pull-up, is turned on when the port register for the pin contains a logic “1” and the pin itself is also at a logic “1” level. This pull-up provides the primary source current for a quasi-bidirectional pin that is outputting a 1. If this pin is pulled low by the external device, this weak pull-up turns off, and only the very weak pull-up remains on. In order to pull the pin low under these conditions, the external device has to sink enough current to over-power the weak pull-up and pull the port pin below its input threshold voltage. The third pull-up is referred to as the “strong” pull-up. This pull-up is used to speed up low-to-high transitions on a quasi-bidirectional port pin when the port register changes from a logic “0” to a logic “1”. When this occurs, the strong pull-up turns on for **two** CPU clocks, quickly pulling the port pin high.

The quasi-bidirectional port configuration is shown in Figure 12–1.

Figure 12–1. Quasi-Bidirectional I/O

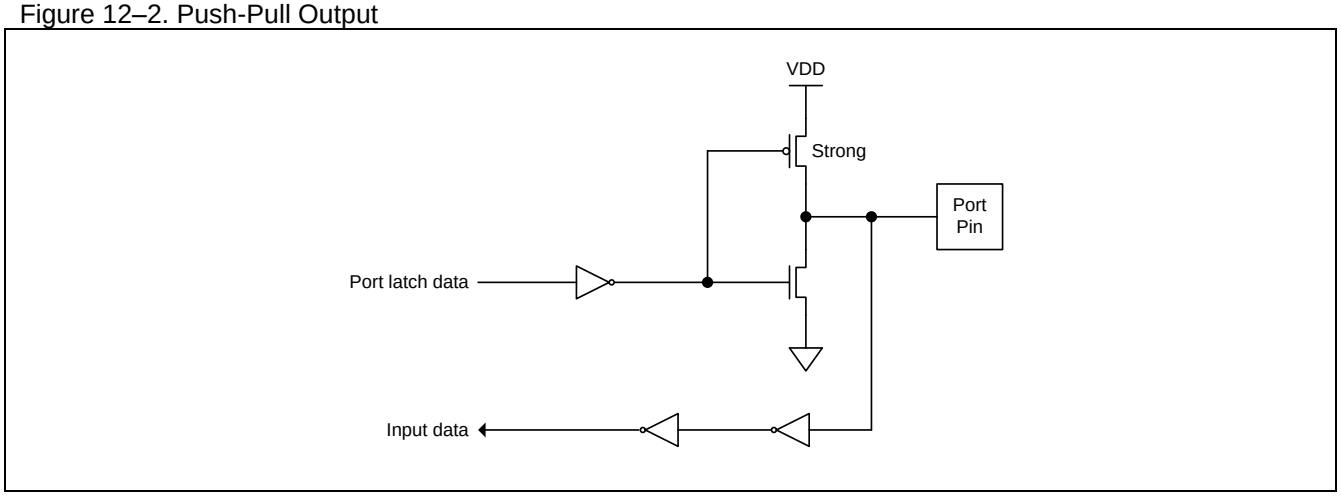


12.1.2. Push-Pull Output Structure

The push-pull output configuration has the same pull-down structure as both the open-drain and the quasi-bidirectional output modes, but provides a continuous strong pull-up when the port register contains a logic “1”. The push-pull mode may be used when more source current is needed from a port output. In addition, the input path of the port pin in this configuration is also the same as quasi-bidirectional mode.

The push-pull port configuration is shown in Figure 12–2.

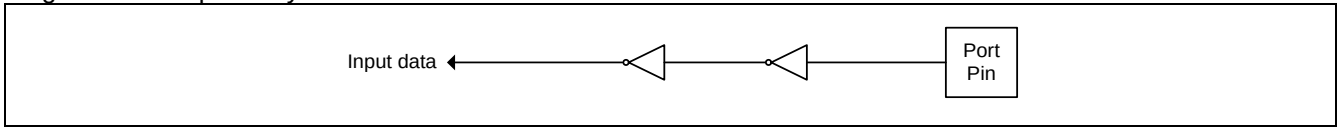
Figure 12–2. Push-Pull Output



12.1.3. Input-Only (High Impedance Input) Structure

The input-only configuration is an input without any pull-up resistors on the pin, as shown in Figure 12–3.

Figure 12–3. Input-Only

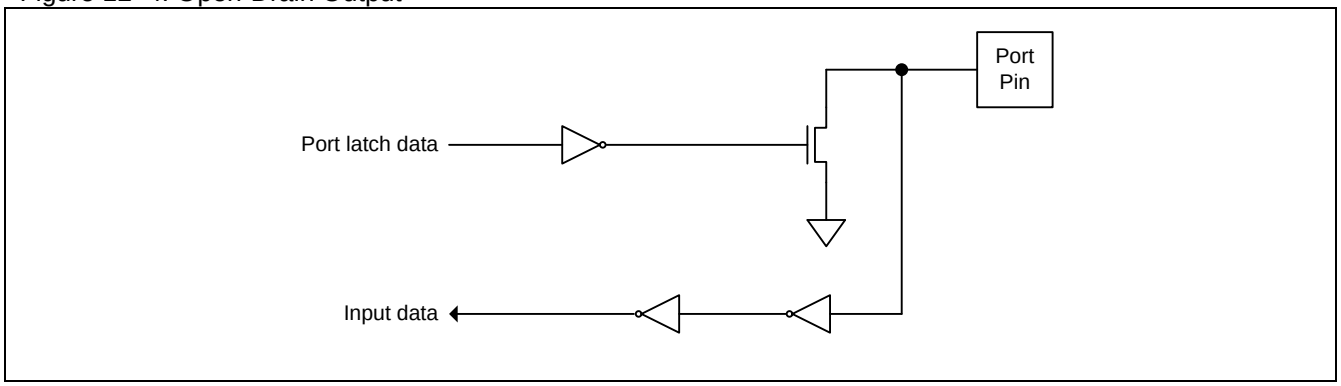


12.1.4. 3 Open-Drain Output Structure

The open-drain output configuration turns off all pull-ups and only drives the pull-down transistor of the port pin when the port register contains a logic “0”. To use this configuration in application, a port pin must have an external pull-up, typically a resistor tied to VDD. The pull-down for this mode is the same as for the quasi-bidirectional mode. In addition, the input path of the port pin in this configuration is also the same as quasi-bidirectional mode.

The open-drain port configuration is shown in Figure 12–4.

Figure 12–4. Open-Drain Output



12.2. I/O Port Register

All I/O port pins on the **MPC82E/L52** may be individually and independently configured by software to one of four types on a bit-by-bit basis, as shown in Table 12–2. Two mode registers select the output type for each I/O pin.

Table 12–2. I/O Port Configuration Settings

PxM0.y	PxM1.y	Port Mode
0	0	Quasi-Bidirectional (default)
0	1	Push-Pull Output
1	0	Input Only (High Impedance Input)
1	1	Open-Drain Output

Where x=1 or 3 (port number), and y=0~7 (port pin). The registers PxM0 and PxM1 are listed in each port description.

12.2.1. Port 1 Register

P1: Port 1 Register

SFR Address = 0x90 RESET = 1111-1111

7	6	5	4	3	2	1	0
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: P1.7~P1.0 could be only set/cleared by CPU.

P1M0: Port 1 Mode Register 0

SFR Address = 0x91 POR+RESET = 0000-0000

7	6	5	4	3	2	1	0
P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

P1M1: Port 1 Mode Register 1

SFR Address = 0x92 POR+RESET = 0000-0000

7	6	5	4	3	2	1	0
P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

12.2.2. Port 3 Register

P3: Port 3 Register

SFR Address = 0xB0 RESET = 1x11-1111

7	6	5	4	3	2	1	0
P3.7	--	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P3.7 could be only set/cleared by CPU.

Bit 6: Reserved. Software must write “1” on this bit when P3 is written.

Bit 5~0: P3.7~P3.0 could be only set/cleared by CPU.

P3M0: Port 3 Mode Register 0

SFR Address = 0xB1 RESET = 0x00-0000

7	6	5	4	3	2	1	0
P3M0.7	--	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 5: Reserved. Software must write “0” on this bit when P3M0 is written.

P3M1: Port 3 Mode Register 1

SFR Address = 0xB2 RESET = 0000-0000

7	6	5	4	3	2	1	0
P3M1.7	--	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 5: Reserved. Software must write “0” on this bit when P3M1 is written.

12.3. GPIO Sample Code

(1). Required Function: Set P1.0 to input-only mode

Assembly Code Example:

```
P1Mn0           EQU             01h

   ORL    P1M0, #P1Mn0
   ANL    P1M1, #(0FFh + P1Mn0)      ; Configure P1.0 to input only mode
   SETB   P1.0                       ; Set P1.0 data latch to “1” to enable input mode
```

C Code Example:

```
#define  P1Mn0          0x01

   P1M0 |= P1Mn0;
   P1M1 &= ~P1Mn0;                   // Configure P1.0 to input only mode
   P10 = 1;                          // Set P1.0 data latch to “1” to enable input mode
```

(2). Required Function: Set P1.0 to push-pull output mode

Assembly Code Example:

```
P1Mn0           EQU             01h

   ANL    P1M0, #(0FFh - P1Mn0)
   ORL    P1M1, #P1Mn0            ; Configure P1.0 to push pull mode
   SETB   P1.0
```

C Code Example:

```
#define  P1Mn0          0x01

   P1M0 &= ~P1Mn0;
   P1M1 |= P1Mn0;                    // Configure P1.0 to push pull mode
   P10 = 1;                          // Set P1.0 data latch to “1” to enable push pull mode
```

(3). Required Function: Set P1.0 to open-drain output mode

Assembly Code Example:

```
P1Mn0           EQU             01h

   ORL    P1M0, #P1Mn0
   ORL    P1M1, #P1Mn0             ; Configure P1.0 to open drain mode
   SETB   P1.0                     ; Set P1.0 data latch to “1” to enable open drain mode
```

C Code Example:

```
#define  P1Mn0          0x01

   P1M0 |= P1Mn0;
   P1M1 |= P1Mn0;                    // Configure P1.0 to open drain mode
   P10 = 1;                          // Set P1.0 data latch to “1” to enable open drain mode
```

13. Interrupt

The **MPC82E/L52** has **7** interrupt sources with a four-level interrupt structure. There are several SFRs associated with the four-level interrupt. They are the IE, IP, IPH, and AUXR. The IPH (Interrupt Priority High) register makes the four-level interrupt structure possible. The four priority level interrupt structure allows great flexibility in handling these interrupt sources.

13.1. Interrupt Structure

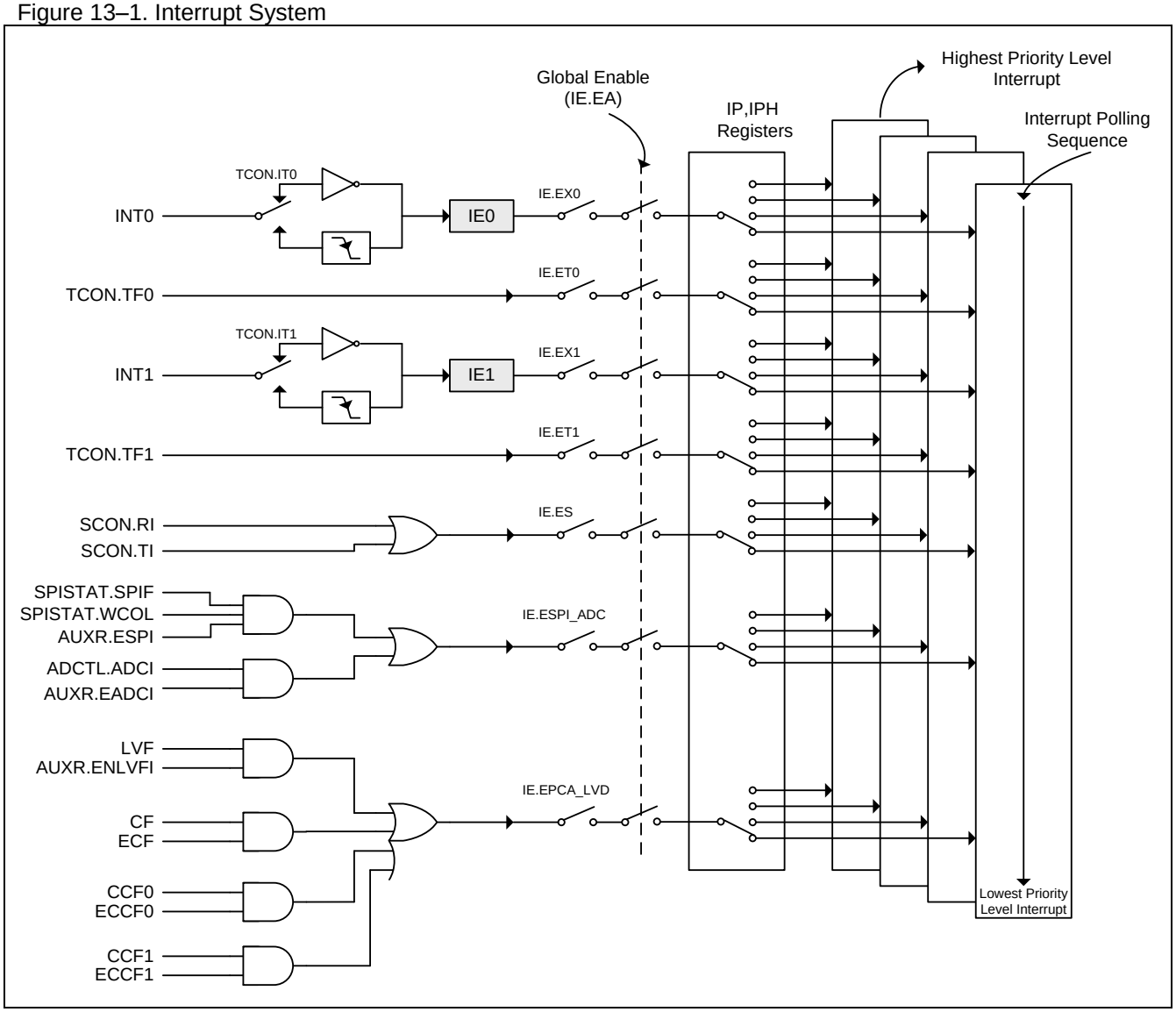
Table 13–1 lists all the interrupt sources. The 'Request Bits' are the interrupt flags that will generate an interrupt if it is enabled by setting the 'Enable Bit'. Of course, the global enable bit EA (in IE register) should have been set previously. The 'Request Bits' can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be cancelled in software. The 'Priority Bits' determine the priority level for each interrupt. The 'Priority within Level' is the polling sequence used to resolve simultaneous requests of the same priority level. The 'Vector Address' is the entry point of an interrupt service routine in the program memory.

Figure 13–1 shows the interrupt system. Each of these interrupts will be briefly described in the following sections.

Table 13–1. Interrupt Sources

No	Source Name	Enable Bit	Request Bits	Priority Bits	Polling Priority	Vector Address
#1	External Interrupt 0, INT0	EX0	IE0	[PX0H, PX0]	(Highest)	0003H
#2	Timer 0	ET0	TF0	[PT0H, PT0]	…	000Bh
#3	External Interrupt 1, INT1	EX1	IE1	[PX1H, PX1]	...	0013H
#4	Timer 1	ET1	TF1	[PT1H, PT1]	…	001BH
#5	Serial Port	ES	RI, TI	[PSH, PS]	…	0023H
#6	SPI/ADC	ESPI_ADC & (ESPI, EADCI)	SPIF, WCOL, ADCI	[PSPIH_ADC, PSPI_ADC]	…	002BH
#7	PCA/LVF	EPCA_LVD & (ECF, ECCFn, ENLVFI)	CF CCFn n=0~1) LVF	[PPCAH_LVD, PPCA_LVD]	(Lowest)	0033H

Figure 13–1. Interrupt System



13.2. Interrupt Source

Table 13–2. Interrupt flags

No	Source Name	Request Bits	Bit Location
#1	External Interrupt, INT0	IE0	TCON.1
#2	Timer 0	TF0	TCON.5
#3	External Interrupt, INT1	IE1	TCON.3
#4	Timer 1	TF1	TCON.7
#5	Serial Port	RI TI	SCON.0 SCON.1
#6	SPI/ADC	SPIF, WCOL, ADCI	SPISTAT.7~6 ADCTL.4
#7	PCA/LVF	CF, CCFn (n=0~1), LVF	CCON.7 CCON.1~0 PCON.5

The external interrupt INT0 and INT1 can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to *only if the interrupt was transition –activated*, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

The Timer0 and Timer1 interrupts are generated by TF0 and TF1, which are set by a rollover in their respective Timer/Counter registers in most cases. When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The serial port interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll RI and TI to determine which one to request service and it will be cleared by software.

The SPI/ADC interrupt is shared by the logical OR of SPI interrupt (SPIF and WCOL) and ADC interrupt (ADCI). Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll them to determine which one to request service and it will be cleared by software.

The PCA/LVF interrupt is shared by the logical OR of PCA interrupt (CF, CCF0 and CCF1) and LVD (Low-Voltage Detector) interrupt (LVF on PCON.5). Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll them to determine which one to request service and it will be cleared by software.

13.3. Interrupt Enable

Table 13–3. Interrupt enable control

No	Source Name	Request Bits	Enable Bit	Bit Location
#1	External Interrupt, INT0	IE0	EX0	IE.0
#2	Timer 0	TF0	ET0	IE.1
#3	External Interrupt, INT1	IE1	EX1	IE.2
#4	Timer 1	TF1	ET1	IE.3
#5	Serial Port	RI TI	ES	IE.4
#6	SPI/ADC	SPIF, WCOL ADCI	ESPI_ADC & (ESPI, EADCI)	IE.5, AUXR.3, AUXR.4
#7	PCA/LVF	CF, CCF0, CCF1, LVF	EPCA_LVD & (ECF, ECCF0, ECCF1, ENLVFI)	IE.6, CMOD.0, CCAPM0.0, CCAPM1.0, AUXR.2

There are **7** interrupt sources available in **MPC82E/L52**. Each of these interrupt sources can be individually enabled or disabled by setting or clearing an interrupt enable bit in the register IE. IE also contains a global disable bit, EA, which can be cleared to disable all interrupts at once. If EA is set to '1', the interrupts are individually enabled or disabled by their corresponding enable bits. If EA is cleared to '0', all interrupts are disabled.

13.4. Interrupt Priority

The priority scheme for servicing the interrupts is the same as that for the 80C51, except there are four interrupt levels rather than two as on the 80C51. The Priority Bits (see Table 13–1) determine the priority level of each interrupt. IP and IPH are combined to 4-level priority interrupt. Table 13–4 shows the bit values and priority levels associated with each combination.

Table 13–4. Interrupt priority level

{IPH.x , IP.x}	Priority Level
11	1 (highest)
10	2
01	3
00	4

Each interrupt source has two corresponding bits to represent its priority. One is located in SFR named IPH and the other in IP register. Higher-priority interrupt will be not interrupted by lower-priority interrupt request. If two interrupt requests of different priority levels are received simultaneously, the request of higher priority is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determine which request is serviced. Table 13–2 shows the internal polling sequence in the same priority level and the interrupt vector address.

13.5. Interrupt Process

Each interrupt flag is sampled at every system clock cycle. The samples are polled during the next system clock. If one of the flags was in a set condition at first cycle, the second cycle(polling cycle) will find it and the interrupt system will generate an hardware LCALL to the appropriate service routine as long as it is not blocked by any of the following conditions.

Block conditions:

- An interrupt of equal or higher priority level is already in progress.
- The current cycle (polling cycle) is not the final cycle in the execution of the instruction in progress.
- The instruction in progress is RETI or any write to the IE, IP and IPH registers.

Any of these three conditions will block the generation of the hardware LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed before vectoring into any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE, IP or IPH, then at least one or more instruction will be executed before any interrupt is vectored to.

13.6. Interrupt Register

TCON: Timer/Counter Control Register

SFR Address = 0x88 RESET = 0000-0000

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: IE1, Interrupt 1 Edge flag.
0: Cleared when interrupt processed on if transition-activated.
1: Set by hardware when external interrupt 1 edge is detected (transmitted or level-activated).

Bit 2: IT1: Interrupt 1 Type control bit.
0: Cleared by software to specify low level triggered external interrupt 1.
1: Set by software to specify falling edge triggered external interrupt 1.

Bit 1: IE0, Interrupt 0 Edge flag.
0: Cleared when interrupt processed on if transition-activated.
1: Set by hardware when external interrupt 0 edge is detected (transmitted or level-activated).

Bit 0: IT0: Interrupt 0 Type control bit.
0: Cleared by software to specify low level triggered external interrupt 0.
1: Set by software to specify falling edge triggered external interrupt 0.

IE: Interrupt Enable Register

SFR Address = 0xE8 RESET = 0000-0000

7	6	5	4	3	2	1	0
EA	EPCA_LVD	ESPI_ADC	ES	ET1	EX1	ET0	EX0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: EA, All interrupts enable register.
0: Global disables all interrupts.
1: Global enables all interrupts.

Bit 6: EPCA_LVD, group interrupt enable register of PCA and Low-Voltage Detector (LVD).
0: Disable group interrupt of PCA and LVD.
1: Enable group interrupt of PCA and LVD.

Bit 5: ESPI_ADC, group interrupt enable register of SPI and ADC.
0: Disable group interrupt SPI and ADC.
1: Enable group interrupt SPI and ADC.

Bit 4: ES, Serial port interrupt enable register.
0: Disable serial port interrupt.
1: Enable serial port interrupt.

Bit 3: ET1, Timer 1 interrupt enable register.
0: Disable Timer 1 interrupt.
1: Enable Timer 1 interrupt.

Bit 2: EX1, External interrupt 1 enable register.
0: Disable external interrupt 1.
1: Enable external interrupt 1.

Bit 1: ET0, Timer 0 interrupt enable register.
0: Disable Timer 0 interrupt.
1: Enable Timer 1 interrupt.

Bit 0: EX0, External interrupt 0 enable register.
0: Disable external interrupt 0.
1: Enable external interrupt 1.

AUXR: Auxiliary Register

SFR Address = 0x8E RESET = 0000-00xx

7	6	5	4	3	2	1	0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	--	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 4: EADCI, ADC interrupt enable register.
0: Disable ADC interrupt.
1: Enable ADC interrupt.

Bit 3: ESPI, Enable SPI interrupt.
0: Disable SPI interrupt.
1: Enable SPI interrupt.

Bit 2: ENLVFI, Enable LVD Interrupt.
0: Disable LVD (LVF) interrupt.
1: Enable LVD (LVF) interrupt.

IP: Interrupt Priority low Register

SFR Address = 0xB8 RESET = xx00-0000

7	6	5	4	3	2	1	0
--	PPCA_LVD	PSPI_ADC	PS	PT1	PX1	PT0	PX0
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: Reserved. Software must write “0” on this bit when IP is written.
Bit 6: PPCA_LVD, interrupt priority-L register of PCA and LVD.
Bit 5: PSPI_ADC, interrupt priority-L register of SPI and ADC.
Bit 4: PS, Serial port interrupt priority-L register.
Bit 3: PT1, Timer 1 interrupt priority-L register.
Bit 2: PX1, external interrupt 1 priority-L register.
Bit 1: PT0, Timer 0 interrupt priority-L register.
Bit 0: PX0, external interrupt 0 priority-L register.

IPH: Interrupt Priority High Register

SFR Address = 0xB7 RESET = xx00-0000

7	6	5	4	3	2	1	0
--	PPCAH_LVD	PSPIH_ADC	PSH	PT1H	PX1H	PT0H	PX0H
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: Reserved. Software must write “0” on this bit when IPH is written.
Bit 6: PPCAH_LVD, interrupt priority-H register of PCA and LVD.
Bit 5: PSPIH_ADC, interrupt priority-H register of SPI and ADC.
Bit 4: PSH, Serial port interrupt priority-H register.
Bit 3: PT1H, Timer 1 interrupt priority-H register.
Bit 2: PX1H, external interrupt 1 priority-H register.
Bit 1: PT0H, Timer 0 interrupt priority-H register.
Bit 0: PX0H, external interrupt 0 priority-H register.

13.7. Interrupt Sample Code

(1). Required Function: Set INT0 wake-up MCU in power-down mode

Assembly Code Example:

```
PX0                 EQU         01h
PX0H                EQU         01h
PD                  EQU         02h

   ORG    0000h
   JMP    main

   ORG    00003h
ext_int0_isr:
   to do.....
   RETI

main:

   SETB    P3.2                ;

   ORL     IP,#PX0             ; Select INT0 interrupt priority
   ORL     IPH,#PX0H           ;

   JB      P3.2, $             ; Confirm P3.2 input low?????

   SETB    EX0                 ; Enable INT0 interrupt
   CLR     IE0                 ; Clear INT0 flag
   SETB    EA                  ; Enable global interrupt

   ORL     PCON,#PD            ; Set MCU into Power Down mode

   JMP     $
```

C Code Example:

```
#define  PX0                    0x01
#define  PX0H                   0x01
#define  PD                     0x02

void ext_int0_isr(void) interrupt 0
{
   To do……
}

void main(void)
{
   P32 = 1;

   IP |= PX0;                   // Select INT0 interrupt priority
   IPH |= PX0H;

   while(P32);                  // Confirm P3.2 input low??????

   EX0 = 1;                     // Enable INT0 interrupt
   IE0 = 0;                     // Clear INT0 flag
   EA = 1;                      // Enable global interrupt

   PCON |= PD;                  // Set MCU into Power Down mode

   while(1);
}
```

14. Timers/Counters

MPC82E/L52 has two Timers/Counters: Timer 0 and Timer 1. Timer0/1 can be configured as timers or event counters.

In the “timer” function, the timer rate is prescaled by 12 clock cycle to increment register value. In other words, it is to count the standard C51 machine cycle. AUXR.T0X12 and AUXR.T1X12 are the function for Timer 0/1 to set the timer rate on every clock cycle.

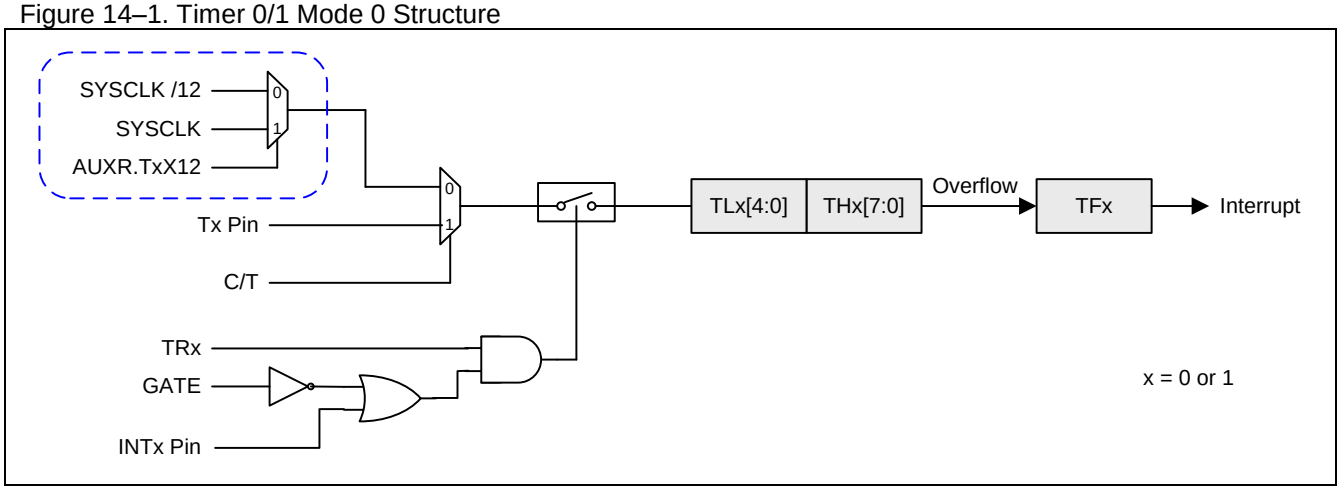
In the “counter” function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0 or T1. In this function, the external input is sampled by every timer rate cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register at the end of the cycle following the one in which the transition was detected.

14.1. Timer0 and Timer1

14.1.1. Mode 0 Structure

The timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TFx. The counted input is enabled to the timer when TRx = 1 and either GATE=0 or INTx = 1. Mode 0 operation is the same for Timer0 and Timer1. The mode 0 function of Timer 0/1 is shown in Figure 14–1.

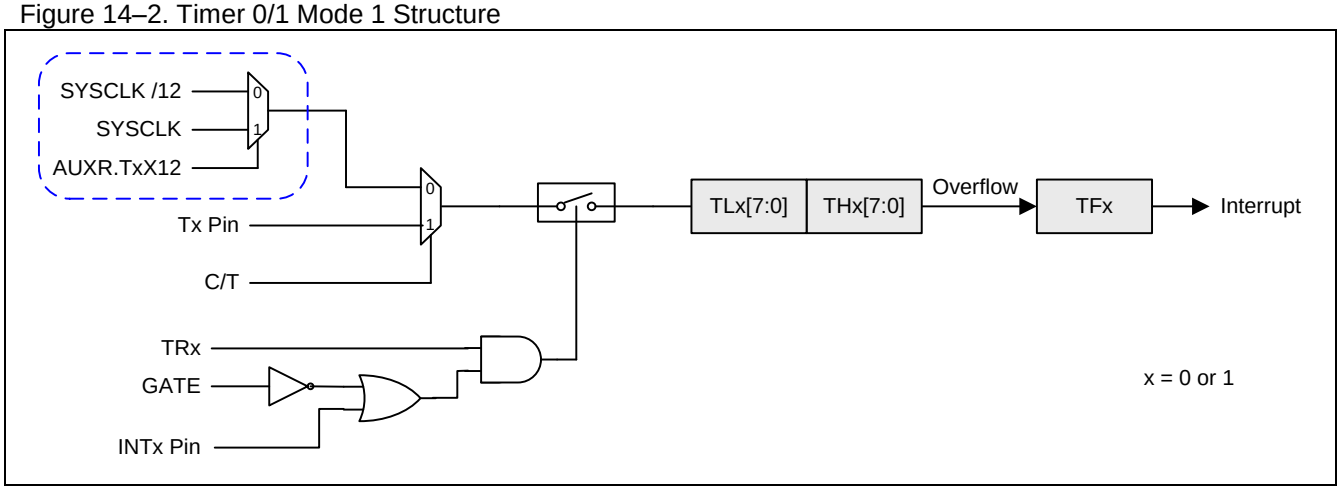
Figure 14–1. Timer 0/1 Mode 0 Structure



14.1.2. Mode 1 Structure

Timer 0/1 in Mode1 is configured as a 16 bit timer or counter. The function of GATE, INTx and TRx is same as mode 0. Figure 14–2 shows the mode 1 structure of Timer 0 and Timer 1.

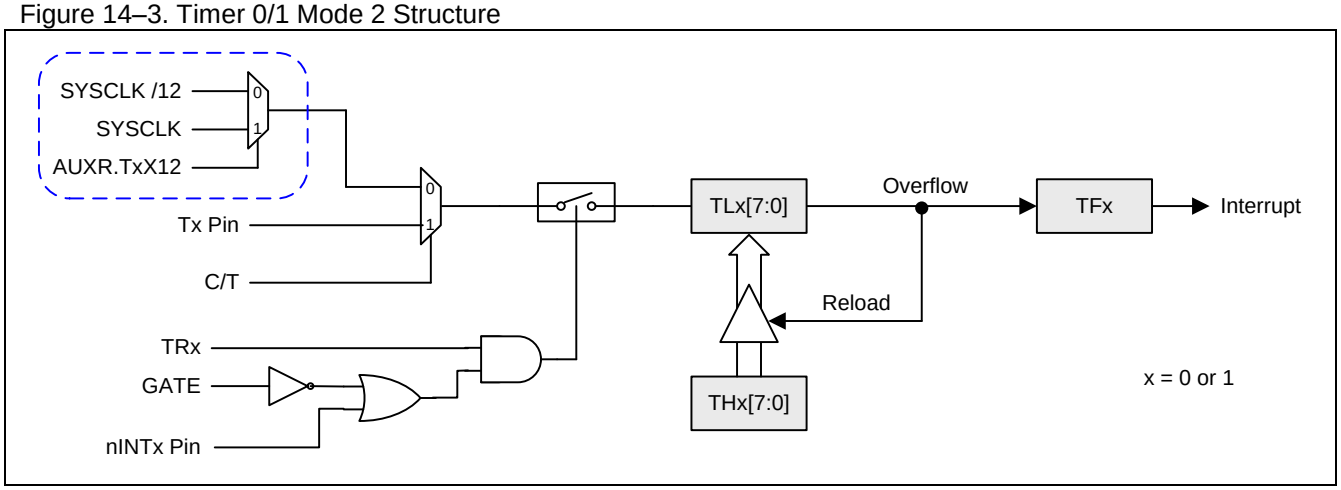
Figure 14–2. Timer 0/1 Mode 1 Structure



14.1.3. Mode 2 Structure

Mode 2 configures the timer register as an 8-bit counter(TLx) with automatic reload. Overflow from TLx not only set TFx, but also reload TLx with the content of THx, which is determined by software. The reload leaves THx unchanged. Mode 2 operation is the same for Timer0 and Timer1.

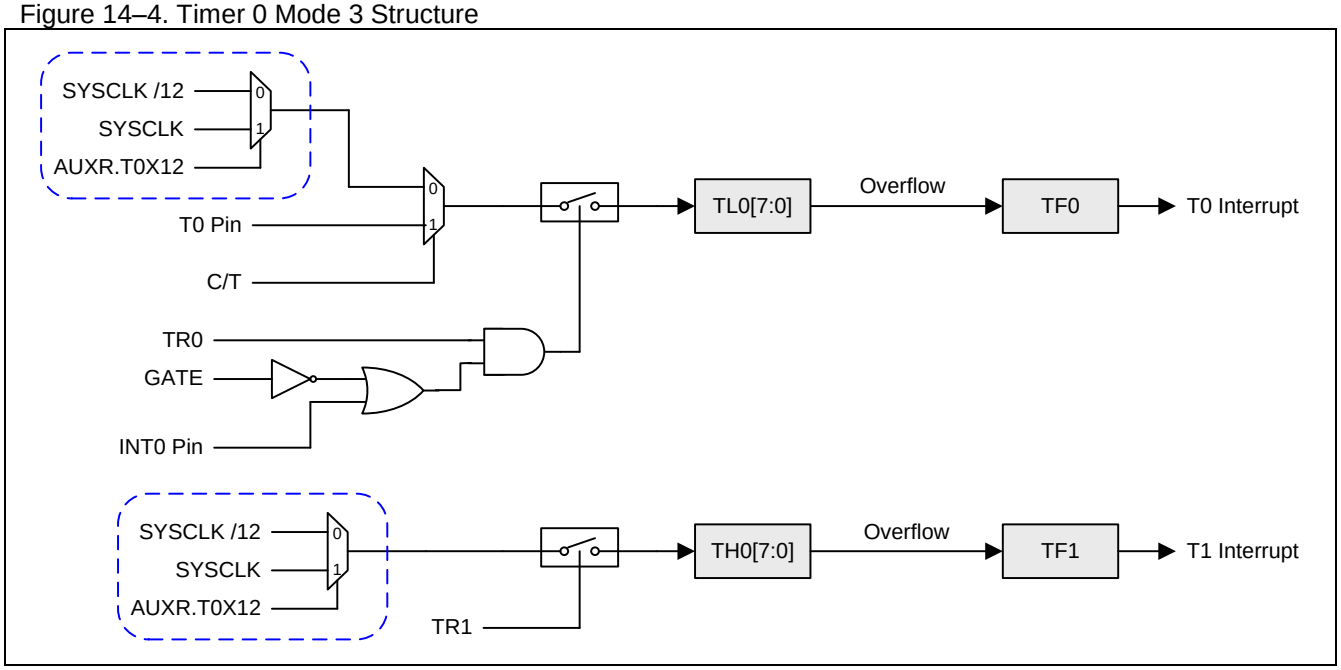
Figure 14–3. Timer 0/1 Mode 2 Structure



14.1.4. Mode 3 Structure

Timer1 in Mode3 simply holds its count, the effect is the same as setting TR1 = 1. Timer0 in Mode 3 enables TL0 and TH0 as two separate 8-bit counters. TL0 uses the Timer0 control bits such like C/T, GATE, TR0, INT0 and TF0. TH0 is locked into a timer function (can not be external event counter) and take over the use of TR1, TF1 from Timer1. TH0 now controls the Timer1 interrupt.

Figure 14–4. Timer 0 Mode 3 Structure



14.1.5. Timer0/1 Register

TCON: Timer/Counter Control Register

SFR Address = 0x88 RESET = 0000-0000

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: TF1, Timer 1 overflow flag.
0: Cleared by hardware when the processor vectors to the interrupt routine, or cleared by software.
1: Set by hardware on Timer/Counter 1 overflow, or set by software.

Bit 6: TR1, Timer 1 Run control bit.
0: Cleared by software to turn Timer/Counter 1 off.
1: Set by software to turn Timer/Counter 1 on.

Bit 5: TF0, Timer 0 overflow flag.
0: Cleared by hardware when the processor vectors to the interrupt routine, or cleared by software.
1: Set by hardware on Timer/Counter 0 overflow, or set by software.

Bit 4: TR0, Timer 0 Run control bit.
0: Cleared by software to turn Timer/Counter 0 off.
1: Set by software to turn Timer/Counter 0 on.

TMOD: Timer/Counter Mode Control Register

SFR Address = 0x89 RESET = 0000-0000

7	6	5	4	3	2	1	0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**
|←---------- Timer1 ----------→|				|←---------- Timer0 ----------→|		

Bit 7/3: Gate, Gating control for Timer1/0.
0: Disable gating control for Timer1/0.
1: Enable gating control for Timer1/0. When set, Timer1/0 or Counter1/0 is enabled only when /INT1 or /INT0 pin is high and TR1 or TR0 control bit is set.

Bit 6/2: C/T, Timer for Counter function selector.
0: Clear for Timer operation, input from internal system clock.
1: Set for Counter operation, input form T1 input pin.

Bit 5~4/1~0: Operating mode selection.

M1	**M0**	**Operating Mode**
0	0	13-bit timer/counter for Timer0 and Timer1
0	1	16-bit timer/counter for Timer0 and Timer1
1	0	8-bit timer/counter with automatic reload for Timer0 and Timer1
1	1 (Timer0)	TL0 is 8-bit timer/counter, TH0 is locked into 8-bit timer
1	1 (Timer1)	Timer/Counter1 Stopped

TL0: Timer Low 0 Register

SFR Address = 0x8A RESET = 0000-0000

7	6	5	4	3	2	1	0
TL0.7	TL0.6	TL0.5	TL0.4	TL0.3	TL02	TL0.1	TL0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

TH0: Timer High 0 Register

SFR Address = 0x8C RESET = 0000-0000

7	6	5	4	3	2	1	0
TH0.7	TH0.6	TH0.5	TH0.4	TH0.3	TH0.2	TH0.1	TH0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

TL1: Timer Low 1 Register

SFR Address = 0x8B RESET = 0000-0000

7	6	5	4	3	2	1	0
TL1.7	TL1.6	TL1.5	TL1.4	TL1.3	TL1.2	TL1.1	TL1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

TH1: Timer High 1 Register

SFR Address = 0x8D RESET = 0000-0000

7	6	5	4	3	2	1	0
TH1.7	TH1.6	TH1.5	TH1.4	TH1.3	TH1.2	TH1.1	TH1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

AUXR: Auxiliary Register

SFR Address = 0x8E RESET = 0000-00xx

7	6	5	4	3	2	1	0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	--	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 7: T0X12, Timer 0 clock source selector while C/T=0.
0: Clear to select SYSCLK/12.
1: Set to select SYSCLK as the clock source.

Bit 6: T1X12, Timer 1 clock source selector while C/T=0.
0: Clear to select SYSCLK/12.
1: Set to select SYSCLK as the clock source.

14.1.6. Timer0/1 Sample Code

(1). Required Function: IDLE mode with T0 wake-up frequency 10KHz, SYSCLK = 12MHz Crystal

Assembly Code Example:

```
T0M0            EQU             01h
T0M1            EQU             02h
PT0             EQU             02h
PT0H            EQU             02h
IDL             EQU             01h

   ORG   0000h
   JMP   main

   ORG   0000Bh
time0_isr:
   to do…
   RETI

main:                          ; (unsigned short value)

   MOV    TH0,#(256-100)           ; Set Timer 0 overflow rate = SYSCLK x 100
   MOV    TL0,#(256-100)           ;
   ANL    TMOD,#0F0h               ; Set Timer 0 to Mode 2
   ORL    TMOD,#T0M1               ;
   CLR    TF0                      ; Clear Timer 0 Flag

   ORL    IP,#PT0                  ; Select Timer 0 interrupt priority
   ORL    IPH,#PT0H                ;

   SETB   ET0                      ; Enable Timer 0 interrupt
   SETB   EA                       ; Enable global interrupt

   SETB   TR0                      ; Start Timer 0 running

   ORL    PCON,#IDL                ; Set MCU into IDLE mode
   JMP    $
```

C Code Example:

```
#define  T0M0                   0x01
#define  T0M1                   0x02
#define  PT0                    0x02
#define  PT0H                   0x02
#define  IDL                    0x01

void time0_isr(void) interrupt 1
{
   To do…
}

void main(void)
{
   TH0 = TL0 = (256-100);           // Set Timer 0 overflow rate = SYSCLK x 100
   TMOD &= 0xF0;                    // Set Timer 0 to Mode 2
   TMOD |= T0M1;
   TF0 = 0;                         // Clear Timer 0 Flag

   IP |= PT0;                       // Select Timer 0 interrupt priority
   IPH |= PT0H;

   ET0 = 1;                         // Enable Timer 0 interrupt
   EA = 1;                          // Enable global interrupt

   TR0 = 1;                         // Start Timer 0 running

   PCON =IDL;                       // Set MCU into IDLE mode
   while(1);
}
```

(2). Required Function: Select Timer 0 clock source from SYSCLK (enable T0X12)

Assembly Code Example:

```
T0M0            EQU             01h
T0M1            EQU             02h
PT0             EQU             02h
PT0H            EQU             02h
T0X12           EQU             80h

   ORG   0000h
   JMP   main

   ORG   0000Bh
time0_isr:
   to do…
   RETI

main:
   ORL    AUXR, #T0X12            ; Select SYSCLK/1 for Timer 0 clock input
   CLR    TF0                     ; Clear Timer 0 Flag

   ORL    IP,#PT0                 ; Select Timer 0 interrupt priority
   ORL    IPH,#PT0H               ;

   SETB   ET0                     ; Enable Timer 0 interrupt
   SETB   EA                      ; Enable global interrupt

   MOV    TH0, #(256 - 240)       ;interrupt interval 20us
   MOV    TL0, #(256 - 240)       ;

   ANL    TMOD,#0F0h              ; Set Timer 0 to Mode 2
   ORL    TMOD,#T0M1              ;


   SETB   TR0                     ; Start Timer 0 running
   JMP    $
```

C Code Example:

```
#define  T0M0                    0x01
#define  T0M1                    0x02
#define  PT0                     0x02
#define  PT0H                    0x02
#define  T0X12                   0x80

   AUXR |= T0X12
   TF0 = 0;

   IP |= PT0;                    // Select Timer 0 interrupt priority
   IPH |= PT0H;

   ET0 = 1;                      // Enable Timer 0 interrupt
   EA = 1;                       // Enable global interrupt

   TH0 = TL0 = (256 - 240);

   TMOD &= 0xF0;                 // Set Timer 0 to Mode 2
   TMOD |= T0M1;

   TR0 = 1;                      // Start Timer 0 running
```

15. Serial Port (UART)

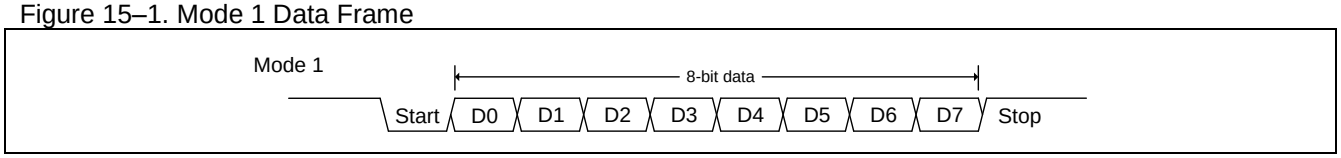
The serial port of **MPC82E/L52** support full-duplex transmission, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the register. However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost. The serial port receive and transmit registers are both accessed at special function register SBUF. Writing to SBUF loads the transmit register, and reading from SBUF accesses a physically separate receive register.

The serial port can operate in 4 modes: Mode 0 provides *synchronous* communication while Modes 1, 2, and 3 provide *asynchronous* communication. The asynchronous communication operates as a full-duplex Universal Asynchronous Receiver and Transmitter (UART), which can transmit and receive simultaneously and at different baud rates.

Mode 0: 8 data bits (LSB first) are transmitted or received through RXD(P3.0). TXD(P3.1) always outputs the shift clock. The baud rate can be selected to 1/12 or **1/2** the system clock frequency by URM0X6 setting in AUXR register.

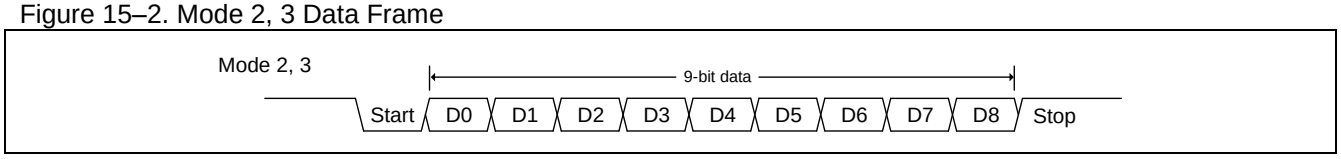
Mode 1: 10 bits are transmitted through TXD or received through RXD. The frame data includes a start bit (0), 8 data bits (LSB first), and a stop bit (1), as shown in Figure 15–1. On receive, the stop bit would be loaded into RB8 in SCON register. The baud rate is variable.

Figure 15–1. Mode 1 Data Frame



Mode 2: 11 bits are transmitted through TXD or received through RXD. The frame data includes a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1), as shown in Figure 15–2. On Transmit, the 9th data bit comes from TB8 in SCON register can be assigned the value of 0 or 1. On receive, the 9th data bit would be loaded into RB8 in SCON register, while the stop bit is ignored. The baud rate can be configured to 1/32 or 1/64 the system clock frequency.

Figure 15–2. Mode 2, 3 Data Frame



Mode 3: Mode 3 is the same as Mode 2 except the baud rate is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. In Mode 0, reception is initiated by the condition RI=0 and REN=1. In the other modes, reception is initiated by the incoming start bit with 1-to-0 transition if REN=1.

In addition to the standard operation, the UART can perform framing error detection by looking for missing stop bits, and automatic address recognition.

15.1. Serial Port Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The shift clock source can be selected to 1/12 or **1/2** the system clock frequency by URM0X6 setting in AUXR register. Figure 15–3 shows a simplified functional diagram of the serial port in Mode 0.

Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal triggers the UART engine to start the transmission. The data in the SBUF would be shifted into the RXD(P3.0) pin by each raising edge shift clock on the TXD(P3.1) pin. After eight raising edge of shift clocks passing, TI would be asserted by hardware to indicate the end of transmission. Figure 15–4 shows the transmission waveform in Mode 0.

Reception is initiated by the condition REN=1 and RI=0. At the next instruction cycle, the Serial Port Controller writes the bits 11111110 to the receive shift register, and in the next clock phase activates Receive.

Receive enables Shift Clock which directly comes from RX Clock to the alternate output function of P3.1 pin. When Receive is active, the contents on the RXD(P3.0) pin would be sampled and shifted into shift register by falling edge of shift clock. After eight falling edge of shift clock, RI would be asserted by hardware to indicate the end of reception. Figure 15–5 shows the reception waveform in Mode 0.

Figure 15–3. Serial Port Mode 0

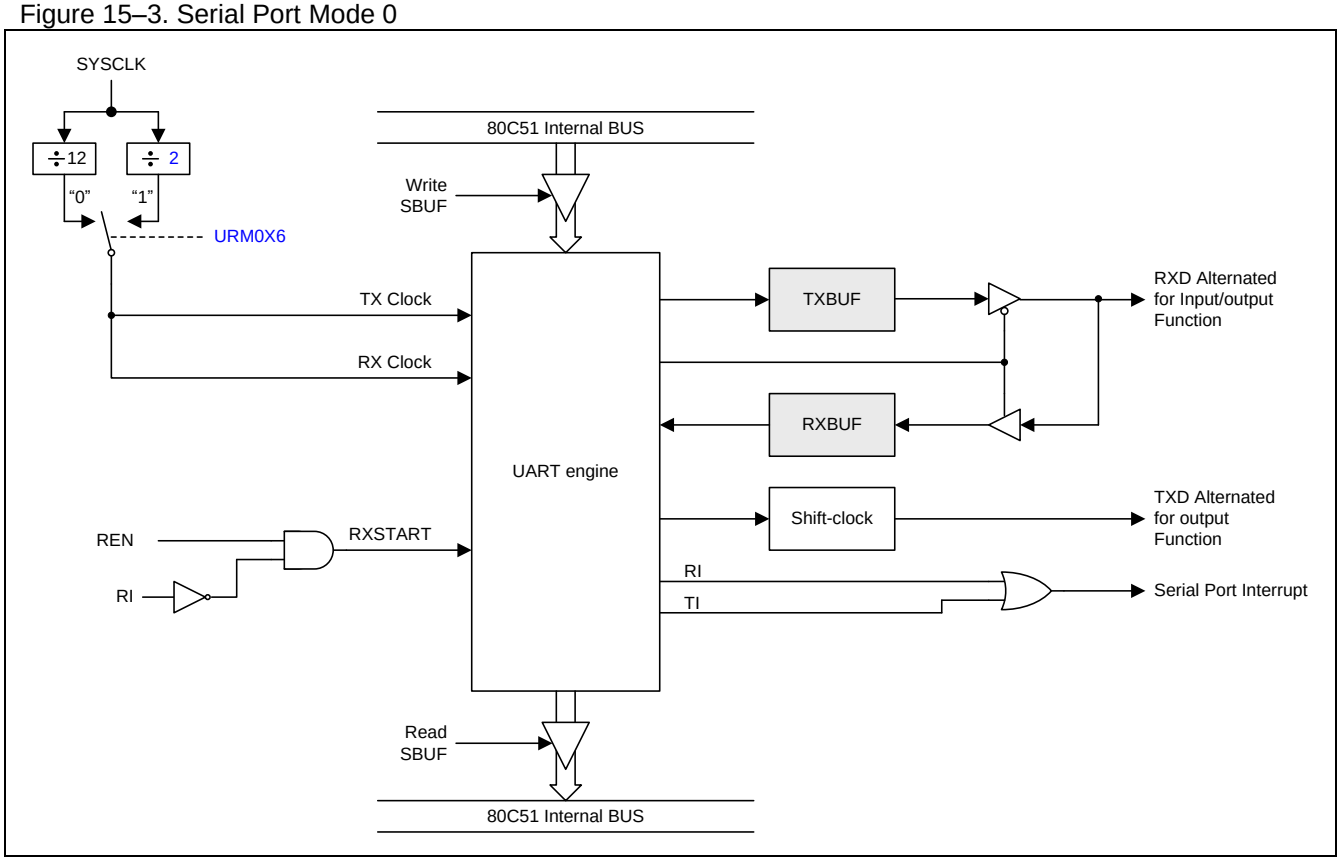


Figure 15–4. Mode 0 Transmission Waveform

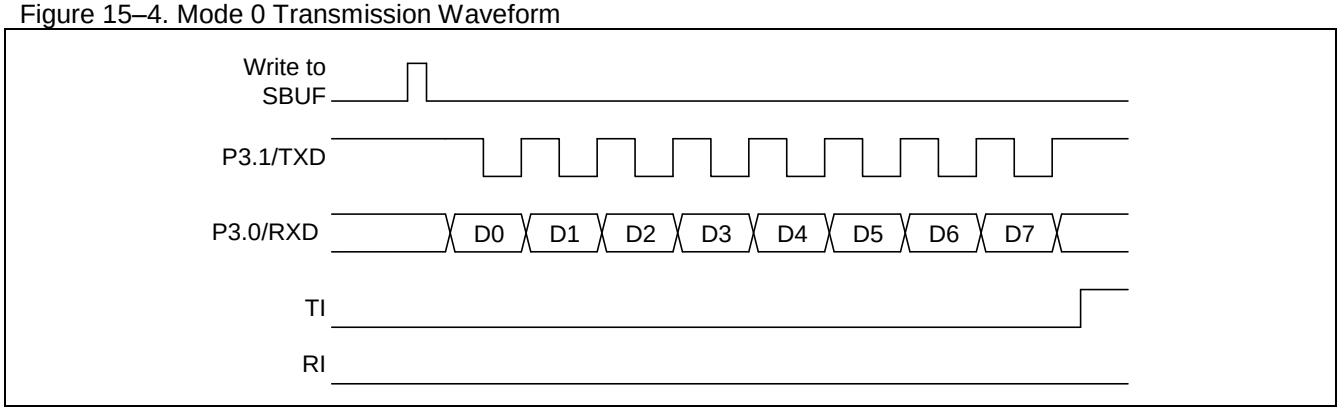
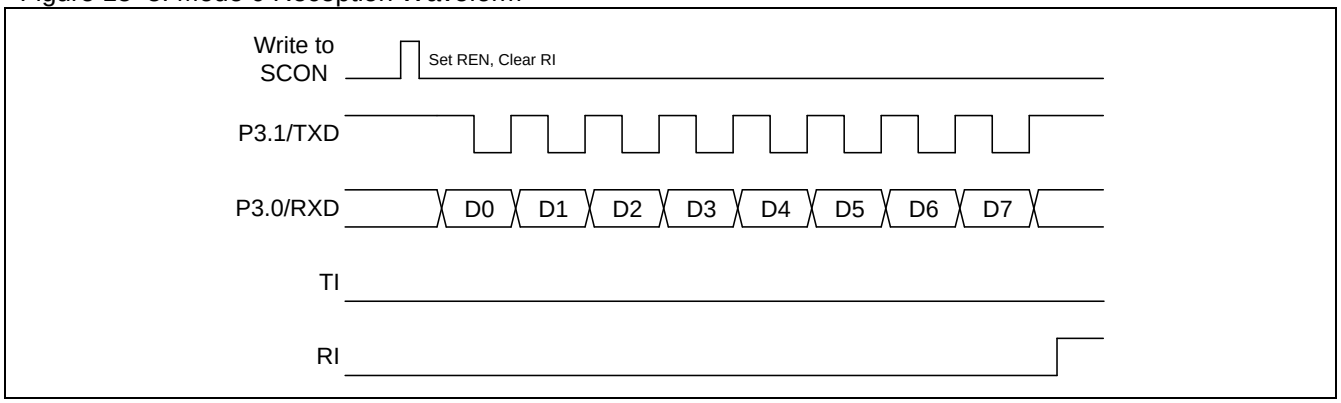


Figure 15–5. Mode 0 Reception Waveform



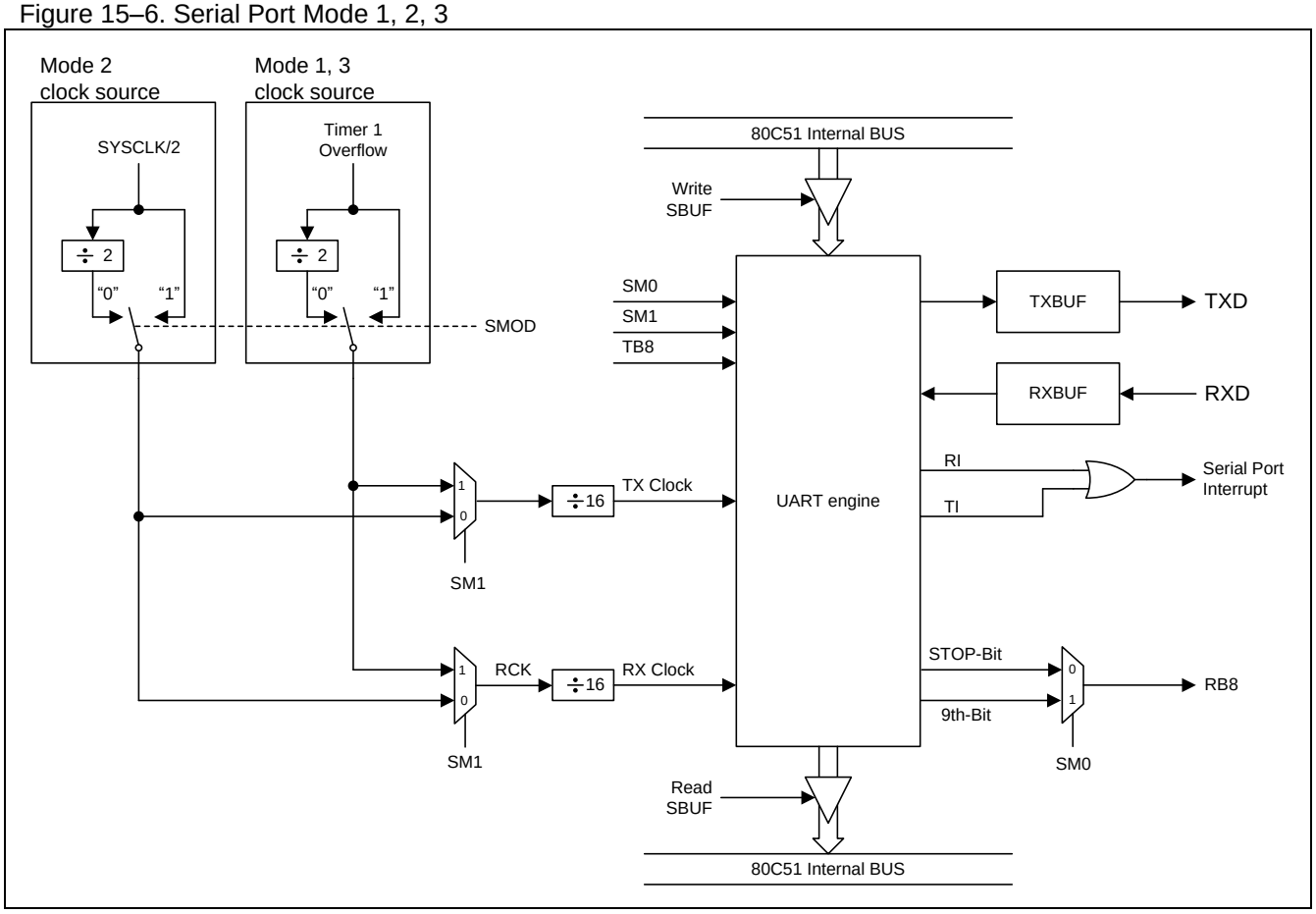
15.2. Serial Port Mode 1

10 bits are transmitted through TXD, or received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. The baud rate is determined by the Timer 1 overflow rate. Figure 15–1 shows the data frame in Mode 1 and Figure 15–6 shows a simplified functional diagram of the serial port in Mode 1.

Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal requests the UART engine to start the transmission. After receiving a transmission request, the UART engine would start the transmission at the raising edge of TX Clock. The data in the SBUF would be serial output on the TXD pin with the data frame as shown in Figure 15–1 and data width depend on TX Clock. After the end of 8th data transmission, TI would be asserted by hardware to indicate the end of data transmission.

Reception is initiated when Serial Port Controller detected 1-to-0 transition at RXD sampled by RCK. The data on the RXD pin would be sampled by Bit Detector in Serial Port Controller. After the end of STOP-bit reception, RI would be asserted by hardware to indicate the end of data reception and load STOP-bit into RB8 in SCON register.

Figure 15–6. Serial Port Mode 1, 2, 3



15.3. Serial Port Mode 2 and Mode 3

11 bits are transmitted through TXD, or received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to select either 1/32 or 1/64 the system clock frequency in Mode 2. Mode 3 may have a variable baud rate generated from Timer 1.

Figure 15–2 shows the data frame in Mode 2 and Mode 3. Figure 15–6 shows a functional diagram of the serial port in Mode 2 and Mode 3. The receive portion is exactly the same as in Mode 1. The transmit portion differs from Mode 1 only in the 9th bit of the transmit shift register.

The “write to SBUF” signal requests the Serial Port Controller to load TB8 into the 9th bit position of the transmit shit register and starts the transmission. After receiving a transmission request, the UART engine would start the transmission at the raising edge of TX Clock. The data in the SBUF would be serial output on the TXD pin with the data frame as shown in Figure 15–2 and data width depend on TX Clock. After the end of 9th data transmission, TI would be asserted by hardware to indicate the end of data transmission.

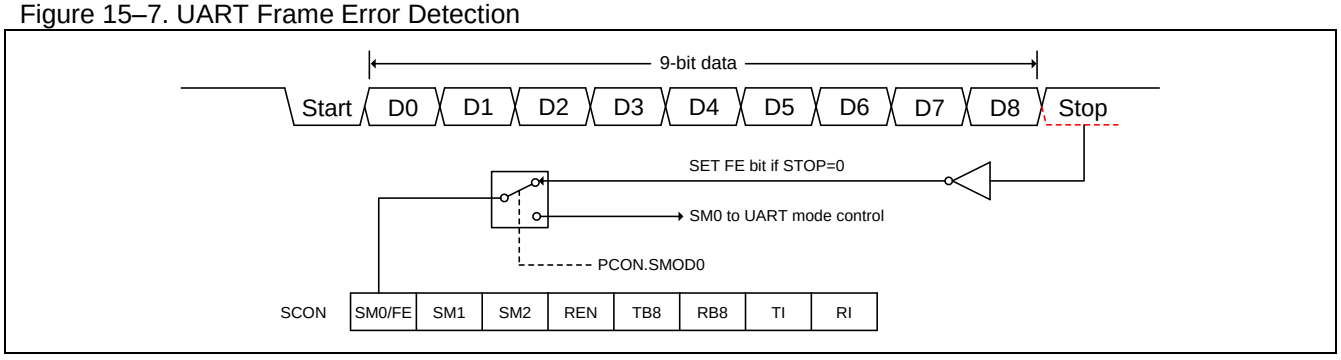
Reception is initiated when the UART engine detected 1-to-0 transition at RXD sampled by RCK. The data on the RXD pin would be sampled by Bit Detector in UART engine. After the end of 9th data bit reception, RI would be asserted by hardware to indicate the end of data reception and load the 9th data bit into RB8 in SCON register.

In all four modes, transmission is initiated by any instruction that use SBUF as a destination register. Reception is initiated in mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit with 1-to-0 transition if REN=1.

15.4. Frame Error Detection

When used for framing error detection, the UART looks for missing stop bits in the communication. A missing stop bit will set the FE bit in the SCON register. The FE bit shares the SCON.7 bit with SM0 and the function of SCON.7 is determined by SMOD0 bit (PCON.6). If SMOD0 is set then SCON.7 functions as FE. SCON.7 functions as SM0 when SMOD0 is cleared. When SCON.7 functions as FE, it can only be cleared by firmware. Refer to Figure 15–7.

Figure 15–7. UART Frame Error Detection



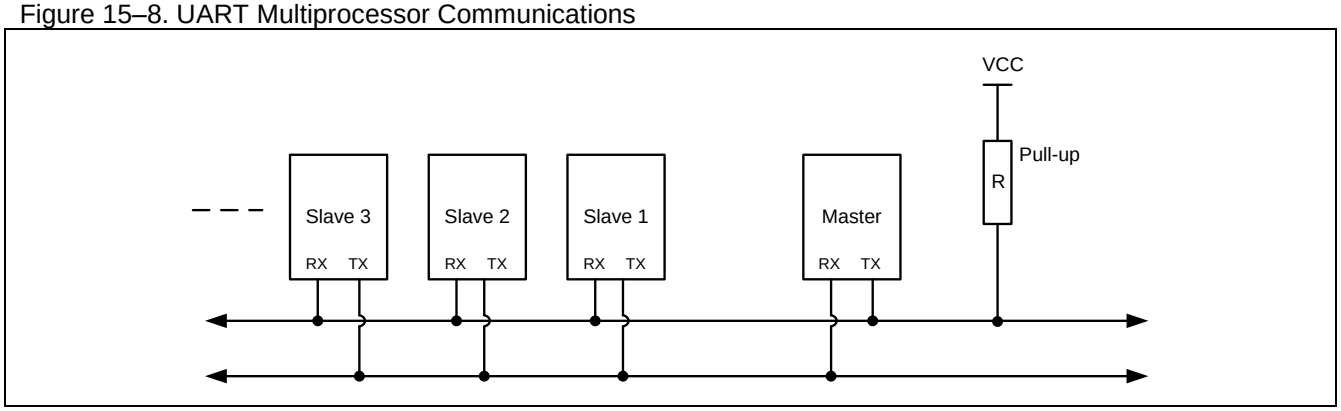
15.5. Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications as shown in Figure 15–8. In these two modes, 9 data bits are received. The 9th bit goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8=1. This feature is enabled by setting bit SM2 (in SCON register). A way to use this feature in multiprocessor systems is as follows:

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2=1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and check if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2 set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2=1, the receive interrupt will not be activated unless a valid stop bit is received.

Figure 15–8. UART Multiprocessor Communications



15.6. Automatic Address Recognition

Automatic Address Recognition is a feature which allows the UART to recognize certain addresses in the serial bit stream by using hardware to make the comparisons. This feature saves a great deal of firmware overhead by eliminating the need for the firmware to examine every serial address which passes by the serial port. This feature is enabled by setting the SM2 bit in SCON.

In the 9 bit UART modes, mode 2 and mode 3, the Receive Interrupt flag (RI) will be automatically set when the received byte contains either the "Given" address or the "Broadcast" address. The 9-bit mode requires that the 9th information bit is a 1 to indicate that the received information is an address and not data. Automatic address recognition is shown in Figure 15–9. The 8 bit mode is called Mode 1. In this mode the RI flag will be set if SM2 is enabled and the information received has a valid stop bit following the 8 address bits and the information is either a Given or Broadcast address. Mode 0 is the Shift Register mode and SM2 is ignored.

Using the Automatic Address Recognition feature allows a master to selectively communicate with one or more slaves by invoking the Given slave address or addresses. All of the slaves may be contacted by using the Broadcast address. Two special Function Registers are used to define the slave's address, SADDR, and the address mask, SADEN.

SADEN is used to define which bits in the SADDR are to be used and which bits are "don't care". The SADEN mask can be logically ANDed with the SADDR to create the "Given" address which the master will use for addressing each of the slaves. Use of the Given address allows multiple slaves to be recognized while excluding others.

The following examples will help to show the versatility of this scheme:

Slave 0	Slave 1
SADDR = 1100 0000	SADDR = 1100 0000
SADEN = 1111 1101	SADEN = 1111 1110
Given = 1100 00X0	Given = 1100 000X

In the above example SADDR is the same and the SADEN data is used to differentiate between the two slaves. Slave 0 requires a 0 in bit 0 and it ignores bit 1. Slave 1 requires a 0 in bit 1 and bit 0 is ignored. A unique address for Slave 0 would be 1100 0010 since slave 1 requires a 0 in bit 1. A unique address for slave 1 would be 1100 0001 since a 1 in bit 0 will exclude slave 0. Both slaves can be selected at the same time by an address which has bit 0 = 0 (for slave 0) and bit 1 = 0 (for slave 1). Thus, both could be addressed with 1100 0000.

In a more complex system the following could be used to select slaves 1 and 2 while excluding slave 0:

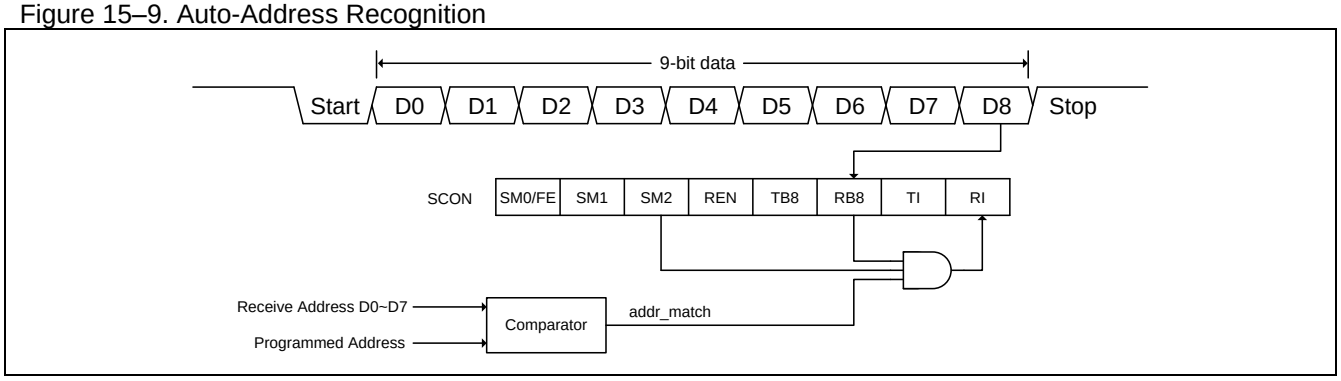
Slave 0	Slave 1	Slave 2
SADDR = 1100 0000	SADDR = 1110 0000	SADDR = 1110 0000
SADEN = 1111 1001	SADEN = 1111 1010	SADEN = 1111 1100
Given = 1100 0XX0	Given = 1110 0X0X	Given = 1110 00XX

In the above example the differentiation among the 3 slaves is in the lower 3 address bits. Slave 0 requires that bit 0 = 0 and it can be uniquely addressed by 1110 0110. Slave 1 requires that bit 1 = 0 and it can be uniquely addressed by 1110 0101. Slave 2 requires that bit 2 = 0 and its unique address is 1110 0011. To select Slaves 0 and 1 and exclude Slave 2 use address 1110 0100, since it is necessary to make bit 2 = 1 to exclude slave 2.

The Broadcast Address for each slave is created by taking the logical OR of SADDR and SADEN. Zeros in this result are treated as don't-cares. In most cases, interpreting the don't-cares as ones, the broadcast address will be FF hexadecimal.

Upon reset SADDR (SFR address 0xA9) and SADEN (SFR address 0xB9) are loaded with 0s. This produces a given address of all "don't cares" as well as a Broadcast address of all "don't cares". This effectively disables the Automatic Addressing mode and allows the micro-controller to use standard 80C51 type UART drivers which do not make use of this feature.

Figure 15–9. Auto-Address Recognition



Note:
(1) After address matching (addr_match=1), Clear SM2 to receive data bytes
(2) After all data bytes have been received, Set SM2 to wait for next address.

15.7. Baud Rate Setting

Bits T1X12 and URM0X6 in AUXR register provide a new option for the baud rate setting, as listed below.

15.7.1. Baud Rate in Mode 0

Figure 15–10. Mode 0 baud rate equation

$$\text{Mode 0 Baud Rate} = \frac{F_{SYSCLK}}{n} \quad \begin{array}{l} ; \text{ n=12, if URM0X6=0} \\ ; \text{ n=2, if URM0X6=1} \end{array}$$

Note:

If URM0X6=0, the baud rate formula is as same as standard 8051.

Table 15–1. Serial Port Mode 0 baud rate example

SYSCLK	URM0X6	Mode 0 Baud Rate
12MHz	0	1M bps
12MHz	1	6M bps
24MHz	0	2M bps
24MHz	1	12M bps

15.7.2. Baud Rate in Mode 2

Figure 15–11. Mode 2 baud rate equation

$$\text{Mode 2 Baud Rate} = \frac{2^{SMOD}}{64} \times (F_{SYSCLK})$$

Table 15–2. Serial Port Mode 2 baud rate example

SYSCLK	SMOD	Mode 2 Baud Rate
22.1184MHz	0	345.6K bps
22.1184MHz	1	172.8K bps
24MHz	0	750K bps
24MHz	1	375K bps

15.7.3. Baud Rate in Mode 1 & 3

Using Timer 1 as the Baud Rate Generator

Figure 15–12. Mode 1/3 baud rate equation

$$\text{Mode 1, 3 Baud Rate} = \frac{2^{SMOD}}{32} \times \frac{F_{SYSCLK}}{n \times (256 - TH1)} \quad \begin{array}{l} ; \text{ n=12, if T1X12=0} \\ ; \text{ n=1, if T1X12=1} \end{array}$$

"Table 15–3 ~ Table 15–6" list various commonly used baud rates and how they can be obtained from Timer 1 in its 8-Bit Auto-Reload Mode.

Table 15–3. Timer 1 Generated Commonly Used Baud Rates @ F_{SYSCLK}=11.0592MHz

Baud Rate	**TH1**, the Reload Value					
	T1X12=0			**T1X12=1**		
	SMOD=0	SMOD=1	Error	SMOD=0	SMOD=1	Error
1200	232	208	0.0%	--	--	--
2400	244	232	0.0%	112	--	0.0%
4800	250	244	0.0%	184	112	0.0%
9600	253	250	0.0%	220	184	0.0%
14400	254	252	0.0%	232	208	0.0%
19200	--	253	0.0%	238	220	0.0%
28800	255	254	0.0%	244	232	0.0%
38400	--	--	--	247	238	0.0%
57600	--	255	0.0%	250	244	0.0%
115200	--	--	--	253	250	0.0%
230400	--	--	--	--	253	0.0%

Table 15–4. Timer 1 Generated Commonly Used Baud Rates @ F_{SYSCLK}=22.1184MHz

Baud Rate	**TH1**, the Reload Value					
	T1X12=0			**T1X12=1**		
	SMOD=0	SMOD=1	Error	SMOD=0	SMOD=1	Error
1200	208	160	0.0%	--	--	--
2400	232	208	0.0%	--	--	0.0%
4800	244	232	0.0%	112	--	0.0%
9600	250	244	0.0%	184	112	0.0%
14400	252	248	0.0%	208	160	0.0%
19200	253	250	0.0%	220	184	0.0%
28800	254	252	0.0%	232	208	0.0%
38400	--	253	0.0%	238	220	0.0%
57600	255	254	0.0%	244	232	0.0%
115200	--	255	0.0%	250	244	0.0%
230400	--	--	--	253	250	0.0%
460800	--	--	--	--	253	0.0%

Table 15–5. Timer 1 Generated Commonly Used Baud Rates @ F_{SYSCLK}=12.0MHz

Baud Rate	**TH1**, the Reload Value					
	T1X12=0			**T1X12=1**		
	SMOD=0	SMOD=1	Error	SMOD=0	SMOD=1	Error
1200	230	204	0.16%	--	--	--
2400	243	230	0.16%	100	--	0.16%
4800	--	243	0.16%	178	100	0.16%
9600	--	--	--	217	178	0.16%
14400	--	--	--	230	204	0.16%
19200	--	--	--	--	217	0.16%
28800	--	--	--	243	230	0.16%
38400	--	--	--	246	236	2.34%
57600	--	--	--	--	243	0.16%
115200	--	--	--	--	--	--

Table 15–6. Timer 1 Generated Commonly Used Baud Rates @ F_{SYSCLK}=24.0MHz

Baud Rate	**TH1**, the Reload Value					
	T1X12=0			**T1X12=1**		
	SMOD=0	SMOD=1	Error	SMOD=0	SMOD=1	Error
1200	204	152	0.16%	--	--	--
2400	230	204	0.16%	--	--	--
4800	243	230	0.16%	100	--	0.16%
9600	--	243	0.16%	178	100	0.16%
14400	--	--	--	204	152	0.16%
19200	--	--	--	217	178	0.16%
28800	--	--	--	230	204	0.16%
38400	--	--	--	--	217	0.16%
57600	--	--	--	243	230	0.16%
115200	--	--	--	--	243	0.16%

15.8. Serial Port Register

All the four operation modes of the serial port are the same as those of the standard 8051 except the baud rate setting. Two registers, PCON and AUXR, are related to the baud rate setting:

SCON: Serial port Control Register

SFR Address = 0x98 RESET = 0000-0000

7	6	5	4	3	2	1	0
SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: FE, Framing Error bit. The SMOD0 bit must be set to enable access to the FE bit.
0: The FE bit is not cleared by valid frames but should be cleared by software.
1: This bit is set by the receiver when an invalid stop bit is detected.

Bit 7: SM0, Serial port mode bit 0, (SMOD0 must = 0 to access bit SM0)
Bit 6: SM1, Serial port mode bit 1.

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	SYSCLK/12 or SYSCLK/2
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	SYSCLK/64, /32
1	1	3	9-bit UART	variable

Bit 5: SM2, Serial port mode bit 2.
0: Disable SM2 function.
1: Enable the automatic address recognition feature in Modes 2 and 3. If SM2=1, RI will not be set unless the received 9th data bit is 1, indicating an address, and the received byte is a Given or Broadcast address. In mode1, if SM2=1 then RI will not be set unless a valid stop Bit was received, and the received byte is a Given or Broadcast address. In Mode 0, SM2 should be 0.

Bit 4: REN, Enable serial reception.
0: Clear by software to disable reception.
1: Set by software to enable reception.

Bit 3: TB8, The 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired.

Bit 2: RB8, In Modes 2 and 3, the 9^{th} data bit that was received. In Mode 1, if SM2 = 0, RB8 is the stop bit that was received. In Mode 0, RB8 is not used.

Bit 1: TI. Transmit interrupt flag.
0: Must be cleared by software.
1: Set by hardware at the end of the 8^{th} bit time in Mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission.

Bit 0: RI. Receive interrupt flag.
0: Must be cleared by software.
1: Set by hardware at the end of the 8^{th} bit time in Mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2).

SBUF: Serial Buffer Register

SFR Address = 0x99 RESET = XXXX-XXXX

7	6	5	4	3	2	1	0
SBUF.7	SBUF.6	SBUF.5	SBUF.4	SBUF.3	SBUF.2	SBUF.1	SBUF.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: It is used as the buffer register in transmission and reception.

SADDR: Slave Address Register

SFR Address = 0xA9 RESET = 0000-0000

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

SADEN: Slave Address Mask Register

SFR Address = 0xB9 RESET = 0000-0000

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

SADDR register is combined with SADEN register to form Given/Broadcast Address for automatic address recognition. In fact, SADEN functions as the "mask" register for SADDR register. The following is the example for it.

SADDR = 1100 0000
SADEN = 1111 1101
Given = 1100 00x0 ⟶ The Given slave address will be checked except bit 1 is treated as "don't care"

The Broadcast Address for each slave is created by taking the logical OR of SADDR and SADEN. Zero in this result is considered as "don't care". Upon reset, SADDR and SADEN are loaded with all 0s. This produces a Given Address of all "don't care" and a Broadcast Address of all "don't care". This disables the automatic address detection feature.

PCON: Power Control Register

SFR Address = 0x87 POR = 0011-0000, RESET = 0000-0000

7	6	5	4	3	2	1	0
SMOD	SMOD0	LVF	POF	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: SMOD, double Baud rate control bit.
0: Disable double Baud rate of the UART.
1: Enable double Baud rate of the UART in mode 1, 2, or 3.

Bit 6: SMOD0, Frame Error select.
0: SCON.7 is SM0 function.
1: SCON.7 is FE function. Note that FE will be set after a frame error regardless of the state of SMOD0.

AUXR: Auxiliary Register

SFR Address = 0x8E RESET = 0000-00xx

7	6	5	4	3	2	1	0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	--	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 6: T1X12, Timer 1 clock source selector while C/T=0.
0: Clear to select SYSCLK/12.
1: Set to select SYSCLK as the clock source.

Bit 5: URM0X6, Serial Port mode 0 baud rate selector.
0: Clear to select SYSCLK/12 as the baud rate for UART Mode 0.
1: Set to select SYSCLK/2 as the baud rate for UART Mode 0.

15.9. Serial Port Sample Code

(1). Required Function: IDLE mode with RI wake-up capability

Assembly Code Example:

```
PS                      EQU                     10h
PSH                     EQU                     10h


   ORG    00023h
uart_ri_idle_isr:
   JB      RI,RI_ISR                ;
   JB      TI,TI_ISR                ;
   RETI                            ;

RI_ISR:
;  Process
   CLR     RI                       ;
   RETI                            ;

TI_ISR:
;  Process
   CLR     TI                       ;
   RETI                            ;

main:
   CLR     TI                       ;
   CLR     RI                       ;
   SETB    SM1                      ;
   SETB    REN                      ; 8bit Mode2, Receive Enable

   CALL    UART_Baud_Rate_Setting       ;refer to "Table 15–3 ~ Table 15–6" for detailed information

   MOV     IP,#PSL                  ; Select UART interrupt priority
   MOV     IPH,#PSH                 ;

   SETB    ES                       ; Enable S0 interrupt
   SETB    EA                       ; Enable global interrupt

   ORL     PCON,#IDL;               ; Set MCU into IDLE mode
```

C Code Example:

```
#define  PS                                 0x10
#define  PSH                                0x10

void uart_ri_idle_isr(void) interrupt 4
{  if(RI)
   {
      RI=0;
      // to do ...
   }

   if(TI)
   {
      TI=0;
      // to do ...
   }
}

void main(void)
{
   TI = RI = 0;
   SM1 = REN = 1;                   // 8bit Mode2, Receive Enable

   UART_Baud_Rate_Setting()         // refer to "Table 15–3 ~ Table 15–6" for detailed information

   IP = PSL;                        // Select S0 interrupt priority
```

```
    IPH = PSH;                      //

    ES = 1;                         // Enable S0 interrupt
    EA = 1;                         // Enable global interrupt

    PCON |= IDL;                        // Set MCU into IDLE mode
}
```

16. Programmable Counter Array (PCA)

The **MPC82E/L52** is equipped with a Programmable Counter Array (PCA), which provides more timing capabilities with less CPU intervention than the standard timer/counters. Its advantages include reduced software overhead and improved accuracy.

16.1. PCA Overview

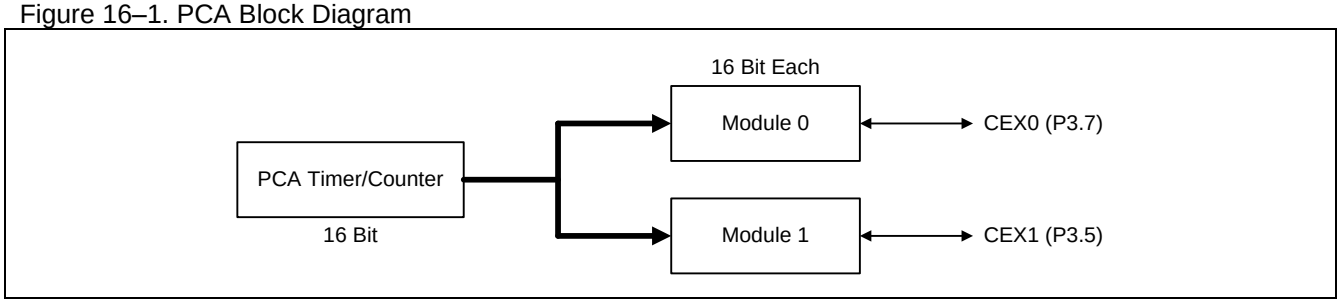
The PCA consists of a dedicated timer/counter which serves as the time base for an array of two compare/capture modules. Figure 16–1 shows a block diagram of the PCA. Notice that the PCA timer and modules are all 16-bits. If an external event is associated with a module, that function is shared with the corresponding Port pin. If the module is not using the port pin, the pin can still be used for standard I/O.

Each of the four modules can be programmed in any one of the following modes:

- Rising and/or Falling Edge Capture
- Software Timer
- High Speed Output
- Pulse Width Modulator (PWM) Output

All of these modes will be discussed later in detail. However, let's first look at how to set up the PCA timer and modules.

Figure 16–1. PCA Block Diagram



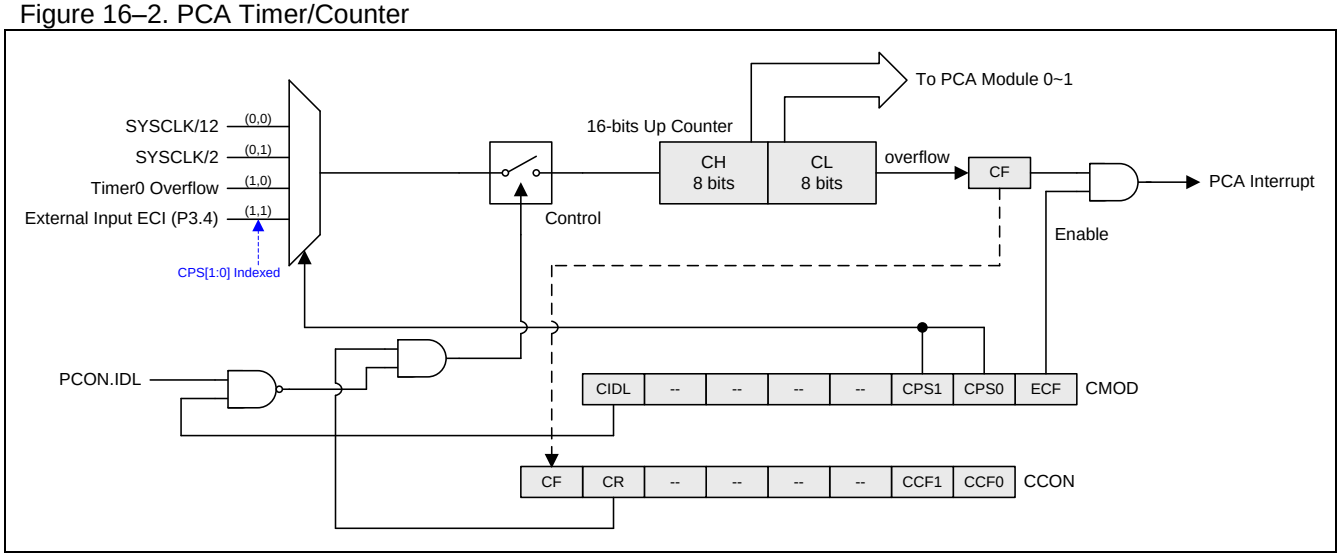
16.2. PCA Timer/Counter

The timer/counter for the PCA is a free-running16-bit timer consisting of registers CH, CL (the high and low bytes of the count values), as shown in Figure 16–2. It is the common time base for all modules and its clock input can be selected from the following source:

- 1/12 the system clock frequency,
- 1/2 the system clock frequency,
- the Timer 0 overflow, which allows for a range of slower clock inputs to the timer.
- external clock input, 1-to-0 transitions, on ECI pin (P3.4),

Special Function Register CMOD contains the Count Pulse Select bits (CPS1 and CPS0) to specify the PCA timer input. This register also contains the ECF bit which enables an interrupt when the counter overflows. In addition, the user has the option of turning off the PCA timer during Idle Mode by setting the Counter Idle bit (CIDL). This can further reduce power consumption during Idle mode.

Figure 16–2. PCA Timer/Counter



CMOD: PCA Counter Mode Register

SFR Address = 0xD9 RESET = 0xxx-x000

7	6	5	4	3	2	1	0
CIDL	--	--	--	--	CPS1	CPS0	ECF
R/W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: CIDL, PCA counter Idle control.
0: Lets the PCA counter continue functioning during Idle mode.
1: Lets the PCA counter be gated off during Idle mode.

Bit 6~3: Reserved. Software must write “0” on these bits when CMOD is written.

Bit 3~1: CPS2-CPS0, PCA counter clock source select bits.

CPS1	CPS0	PCA Clock Source
0	0	Internal clock, SYSCLK/12
0	1	Internal clock, SYSCLK/2
1	0	Timer 0 overflow
1	1	External clock at the ECI pin

Bit 0: ECF, Enable PCA counter overflow interrupt.
0: Disables an interrupt when CF bit (in CCON register) is set.
1: Enables an interrupt when CF bit (in CCON register) is set.

The CCON register shown below contains the run control bit for the PCA and the flags for the PCA timer and each module. To run the PCA the CR bit (CCON.6) must be set by software. The PCA is shut off by clearing this bit. The CF bit (CCON.7) is set when the PCA counter overflows and an interrupt will be generated if the ECF bit in the CMOD register is set. The CF bit can only be cleared by software. CCF0 and CCF1 are the interrupt flags for module 0 and module 1, respectively, and they are set by hardware when either a match or a capture occurs. These flags also can only be cleared by software. The PCA interrupt system is shown Figure 16–3.

CCON: PCA Counter Control Register

SFR Address = 0xD8 RESET = 00xx-xx00

7	6	5	4	3	2	1	0
CF	CR	--	--	--	--	CCF1	CCF0
R/W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7: CF, PCA Counter Overflow flag.
0: Only be cleared by software.
1: Set by hardware when the counter rolls over. CF flag can generate an interrupt if bit ECF in CMOD is set. CF may be set by either hardware or software.

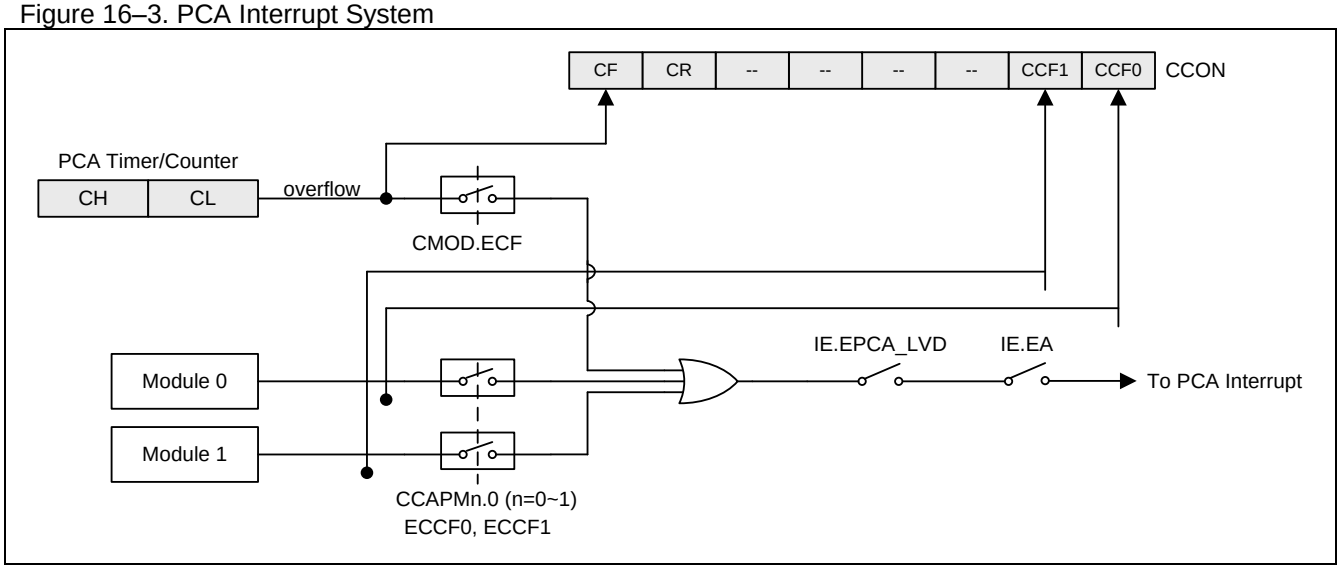
Bit 6: CR, PCA Counter Run control bit.
0: Must be cleared by software to turn the PCA counter off.
1: Set by software to turn the PCA counter on.

Bit 5~2: Reserved. Software must write “0” on these bits when CCON is written.

Bit 1: CCF1, PCA Module 1 interrupt flag.
0: Must be cleared by software.
1: Set by hardware when a match or capture occurs.

Bit 0: CCF0, PCA Module 0 interrupt flag.
0: Must be cleared by software.
1: Set by hardware when a match or capture occurs.

Figure 16–3. PCA Interrupt System



16.3. Compare/Capture Modules

Each of the four compare/capture modules has a mode register called CCAPMn (n= 0 or 1) to select which function it will perform. Note the ECCFn bit which enables an interrupt to occur when a module's interrupt flag is set.

CCAPMn: PCA Module Compare/Capture Register, n=0~1

SFR Address = 0xDA~0xDB RESET = x000-0000

7	6	5	4	3	2	1	0
--	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: Reserved. Software must write “0” on this bit when the CCAPMn is written.

Bit 6: ECOMn, Enable Comparator
0: Disable the digital comparator function.
1: Enables the digital comparator function.

Bit 5: CAPPn, Capture Positive enabled.
0: Disable the PCA capture function on CEXn positive edge detected.
1: Enable the PCA capture function on CEXn positive edge detected.

Bit 4: CAPNn, Capture Negative enabled.
0: Disable the PCA capture function on CEXn positive edge detected.
1: Enable the PCA capture function on CEXn negative edge detected.

Bit 3: MATn, Match control.
0: Disable the digital comparator match event to set CCFn.
1: A match of the PCA counter with this module’s compare/capture register causes the CCFn bit in CCON to be set.

Bit 2: TOGn, Toggle control.
0: Disable the digital comparator match event to toggle CEXn.
1: A match of the PCA counter with this module’s compare/capture register causes the CEXn pin to toggle.

Bit 1: PWMn, PWM control.
0: Disable the PWM mode in PCA module.
1: Enable the PWM function and cause CEXn pin to be used as a pulse width modulated output.

Bit 0: ECCFn, Enable CCFn interrupt.
0: Disable compare/capture flag CCFn in the CCON register to generate an interrupt.
1: Enable compare/capture flag CCFn in the CCON register to generate an interrupt.

Note: The bits CAPNn (CCAPMn.4) and CAPPn (CCAPMn.5) determine the edge on which a capture input will be active. If both bits are set, both edges will be enabled and a capture will occur for either transition.

Each module also has a pair of 8-bit compare/capture registers (CCAPnH, CCAPnL) associated with it. These registers are used to store the time when a capture event occurred or when a compare event should occur.
When a module is used in the PWM mode, in addition to the above two registers, an extended register PCAPWMn is used to improve the range of the duty cycle of the output. The improved range of the duty cycle starts from 0%, up to 100%, with a step of 1/256.

CCAPnH: PCA Module n Capture High Register, n=0~1

SFR Address = 0xFA~0xFD RESET = 0000-0000

7	6	5	4	3	2	1	0
CCAPnH.7	CCAPnH.6	CCAPnH.5	CCAPnH.4	CCAPnH.3	CCAPnH.2	CCAPnH.1	CCAPnH.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CCAPnL: PCA Module n Capture Low Register, n=0~1

SFR Address = 0xEA~0xED RESET = 0000-0000

7	6	5	4	3	2	1	0
CCAPnL.7	CCAPnL.6	CCAPnL.5	CCAPnL.4	CCAPnL.3	CCAPnL.2	CCAPnL.1	CCAPnL.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

PCAPWMn: PWM Mode Auxiliary Register, n=0~1

SFR Address = 0xF2~0xF5 RESET = xxxx-xx00

7	6	5	4	3	2	1	0
--	--	--	--	--	--	ECAPnH	ECAPnL
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7~2: Reserved. Software must write “0” on these bits when PCAPWMn is written.

Bit 1: ECAPnH, Extended 9th bit (MSB bit), associated with CCAPnH to become a 9-bit register used in PWM mode.

Bit 0: ECAPnL, Extended 9th bit (MSB bit), associated with CCAPnL to become a 9-bit register used in PWM mode.

16.4. Operation Modes of the PCA

Table 16–1 shows the CCAPMn register settings for the various PCA functions.

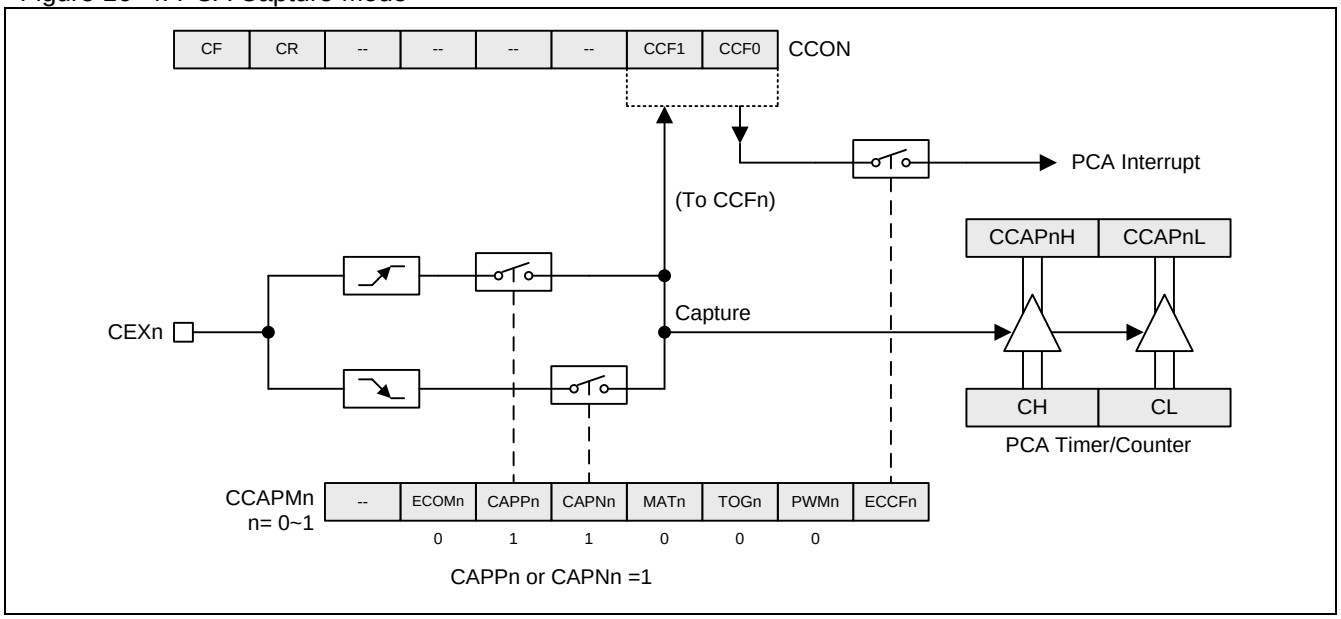
Table 16–1. PCA Module Modes

ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	Module Function
0	0	0	0	0	0	0	No operation
X	1	0	0	0	0	X	16-bit capture by a positive-edge trigger on CEXn
X	0	1	0	0	0	X	16-bit capture by a negative-edge trigger on CEXn
X	1	1	0	0	0	X	16-bit capture by a transition on CEXn
1	0	0	1	0	0	X	16-bit Software Timer
1	0	0	1	1	0	X	16-bit High Speed Output
1	0	0	0	0	1	0	8-bit Pulse Width Modulator (PWM)

16.4.1. Capture Mode

To use one of the PCA modules in the capture mode, either one or both of the bits CAPN and CAPP for that module must be set. The external CEX input for the module is sampled for a transition. When a valid transition occurs, the PCA hardware loads the value of the PCA counter registers (CH and CL) into the module's capture registers (CCAPnL and CCAPnH). If the CCFn and the ECCFn bits for the module are both set, an interrupt will be generated.

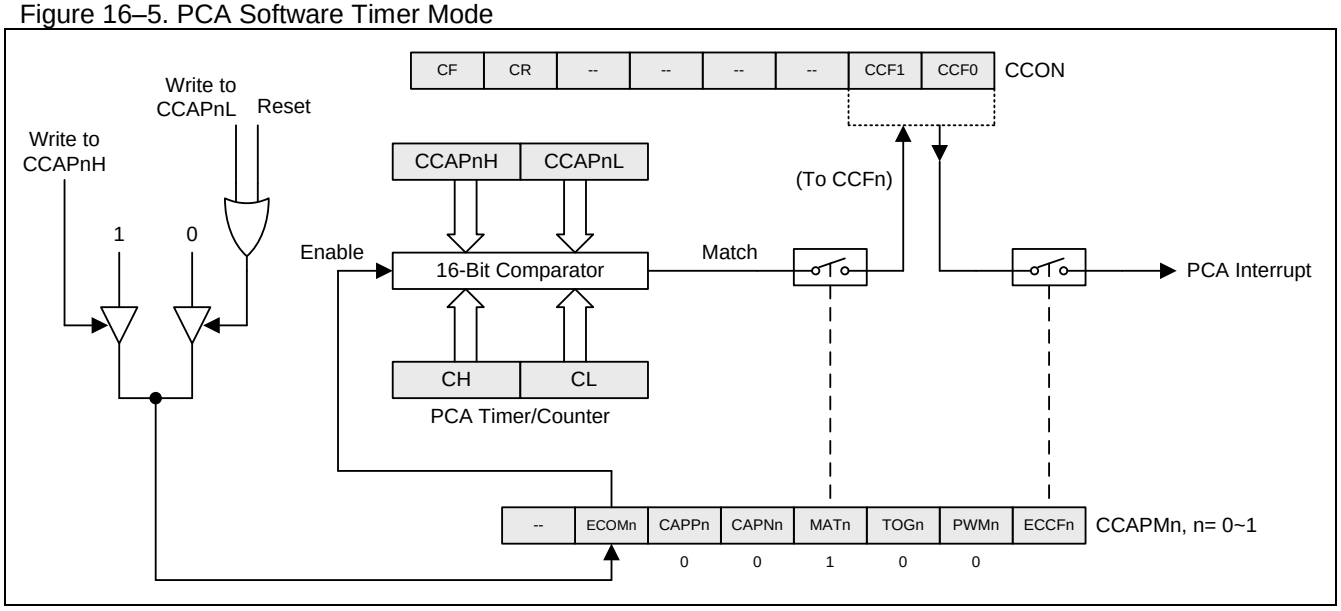
Figure 16–4. PCA Capture Mode



16.4.2. 16-bit Software Timer Mode

The PCA modules can be used as software timers by setting both the ECOM and MAT bits in the module's CCAPMn register. The PCA timer will be compared to the module's capture registers, and when a match occurs an interrupt will occur if the CCFn and the ECCFn bits for the module are both set.

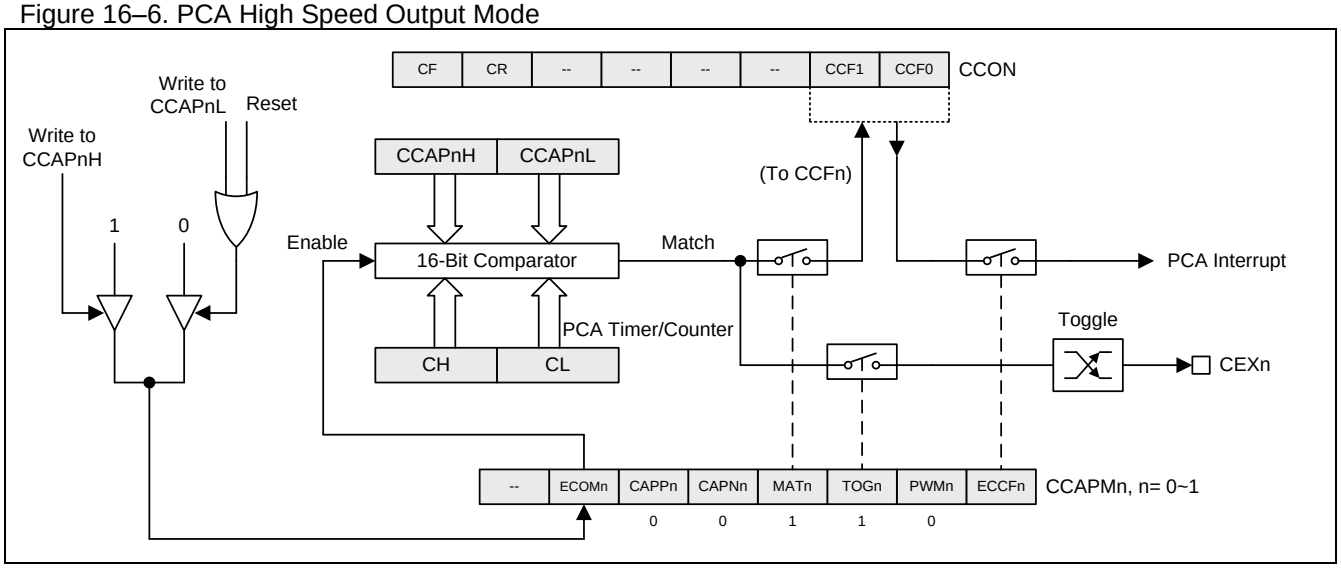
Figure 16–5. PCA Software Timer Mode



16.4.3. High Speed Output Mode

In this mode the CEX output associated with the PCA module will toggle each time a match occurs between the PCA counter and the module's capture registers. To activate this mode, the TOG, MAT and ECOM bits in the module's CCAPMn register must be set.

Figure 16–6. PCA High Speed Output Mode



16.4.4. PWM Mode

All of the PCA modules can be used as PWM outputs. The frequency of the output depends on the clock source for the PCA timer. All of the modules will have the same frequency of output because they all share the PCA timer.

The duty cycle of each module is determined by the module's capture register CCAPnL and the extended 9th bit, ECAPnL. When the 9-bit value of { 0, [CL] } is *less than* the 9-bit value of { ECAPnL, [CCAPnL] } the output will be low, and if *equal to or greater than* the output will be high.

When CL overflows from 0xFF to next clock input, { ECAPnL, [CCAPnL] } is reloaded with the value of { ECAPnH, [CCAPnH] }. This allows updating the PWM without glitches. The PWMn and ECOMn bits in the module's CCAPMn register must be set to enable the PWM mode.

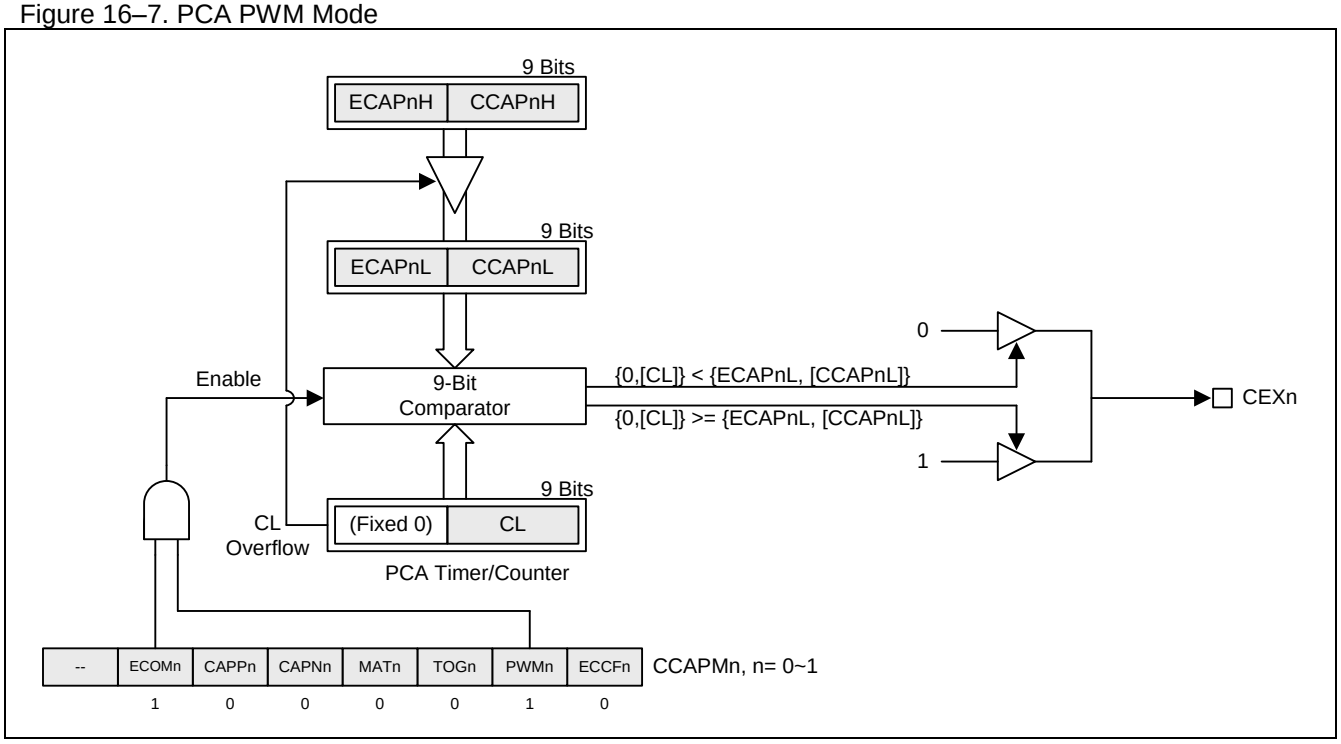
Using the 9-bit comparison, the duty cycle of the output can be improved to really start from 0%, and up to 100%. The formula for the duty cycle is:

Duty Cycle = 1 – { ECAPnH, [CCAPnH] } / 256.

Where, *[CCAPnH]* is the 8-bit value of the CCAPnH register, and *ECAPnH* (bit-1 in the PCAPWMn register) is 1-bit value. So, *{ ECAPnH, [CCAPnH] }* forms a 9-bit value for the 9-bit comparator. For examples,

a. If ECAPnH=0 & CCAPnH=0x00 (i.e., 0x000), the duty cycle is 100%.
b. If ECAPnH=0 & CCAPnH=0x40 (i.e., 0x040) the duty cycle is 75%.
c. If ECAPnH=0 & CCAPnH=0xC0 (i.e., 0x0C0), the duty cycle is 25%.
d. If ECAPnH=1 & CCAPnH=0x00 (i.e., 0x100), the duty cycle is 0%.

Figure 16–7. PCA PWM Mode



16.5. PCA Sample Code

(1). Required Function: Set Module 0 for CEX0 rising capture function and Module 1 for PWM output with 25% duty cycle

Assembly Code Example:

```
PWM1            EQU             02h
CAPP0           EQU             20h
ECOM1           EQU             40h

PWM2_PWM3:
        MOV             CCON,#00H                       ; stop CR
        MOV             CMOD,#02H                       ; PCA clock source = system clock / 2

        ORL             CCAPM0, #CAPP0                  ; capture rising edge

        ORL             CCAPM1, #(ECOM1 + PWM1)         ;module 1
        MOV             CCAP3H, #0C0h                   ; 25 %
        SETB            CR                              ; start PCA's PWM output
```

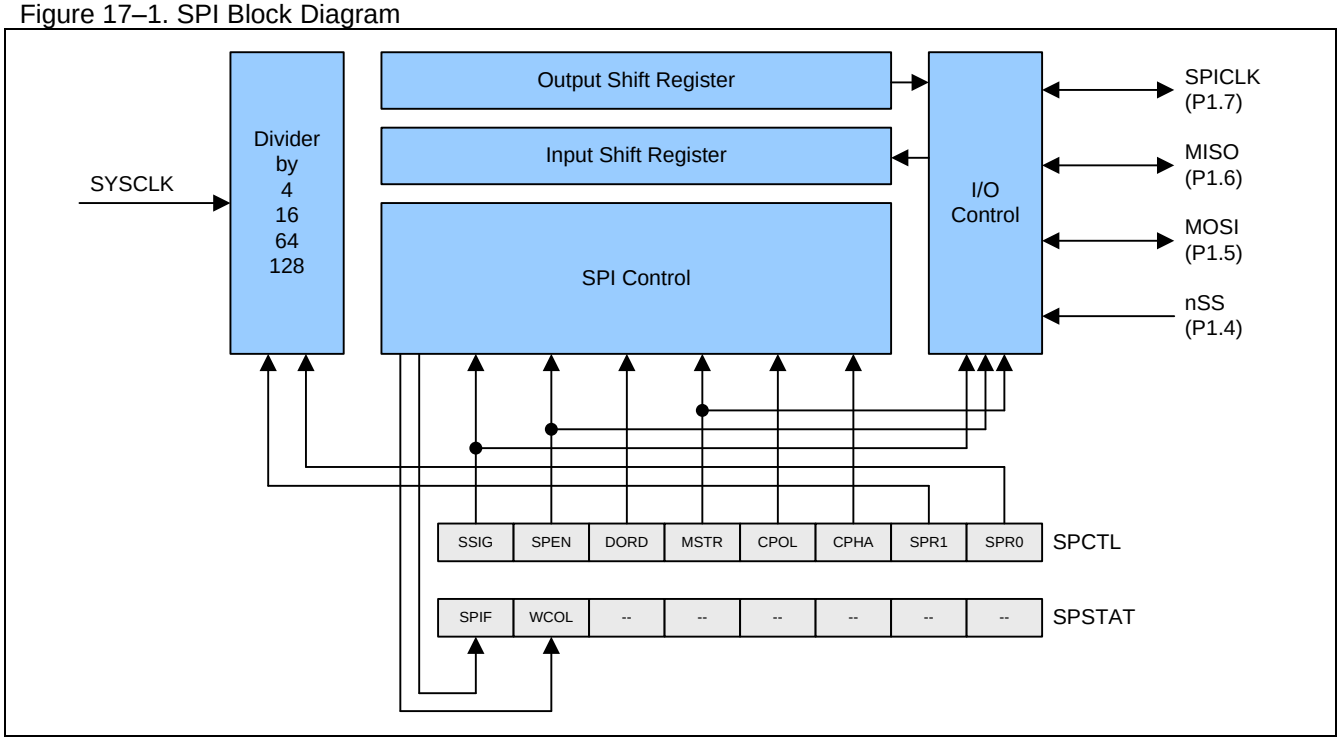
C Code Example:

```
#define  PWM1                   EQU             0x02
#define  CAPP0                  EQU             0x20
#define  ECOM1                  EQU             0x40


void main(void)
{
        // set PCA
        CCON = 0x00;                            // disable PCA & clear CCF0, CCF1, CF flag
        CMOD = 0x02;                            // PCA clock source = system clock / 2

    CCAPM0 |= CAPP0;                            // capture rising edge

        CCAPM1 |= (ECOM1 | PWM1);               // module 1
        CCAP3H = 0xC0;                          // 25 %

        CR = 1;                                 // start PCA's PWM output

        while (1);
}
```

17. Serial Peripheral Interface (SPI)

The **MPC82E/L52** provides a high-speed serial communication interface, the SPI interface. SPI is a full-duplex, high-speed and synchronous communication bus with two operation modes: Master mode and Slave mode. Up to 3 Mbps can be supported in either Master or Slave mode under a 12MHz system clock. It has a Transfer Completion Flag (SPIF) and Write Collision Flag (WCOL) in the SPI status register (SPSTAT).

Figure 17–1. SPI Block Diagram



The SPI interface has four pins: MISO (P1.6), MOSI (P1.5), SPICLK (P1.7) and nSS (P1.4):

• SPICLK, MOSI and MISO are typically tied together between two or more SPI devices. Data flows from master to slave on the MOSI pin (Master Out / Slave In) and flows from slave to master on the MISO pin (Master In / Slave Out). The SPICLK signal is output in the master mode and is input in the slave mode. If the SPI system is disabled, i.e., SPEN (SPCTL.6) = 0, these pins function as normal I/O pins.

• nSS is the optional slave select pin. In a typical configuration, an SPI master asserts one of its port pins to select one SPI device as the current slave. An SPI slave device uses its nSS pin to determine whether it is selected. The nSS is ignored if any of the following conditions are true:

- If the SPI system is disabled, i.e. SPEN (SPCTL.6) = 0 (reset value).
- If the SPI is configured as a master, i.e., MSTR (SPCTL.4) = 1, and P1.4 (nSS) is configured as an output.
- If the nSS pin is ignored, i.e. SSIG (SPCTL.7) bit = 1, this pin is configured for port functions.

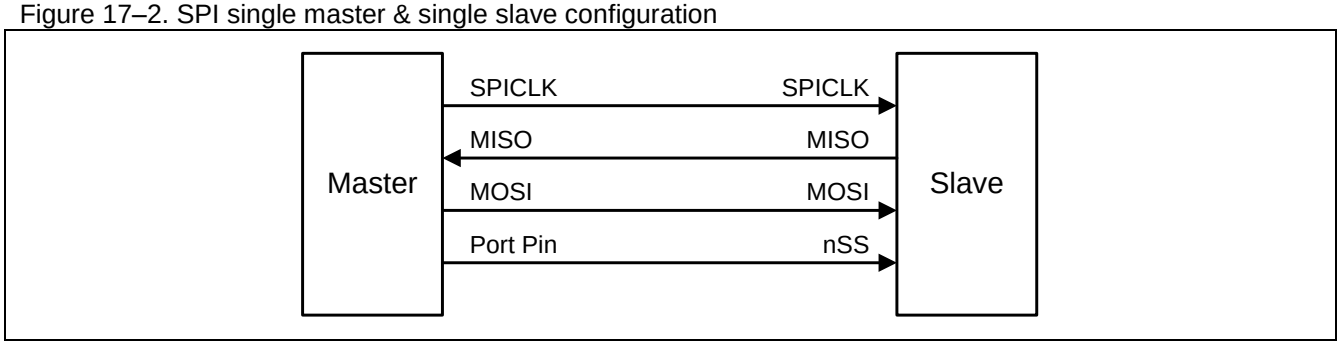
Note that even if the SPI is configured as a master (MSTR=1), it can still be converted to a slave by driving the nSS pin low (if SSIG=0). Should this happen, the SPIF bit (SPSTAT.7) will be set. (See Section “17.2.3 Mode Change on nSS-pin”)

17.1. Typical SPI Configurations

17.1.1. Single Master & Single Slave

For the master: any port pin, including P1.4 (nSS), can be used to drive the nSS pin of the slave.
For the slave: SSIG is '0', and nSS pin is used to determine whether it is selected.

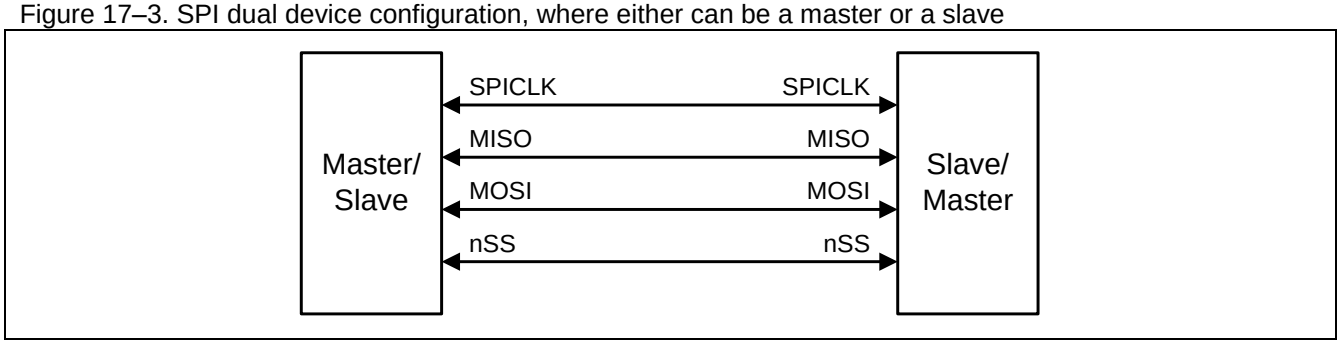
Figure 17–2. SPI single master & single slave configuration



17.1.2. Dual Device, where either can be a Master or a Slave

Two devices are connected to each other and either device can be a master or a slave. When no SPI operation is occurring, both can be configured as masters with MSTR=1, SSIG=0 and P1.4 (nSS) configured in quasi-bidirectional mode. When any device initiates a transfer, it can configure P1.4 as an output and drive it low to force a "mode change to slave" in the other device. (See Section "17.2.3 Mode Change on nSS-pin")

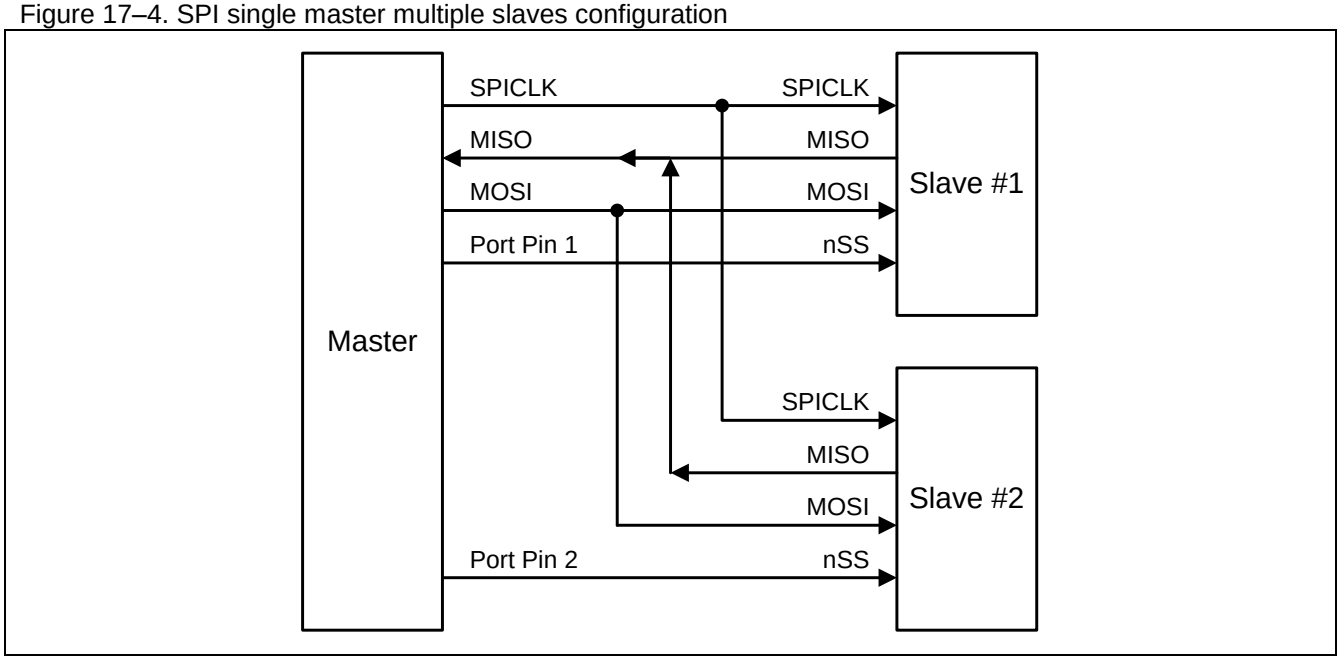
Figure 17–3. SPI dual device configuration, where either can be a master or a slave



17.1.3. Single Master & Multiple Slaves

For the master: any port pin, including P1.4 (nSS), can be used to drive the nSS pins of the slaves.
For all the slaves: SSIG is '0', and nSS pin are used to determine whether it is selected.

Figure 17–4. SPI single master multiple slaves configuration



17.2. Configuring the SPI

Table 17–1 shows configuration for the master/slave modes as well as usages and directions for the modes.

Table 17–1. SPI Master and Slave Selection

SPEN (SPCTL.6)	SSIG (SPCTL.7)	nSS -pin	MSTR (SPCTL.4)	Mode	MISO -pin	MOSI -pin	SPICLK -pin	Remarks
0	X	X	X	SPI disabled	input	input	input	P1.4~P1.7 are used as general port pins.
1	0	0	0	Salve (selected)	output	input	input	Selected as slave.
1	0	1	0	Slave (not selected)	Hi-Z	input	input	Not selected.
1	0	0	1 ➔ 0	Slave (by mode change)	output	input	input	Mode change to slave if nSS pin is driven low, and MSTR will be cleared to '0' by H/W automatically.
1	0	1	1	Master (idle)	input	Hi-Z	Hi-Z	MOSI and SPICLK are at high impedance to avoid bus contention when the Master is idle.
				Master (active)		output	output	MOSI and SPICLK are push-pull when the Master is active.
1	1	X	0	Slave	output	input	input	
1	1	X	1	Master	input	output	output	

"X" means "don't care".

17.2.1. Additional Considerations for a Slave

When CPHA is 0, SSIG must be 0 and nSS pin must be negated and reasserted between each successive serial byte transfer. Note the SPDAT register cannot be written while nSS pin is active (low), and the operation is undefined if CPHA is 0 and SSIG is 1.

When CPHA is 1, SSIG may be 0 or 1. If SSIG=0, the nSS pin may remain active low between successive transfers (can be tied low at all times). This format is sometimes preferred for use in systems having a single fixed master and a single slave configuration.

17.2.2. Additional Considerations for a Master

In SPI, transfers are always initiated by the master. If the SPI is enabled (SPEN=1) and selected as master, writing to the SPI data register (SPDAT) by the master starts the SPI clock generator and data transfer. The data will start to appear on MOSI about one half SPI bit-time to one SPI bit-time after data is written to SPDAT.

Before starting the transfer, the master may select a slave by driving the nSS pin of the corresponding device low. Data written to the SPDAT register of the master is shifted out of MOSI pin of the master to the MOSI pin of the slave. And, at the same time the data in SPDAT register of the selected slave is shifted out on MISO pin to the MISO pin of the master.

After shifting one byte, the SPI clock generator stops, setting the transfer completion flag (SPIF) and an interrupt will be created if the SPI interrupt is enabled. The two shift registers in the master CPU and slave CPU can be considered as one distributed 16-bit circular shift register. When data is shifted from the master to the slave, data is also shifted in the opposite direction simultaneously. This means that during one shift cycle, data in the master and the slave are interchanged.

17.2.3. Mode Change on nSS-pin

If SPEN=1, SSIG=0, MSTR=1 and nSS pin=1, the SPI is enabled in master mode. In this case, another master can drive this pin low to select this device as an SPI slave and start sending data to it. To avoid bus contention, the SPI becomes a slave. As a result of the SPI becoming a slave, the MOSI and SPICLK pins are forced to be an input and MISO becomes an output. The SPIF flag in SPSTAT is set, and if the SPI interrupt is enabled, an SPI interrupt will occur. User software should always check the MSTR bit. If this bit is cleared by a slave select and the user wants to continue to use the SPI as a master, the user must set the MSTR bit again, otherwise it will stay in slave mode.

17.2.4. Write Collision

The SPI is single buffered in the transmit direction and double buffered in the receive direction. New data for transmission can not be written to the shift register until the previous transaction is complete. The WCOL (SPSTAT.6) bit is set to indicate data collision when the data register is written during transmission. In this case, the data currently being transmitted will continue to be transmitted, but the new data, i.e., the one causing the collision, will be lost.

While write collision is detected for both a master or a slave, it is uncommon for a master because the master has full control of the transfer in progress. The slave, however, has no control over when the master will initiate a transfer and therefore collision can occur.

For receiving data, received data is transferred into a parallel read data buffer so that the shift register is free to accept a second character. However, the received character must be read from the Data Register (SPDAT) before the next character has been completely shifted in. Otherwise, the previous data is lost.

WCOL can be cleared in software by writing ‘1’ to the bit.

17.2.5. SPI Clock Rate Select

The SPI clock rate selection (in master mode) uses the SPR1 and SPR0 bits in the SPCTL register, as shown in Table 17–2.

Table 17–2. SPI Serial Clock Rates

SPR1	SPR0	SPI Clock Rate @ SYSCLK=12MHz	SYSCLK divided by
0	0	3 MHz	4
0	1	750 KHz	16
1	0	187.5 KHz	64
1	1	93.75 KHz	128

Where, SYSCLK is the system clock.

17.3. Data Mode

Clock Phase Bit (CPHA) allows the user to set the edges for sampling and changing data. The Clock Polarity bit, CPOL, allows the user to set the clock polarity. The following figures show the different settings of Clock Phase Bit, CPHA.

Figure 17–5. SPI Slave Transfer Format with CPHA=0

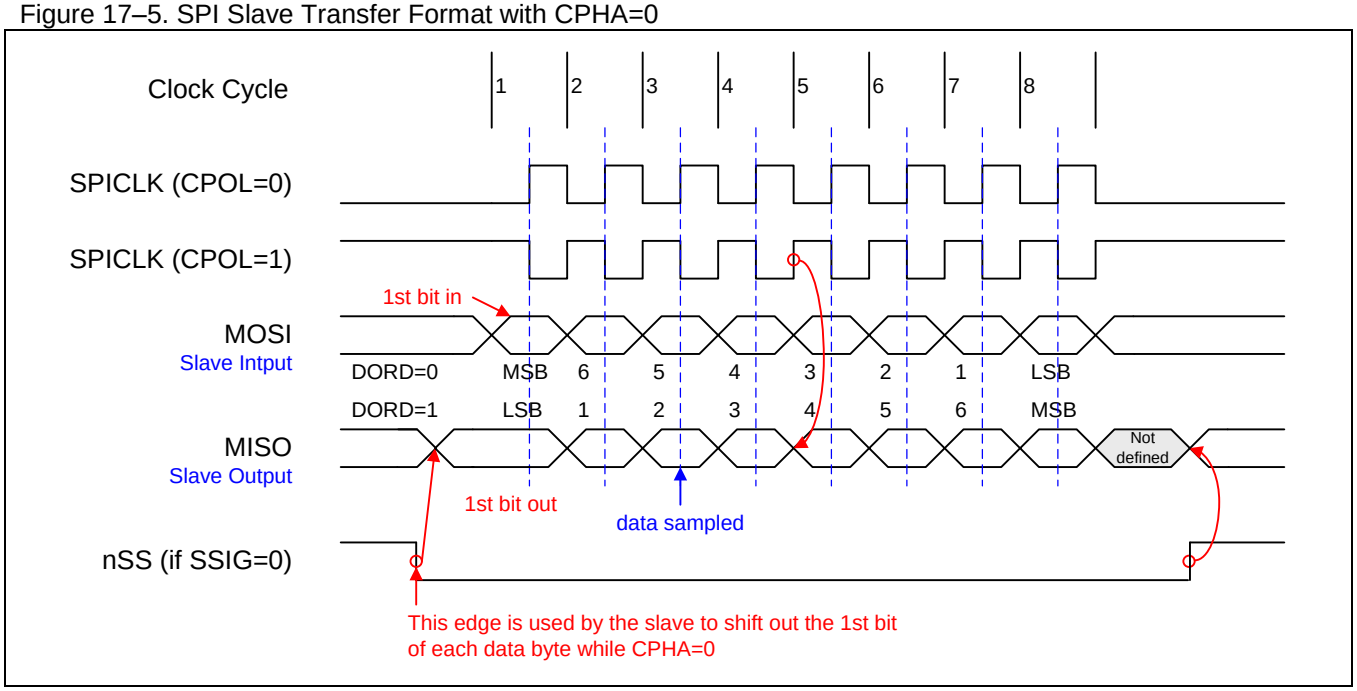


Figure 17–6. SPI Slave Transfer Format with CPHA=1

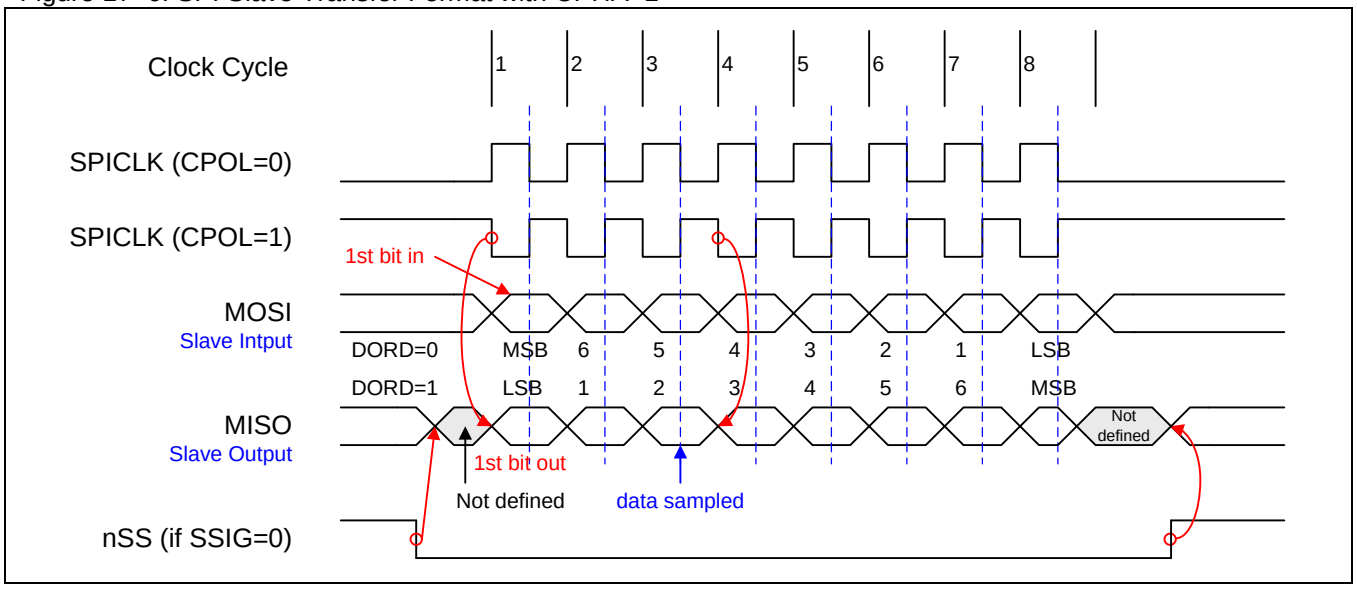


Figure 17–7. SPI Master Transfer Format with CPHA=0

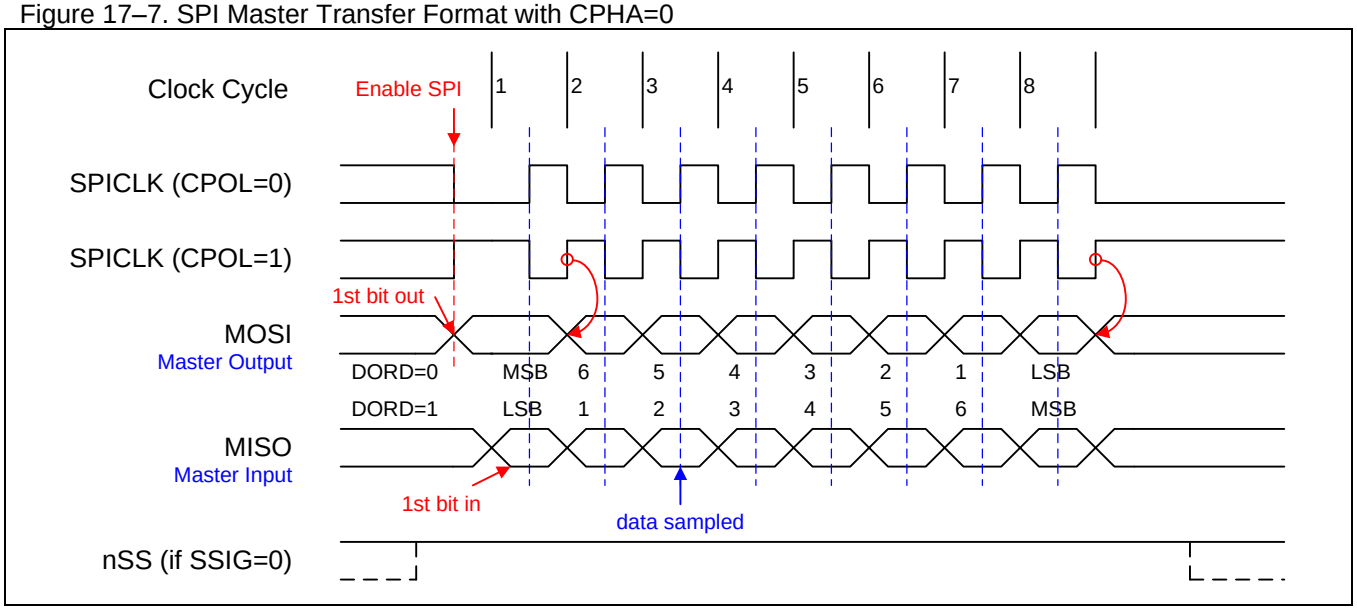
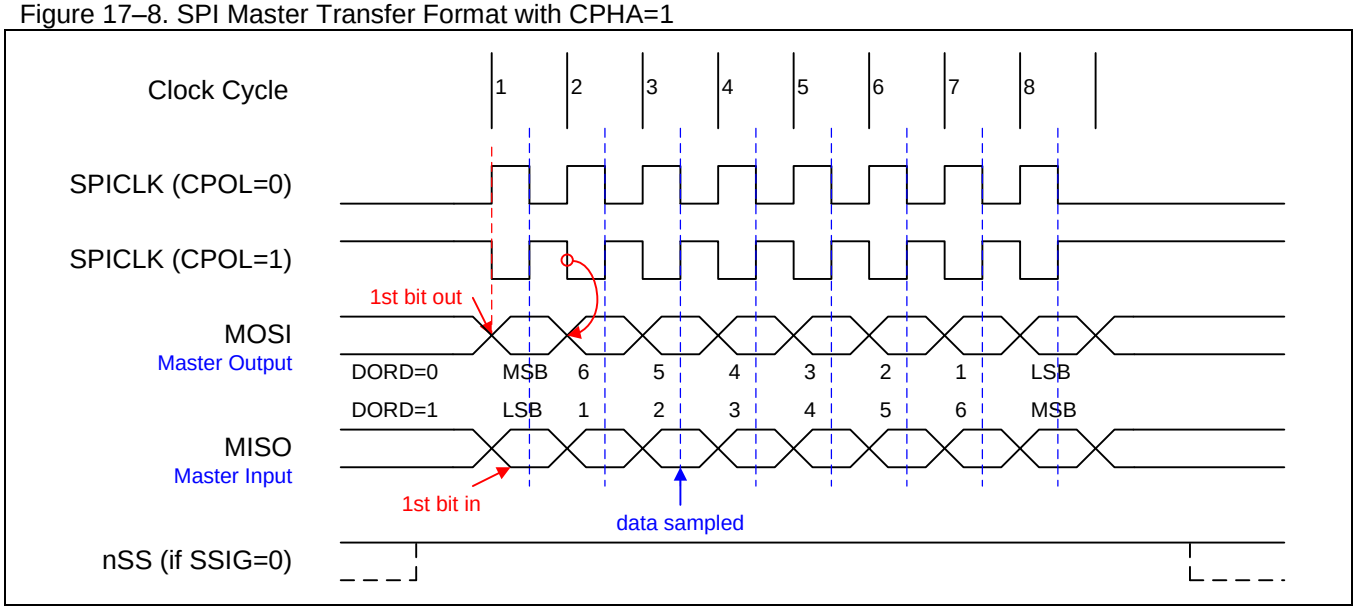


Figure 17–8. SPI Master Transfer Format with CPHA=1



17.4. SPI Register

The following special function registers are related to the SPI operation:

SPCON: SPI Control Register

SFR Address = 0x85 RESET= 0000-0100

7	6	5	4	3	2	1	0
SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: SSIG, nSS is ignored.
0: The nSS pin decides whether the device is a master or slave.
1: MSTR decides whether the device is a master or slave.

Bit 6: SPEN, SPI enable.
0: The SPI interface is disabled and all SPI pins will be general-purpose I/O ports.
1: The SPI is enabled.

Bit 5: DORD, SPI data order.
0: The MSB of the data byte is transmitted first.
1: The LSB of the data byte is transmitted first.

Bit 4: MSTR, Master/Slave mode select
0: Selects slave SPI mode.
1: Selects master SPI mode.

Bit 3: CPOL, SPI clock polarity select
0: SPICLK is low when Idle. The leading edge of SPICLK is the rising edge and the trailing edge is the falling edge.
1: SPICLK is high when Idle. The leading edge of SPICLK is the falling edge and the trailing edge is the rising edge.

Bit 2: CPHA, SPI clock phase select
0: Data is driven when /SS pin is low (SSIG=0) and changes on the trailing edge of SPICLK. Data is sampled on the leading edge of SPICLK.
1: Data is driven on the leading edge of SPICLK, and is sampled on the trailing edge.
(Note: If SSIG=1, CPHA must not be 1, otherwise the operation is not defined.)

Bit 1~0: SPR1-SPR0, SPI clock rate select (in master mode)
00: SYSCLK/4
01: SYSCLK/16
10: SYSCLK/64
11: SYSCLK/128 (Where, SYSCLK is the system clock.)

SPSTAT: SPI Status Register

SFR Address = 0x84 RESET= 00XX-XXXX

7	6	5	4	3	2	1	0
SPIF	WCOL	--	--	--	--	--	--
R/W	**R/W**	**R**	**R**	**R**	**R**	**R**	**R**

Bit 7: SPIF, SPI transfer completion flag
0: **The SPIF is cleared in software by writing '1' to this bit.** Software writing ":0" is no operation.
1: When a serial transfer finishes, the SPIF bit is set and an interrupt is generated if SPI interrupt is enabled. If nSS pin is driven low when SPI is in master mode with SSIG=0, SPIF will also be set to signal the "mode change".

Bit 6: WCOL, SPI write collision flag.
0: The WCOL flag is cleared in software by writing '1' to this bit. Software writing ":0" is no operation.
1: The WCOL bit is set if the SPI data register, SPDAT, is written during a data transfer (see Section 15.2.4: Write Collision).

Bit 5~0: Reserved. Software must write “0” on these bits when SPSTAT is written.

SPDAT: SPI Data Register

SFR Address = 0x86 RESET= 0000-0000

7	6	5	4	3	2	1	0
(MSB)							(LSB)
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

SPDAT has two physical buffers for writing to and reading from during transmit and receive, respectively.

17.5. SPI Sample Code

(1). Required Function: SPI Master Read and Write, sample data at rising edge and clock leading edge is rising.

Assembly Code Example:

```
CPHA            EQU             04h
CPOL            EQU             08h
MSTR            EQU             10h
SPEN            EQU             40h
SSIG            EQU             80h
SPIF            EQU             80h

Initial_SPI:                                    ;initial SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR)         ;enable SPI and Master mode
   RET

SPI_Write:
   MOV    SPIDAT, R7                            ;write arg R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                     ;wait transfer finishes
   ANL    SPISTAT, #(0FFh - SPIF)               ;clear SPI interrupt flag
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                         ;trigger SPI read
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                      ;wait read finishes
   ANL    SPISTAT, #(0FFh - SPIF)               ;clear SPI interrupt flag
   MOV    A, SPIDAT                             ;move read data to accumulator
   RET
```

C Code Example:

```
#define  CPHA                0x04
#define  CPOL                0x08
#define  MSTR                0x10
#define  SPEN                0x40
#define  SSIG                0x80
#define  SPIF                0x80

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR);              // enable SPI and Master mode
}

void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                //write arg
   while(!(SPISTAT & SPIF));                    //wait transfer finishes
   SPISTAT &= ~SPIF;                            //clear SPI interrupt flag
}

unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                               //trigger SPI read
   while(!SPISTAT & SPIF);                      //wait transfer finishes
   SPISTAT &= ~SPIF;                            //clear SPI interrupt flag
   return SPIDAT;
}
```

(2). Required Function: SPI Master Read and Write, sample data at rising edge and clock leading edge is falling.

Assembly Code Example:

```
CPHA            EQU             04h
CPOL            EQU             08h
MSTR            EQU             10h
SPEN            EQU             40h
SSIG            EQU             80h
SPIF            EQU             80h

Initial_SPI:                                        ;initial SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR + CPOL)      ;enable SPI and Master mode
   RET

SPI_Write:
   MOV    SPIDAT, R7                                ;write arg R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                         ;wait transfer finishes
   ANL    SPISTAT, #(0FFh - SPIF)                   ;clear SPI interrupt flag
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                             ;trigger SPI read
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                          ;wait read finishes
   ANL    SPISTAT, #(0FFh - SPIF)                   ;clear SPI interrupt flag
   MOV    A, SPIDAT                                 ;move read data to accumulator
   RET
```

C Code Example:

```
#define  CPHA                0x04
#define  CPOL                0x08
#define  MSTR                0x10
#define  SPEN                0x40
#define  SSIG                0x80
#define  SPIF                0x80

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR | CPOL);           // enable SPI and Master mode
}

void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                    //write arg
   while(!(SPISTAT & SPIF));                        //wait transfer finishes
   SPISTAT &= ~SPIF;                                //clear SPI interrupt flag
}

unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                   //trigger SPI read
   while(!SPISTAT & SPIF);                          //wait transfer finishes
   SPISTAT &= ~SPIF;                                //clear SPI interrupt flag
   return SPIDAT;
}
```

(3). Required Function: SPI Master Read and Write, sample data at falling edge and clock leading edge is rising.

Assembly Code Example:

```
CPHA            EQU             04h
CPOL            EQU             08h
MSTR            EQU             10h
SPEN            EQU             40h
SSIG            EQU             80h
SPIF            EQU             80h

Initial_SPI:                                        ;initial SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR + CPHA)      ;enable SPI and Master mode
   RET

SPI_Write:
   MOV    SPIDAT, R7                                ;write arg R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                         ;wait transfer finishes
   ANL    SPISTAT, #(0FFh - SPIF)                   ;clear SPI interrupt flag
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                             ;trigger SPI read
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                          ;wait read finishes
   ANL    SPISTAT, #(0FFh - SPIF)                   ;clear SPI interrupt flag
   MOV    A, SPIDAT                                 ;move read data to accumulator
   RET
```

C Code Example:

```
#define  CPHA            0x04
#define  CPOL            0x08
#define  MSTR            0x10
#define  SPEN            0x40
#define  SSIG            0x80
#define  SPIF            0x80

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR | CPHA);           // enable SPI and Master mode
}

void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                    //write arg
   while(!(SPISTAT & SPIF));                        //wait transfer finishes
   SPISTAT &= ~SPIF;                                //clear SPI interrupt flag
}

unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                   //trigger SPI read
   while(!SPISTAT & SPIF);                          //wait transfer finishes
   SPISTAT &= ~SPIF;                                //clear SPI interrupt flag
   return SPIDAT;
}
```

(4). Required Function: SPI Master Read and Write, sample data at falling edge and clock leading edge is falling.

Assembly Code Example:

```
CPHA            EQU             04h
CPOL            EQU             08h
MSTR            EQU             10h
SPEN            EQU             40h
SSIG            EQU             80h
SPIF            EQU             80h

Initial_SPI:                                            ;initial SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR + CPOL + CPHA)   ;enable SPI and Master mode
   RET

SPI_Write:
   MOV    SPIDAT, R7                                    ;write arg R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                             ;wait transfer finishes
   ANL    SPISTAT, #(0FFh - SPIF)                       ;clear SPI interrupt flag
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                                 ;trigger SPI read
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                              ;wait read finishes
   ANL    SPISTAT, #(0FFh - SPIF)                       ;clear SPI interrupt flag
   MOV    A, SPIDAT                                     ;move read data to accumulator
   RET
```

C Code Example:

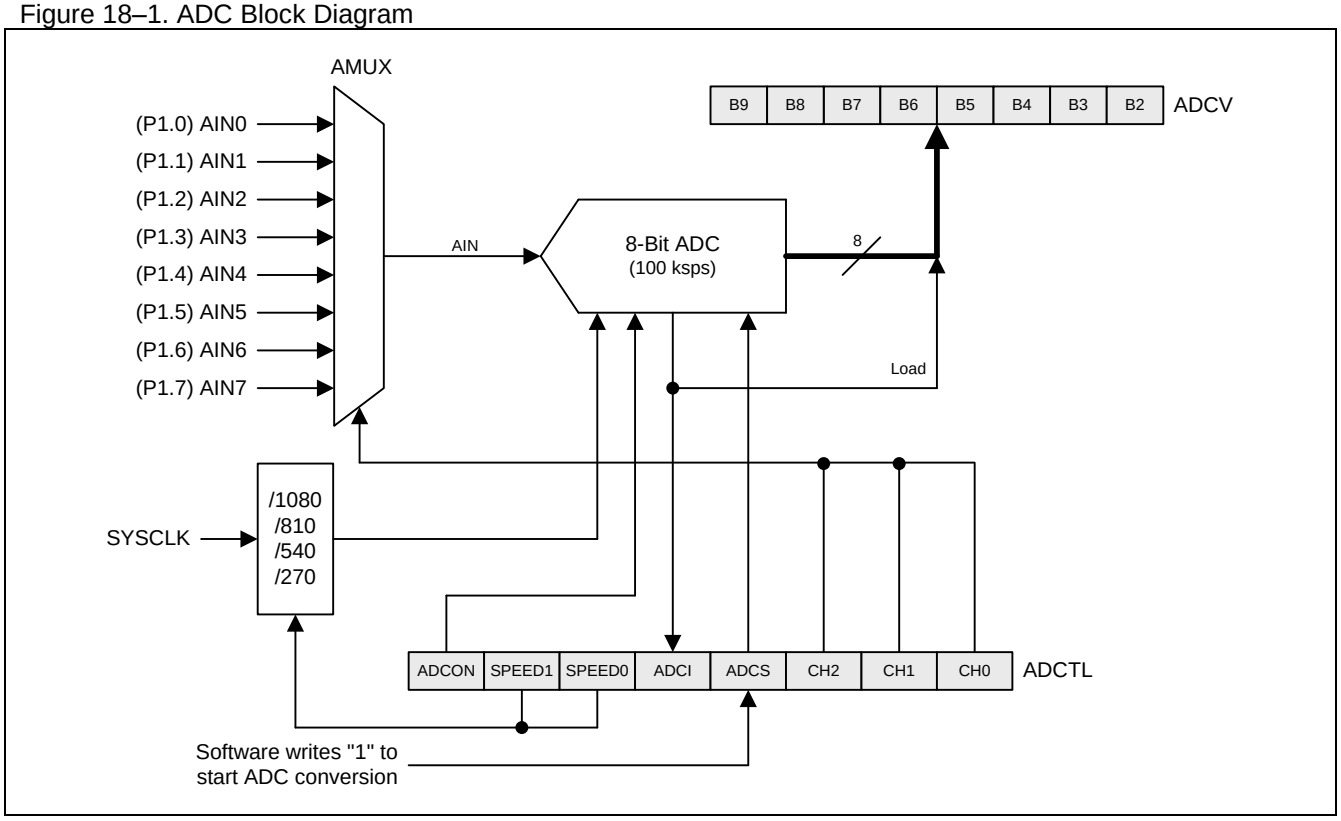
```
#define  CPHA            0x04
#define  CPOL            0x08
#define  MSTR            0x10
#define  SPEN            0x40
#define  SSIG            0x80
#define  SPIF            0x80

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR | CPOL | CPHA);       // enable SPI and Master mode
}

void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                        //write arg
   while(!(SPISTAT & SPIF));                            //wait transfer finishes
   SPISTAT &= ~SPIF;                                    //clear SPI interrupt flag
}

unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                       //trigger SPI read
   while(!SPISTAT & SPIF);                              //wait transfer finishes
   SPISTAT &= ~SPIF;                                    //clear SPI interrupt flag
   return SPIDAT;
}
```

18. 8-Bit ADC

The ADC subsystem for the **MPC82E/L52** consists of an analog multiplexer (AMUX), and a **100** ksps, **8**-bit successive-approximation-register ADC. The AMUX can be configured via the Special Function Registers shown in Figure 18–1. ADC operates in Single-ended modes, and may be configured to measure any of the pins on Port 1. The ADC subsystem is enabled only when the ADCON bit in the ADC Control register (ADCTL) is set to logic 1. The ADC subsystem is in low power shutdown when this bit is logic 0.

18.1. ADC Structure

Figure 18–1. ADC Block Diagram



18.2. ADC Operation

ADC has a maximum conversion speed of 100 ksps. The ADC conversion clock is a divided version of the system clock, determined by the SPEED1~0 bits in the ADCTL register.

After the conversion is complete (ADCI is high), the conversion result can be found in the ADC Result Registers (ADCV). For single ended conversion, the result is

$$\text{ADC Result} = \frac{V_{IN} \times 256}{\text{VDD Voltage}}$$

18.2.1. ADC Input Channels

The analog multiplexer (AMUX) selects the inputs to the ADC, allowing any of the pins on Port 1 to be measured in single-ended mode. The ADC input channels are configured and selected by CHS.2~0 in the ADCTL register as described in Figure 18–1. The selected pin is measured with respect to GND.

18.2.2. Starting a Conversion

Prior to using the ADC function, the user should:

1) Turn on the ADC hardware by setting the ADCON bit,
2) Configure the ADC input clock by bits SPEED1 and SPEED0,
3) Select the analog input channel by bits CHS2, CHS1 and CHS0,
4) Configure the selected input (shared with P1) to the Input-Only mode by P1M0 and P1M1 registers, and

Now, user can set the ADCS bit to start the A-to-D conversion. The conversion time is controlled by bits SPEED1 and SPEED0. Once the conversion is completed, the hardware will automatically clear the ADCS bit, set the interrupt flag ADCI and load the **8** bits of conversion result into ADCV simultaneously.

As described above, the interrupt flag ADCI, when set by hardware, shows a completed conversion. Thus two ways may be used to check if the conversion is completed: (1) Always polling the interrupt flag ADCI by software; (2) Enable the ADC interrupt by setting bits EADCI (in AUXR register), ESPI_ADC (in IE register) and EA (in IE register), and then the CPU will jump into its Interrupt Service Routine when the conversion is completed. Regardless of (1) or (2), the ADCI flag should be cleared by software before next conversion.

18.2.3. ADC Conversion Time

The user can select the appropriate conversion speed according to the frequency of the analog input signal. User can configure the SPEED1~0 in ADCTL to specify the conversion rate. For example, if SYSCLK =25MHz and the SPEED[1:0] = SYSCLK/270 is selected, then the frequency of the analog input should be no more than 92.6KHz to maintain the conversion accuracy. (Conversion rate = 25MHz/270 = 92.6KHz.)

18.2.4. I/O Pins Used with ADC Function

The analog input pins used for the A/D converters also have its I/O port ‘s digital input and output function. In order to give the proper analog performance, a pin that is being used with the ADC should have its digital output as disabled. It is done by putting the port pin into the input-only mode as described in the Section “12 Configurable I/O Ports”.

18.2.5. Idle and Power-Down Mode

In Power-Down mode, the ADC does not function. If the A/D is turned on, it will consume a little power. So, power consumption can be reduced by turning off the ADC hardware (ADCON=0) before entering Idle mode and Power-Down mode.

18.3. ADC Register

ADCTL: ADC Control Register

SFR Address = 0xC5 RESET = 0000-0000

7	6	5	4	3	2	1	0
ADCON	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: ADCON, ADC Enable.
0: Clear to turn off the ADC block.
1: Set to turn on the ADC block. At least 5us ADC enabled time is required before set ADCS.

Bit 6~5: SPEED1 and SPEED0, ADC conversion speed control.

SPEED[1:0]	ADC Clock Selection
0 0	SYSCLK/1080
0 1	SYSCLK/810
1 0	SYSCLK/540
1 1	SYSCLK/270

The recommended ADC clock is no more than 12MHz.

Bit 4: ADCI, ADC Interrupt Flag.
0: The flag must be cleared by software.
1: This flag is set when an A/D conversion is completed. An interrupt is invoked if it is enabled.

Bit 3: ADCS. ADC Start of conversion.
0: ADCS cannot be cleared by software.
1: Setting this bit by software starts an A/D conversion. On completion of the conversion, the ADC hardware will clear ADCS and set the ADCI. A new conversion may not be started while either ADCS or ADCI is high.

Bit 2~0: CHS2 ~ CHS1, Input Channel Selection for ADC analog multiplexer.

CHS[2:0]	Selected Channel
0 0 0	AIN0 (P1.0)
0 0 1	AIN1 (P1.1)
0 1 0	AIN2 (P1.2)
0 1 1	AIN3 (P1.3)
1 0 0	AIN4 (P1.4)
1 0 1	AIN5 (P1.5)
1 1 0	AIN6 (P1.6)
1 1 1	AIN7 (P1.7)

ADCV: ADC Result Register

SFR Address = 0xC6 RESET = xxxx-xxxx

7	6	5	4	3	2	1	0
ADCV.7	ADCV.6	ADCV.5	ADCV.4	ADCV.3	ADCV.2	ADCV.1	ADCV.0
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**

In **MPC82E/L52**, conversion codes are represented as **8**-bit unsigned integers. Inputs are measured from '0' to VDD x 255/256. Example codes are shown below.

Input Voltage	ADCV
VDD x 255/256	0xFF
VDD x 128/256	0x80
VDD x 64/256	0x40
0	0x00

18.4. ADC Sample Code

(1). Required Function: ADC sample code for SYSCLK=24MHz, transfer analog input on P1.0/P1.1/P1.2 with SPEED[1:0]=SYSCLK/270 for 88.9KHz conversion rate.

Assembly Code Example:

```
CHS0            EQU             01h
CHS1            EQU             02h
ADCS            EQU             08h
ADCI            EQU             10h
SPEED0          EQU             20h
SPEED1          EQU             40h
ADCON           EQU             80h

INITIAL_ADC_PIN:
        ORL     P1M0, #00000111B                ; P1.0, P1.1, P1.2 = input only
        ANL     P1M1,#11111000B

        MOV     ADCTL,#ADCON                    ; Enable ADC block
        ; delay 5us
        ; call ....

Get_P10:
        MOV     ADCTL, #(ADCON + SPEED1 + SPEED0)
                                                ; Enable ADC block & start conversion
                                                ; Speed at 114.3k @ 24MHz, select P1.0 for ADC input pin
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     A, ADCTL                        ; check ready?
        JNB     ACC.4,$-3
        ANL     ADCTL,#(0FFh - ADCI - ADCS)     ; clear ADCI & ADCS
        MOV     AIN0_data_V,ADCV                ; reserve P1.0 ADC data
        ; to do ...

Get_P11:
        MOV     ADCTL,#(ADCON + SPEED1 + SPEED0 + CHS0)  ; select P1.1
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     A, ADCTL                        ; check ready?
        JNB     ACC.4,$-3
        ANL     ADCTL,# (0FFh - ADCI - ADCS)    ; clear ADCI & ADCS
        MOV     AIN1_data_V,ADCV
    ; to do ...

Get_P12:
        MOV     ADCTL,#(ADCON + SPEED1 + SPEED0 + CHS1)  ; select P1.2
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     ACC,ADCTL                       ; check ready?
        JNB     ACC.4,$-3
        ANL     ADCTL,# (0FFh - ADCI - ADCS)    ; clear ADCI & ADCS
        MOV     AIN2_data_V,ADCV


    ; to do ...

        RET
```

C Code Example:

```
#define  CHS0           0x01
#define  CHS1           0x02
#define  ADCS           0x08
#define  ADCI           0x10
#define  SPEED0         0x20
#define  SPEED1         0x40
```

```
#define  ADCON                  0x80


void main(void)
{
        unsigned char AIN0_data_V, AIN1_data_V, AIN2_data_V;

        P1M0 |= 0x07;                                   // P1.0, P1.1, P1.2 = input only
        P1M1 &= ~0x07;

        ADCTL = ADCON;                                  // Enable ADC block
        // delay 5us
        // ...


        // select P1.0
        ADCTL = (ADCON | SPEED1 | SPEED0);
                                                        // Enable ADC block & start conversion
                                                // Speed at 114.3k @ 24MHz, select P1.0 for ADC input pin
    Delay_5us();
    ADCTL |= ADCS;


        while ((ADCTL & ADCI) == 0x00);                 //wait for complete
        ADCTL &= ~(ADCI | ADCS);
        AIN0_data_V = ADCV;

        // to do ...

        // select P1.1
        ADCTL = (ADCON | SPEED1 | SPEED0 | CHS0);       // select P1.1
    Delay_5us();
    ADCTL |= ADCS;


        while ((ADCTL & ADCI) == 0x00);                 //wait for complete
        ADCTL &= ~(ADCI | ADCS);
        AIN1_data_V = ADCV;

        // to do ...

        // select P1.2
        ADCTL = (ADCON | SPEED1 | SPEED0 | CHS1);       // select P1.2
    Delay_5us();
    ADCTL |= ADCS;


        while ((ADCTL & ADCI) == 0x00);                 //wait for complete
        ADCTL &= ~(ADCI | ADCS);
        AIN2_data_V = ADCV;

        // to do ...

        while (1);

}
```

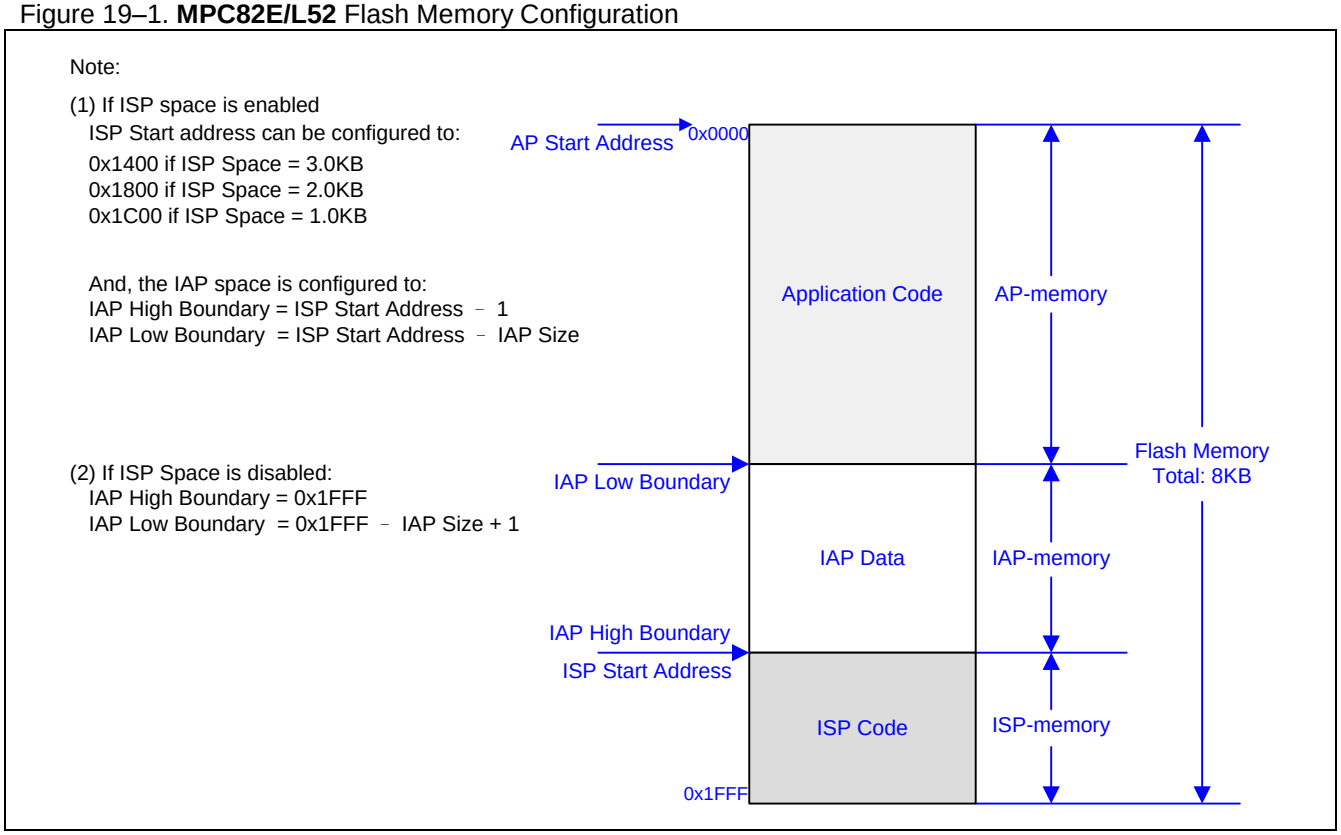
19. ISP and IAP

The flash memory of **MPC82E/L52** is partitioned into AP-memory, IAP-memory and ISP-memory. AP-memory is used to store user's application program; IAP-memory is used to store the non-volatile application data; and, ISP-memory is used to store the boot loader program for In-System Programming. When MCU is running in ISP region, MCU could modify the AP and IAP memory for software upgraded. If MCU is running in AP region, software could only modify the IAP memory for storage data updated.

19.1. MPC82E/L52 Flash Memory Configuration

There are total **8K** bytes of Flash Memory in **MPC82E/L52** and Figure 19–1 shows the device flash configuration of **MPC82E/L52**. The ISP-memory can be configured as disabled or up to **3K** bytes space by hardware option. The flash size of IAP memory is located between the IAP low boundary and IAP high boundary. The IAP low boundary is defined by the value of IAPLB register. The IAP high boundary is associated with ISP start address which decides ISP memory size by hardware option. The IAPLB register value is configured by hardware option. All of the AP, IAP and ISP memory are shared the total **8K** bytes flash memory.

Figure 19–1. **MPC82E/L52** Flash Memory Configuration



Note:

*In default, the **MPC82E/L52** that Megawin shipped had configured the flash memory for **1K ISP, 1K IAP** and Lock enabled. The **1K** ISP region is inserted Megawin proprietary ISP code to perform In-System-Programming through Megawin 1-Line ISP protocol. The IAP size can be re-configured by writer to modify IAPLB.*

19.2. MPC82E/L52 Flash Access in ISP/IAP

There are 3 flash access modes are provided in **MPC82E/L52** for ISP and IAP application: page erase mode, program mode and read mode. MCU software uses these three modes to update new data into flash storage and get flash content. This section shows the flow chart and demo code for the various flash modes.

19.2.1. ISP/IAP Flash Page Erase Mode

The any bit in flash data of **MPC82E/L52** only can be programmed to “0”. If user would like to write a “1” into flash data, the flash erase is necessary. But the flash erase in **MPC82E/L52** ISP/IAP operation only support “page erase” mode, a page erase will write all data bits to “1” in one page. There are 512 bytes in one page of **MPC82E/L52** and the page start address is aligned to A8~A0 = 0x000. The targeted flash address is defined in IFADRH and IFADRL. So, in flash page erase mode, the IFADRH.0(A8) and IFADRL.7~0(A7~A0) must be written to “0” for right page address selection. Figure 19–2 shows the flash page erase flow in ISPIAP operation.

Figure 19–2. ISP/IAP Page Erase Flow

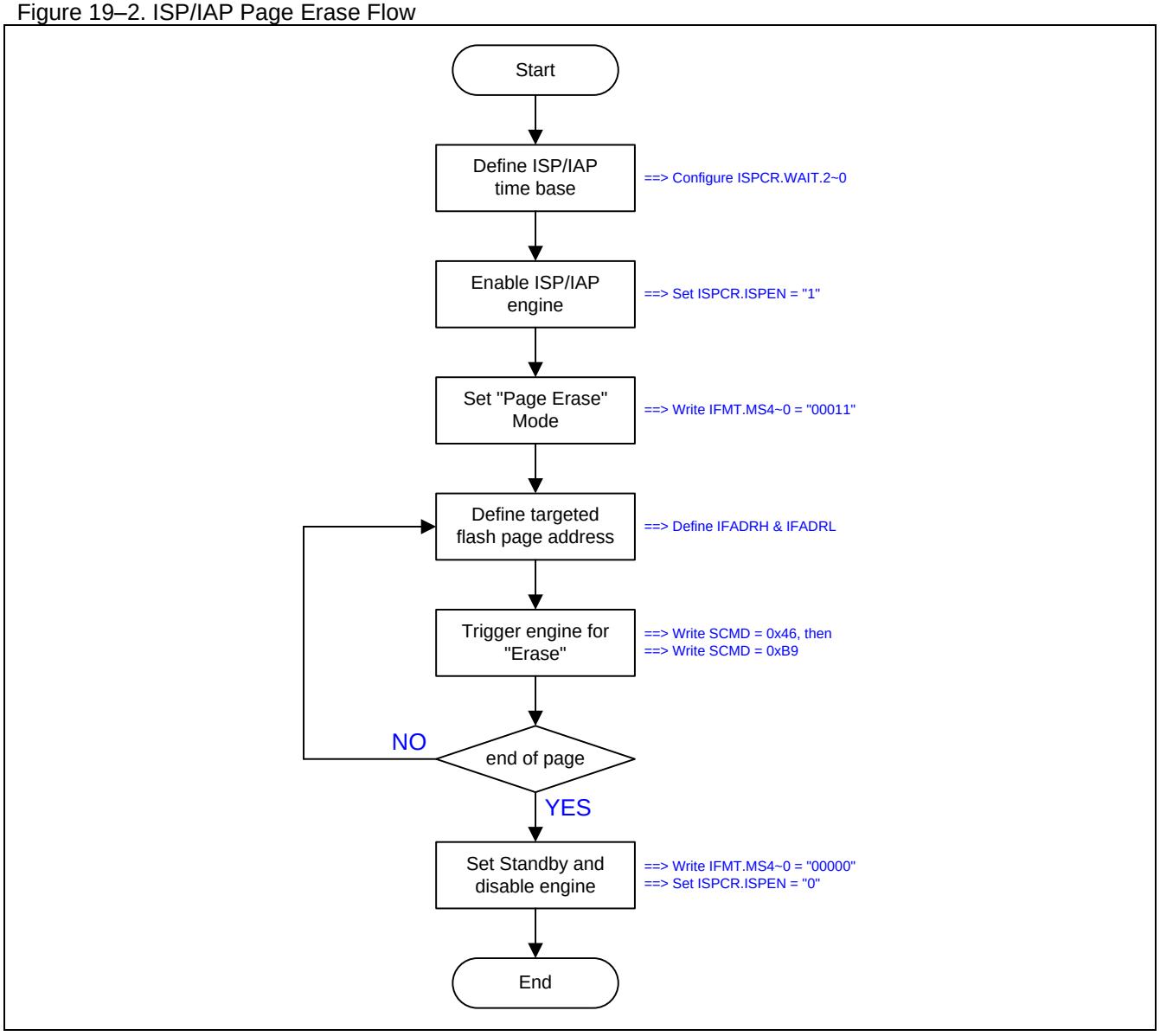


Figure 19–3 shows the demo code of the ISP/IAP page erase operation.

Figure 19–3. Assemble code example for ISP/IAP Page Erase

```
MOV    ISPCR,#00010111b ; XCKS4~0 = decimal 23 when OSCin = 24MHz

MOV    ISPCR,#10000000b ; ISPCR.7 = 1, enable ISP

MOV    IFMT,#03h        ; select Page Erase Mode

MOV    IFADRH,??        ; fill [IFADRH,IFADRL] with page address
MOV    IFADRL,??        ;

MOV    SCMD,#46h        ; trigger ISP/IAP processing
MOV    SCMD,#0B9h       ;

;Now, MCU will halt here until processing completed

MOV    IFMT,#00h        ; select Standby Mode
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, disable ISP
```

19.2.2. ISP/IAP Flash Program Mode

The “program” mode of **MPC82E/L52** provides the byte write operation into flash memory for new data updated. The IFADRH and IFADRL point to the physical flash byte address. IFD stores the content which will be programmed into the flash. Figure 19–4 shows the flash byte program flow in ISP/IAP operation.

Figure 19–4. ISP/IAP byte Program Flow

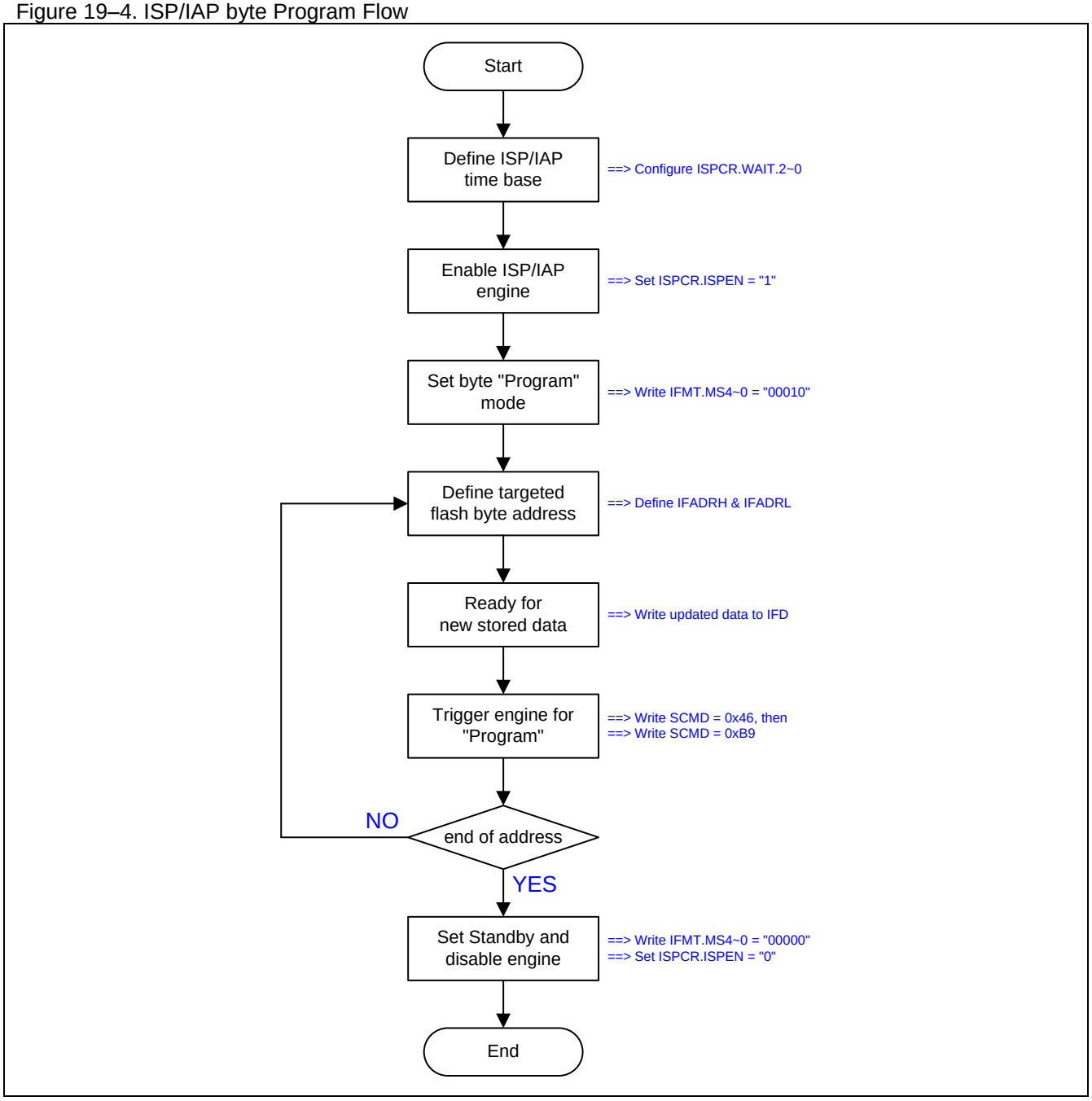


Figure 19–5 shows the demo code of the ISP/IAP byte program operation.

Figure 19–5. Assemble code example for ISP/IAP byte Program

```
MOV    ISPCR,#00010111b ; XCKS4~0 = decimal 23 when OSCin = 24MHz

MOV    ISPCR,#10000011b ; ISPCR.7=1, enable ISP

MOV    IFMT,#02h        ; select Program Mode

MOV    IFADRH,??        ; fill [IFADRH,IFADRL] with byte address
MOV    IFADRL,??        ;

MOV    IFD,??           ; fill IFD with the data to be programmed

MOV    SCMD,#46h        ;trigger ISP/IAP processing
MOV    SCMD,#0B9h       ;

;Now, MCU will halt here until processing completed

MOV    IFMT,#00h        ; select Standby Mode
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, disable ISP
```

19.2.3. ISP/IAP Flash Read Mode

The “read” mode of **MPC82E/L52** provides the byte read operation from flash memory to get the stored data. The IFADRH and IFADRL point to the physical flash byte address. IFD stores the data which is read from the flash content. It is recommended to verify the flash data by read mode after data programmed or page erase.
Figure 19–6 shows the flash byte read flow in ISP/IAP operation.

Figure 19–6. ISP/IAP byte Read Flow

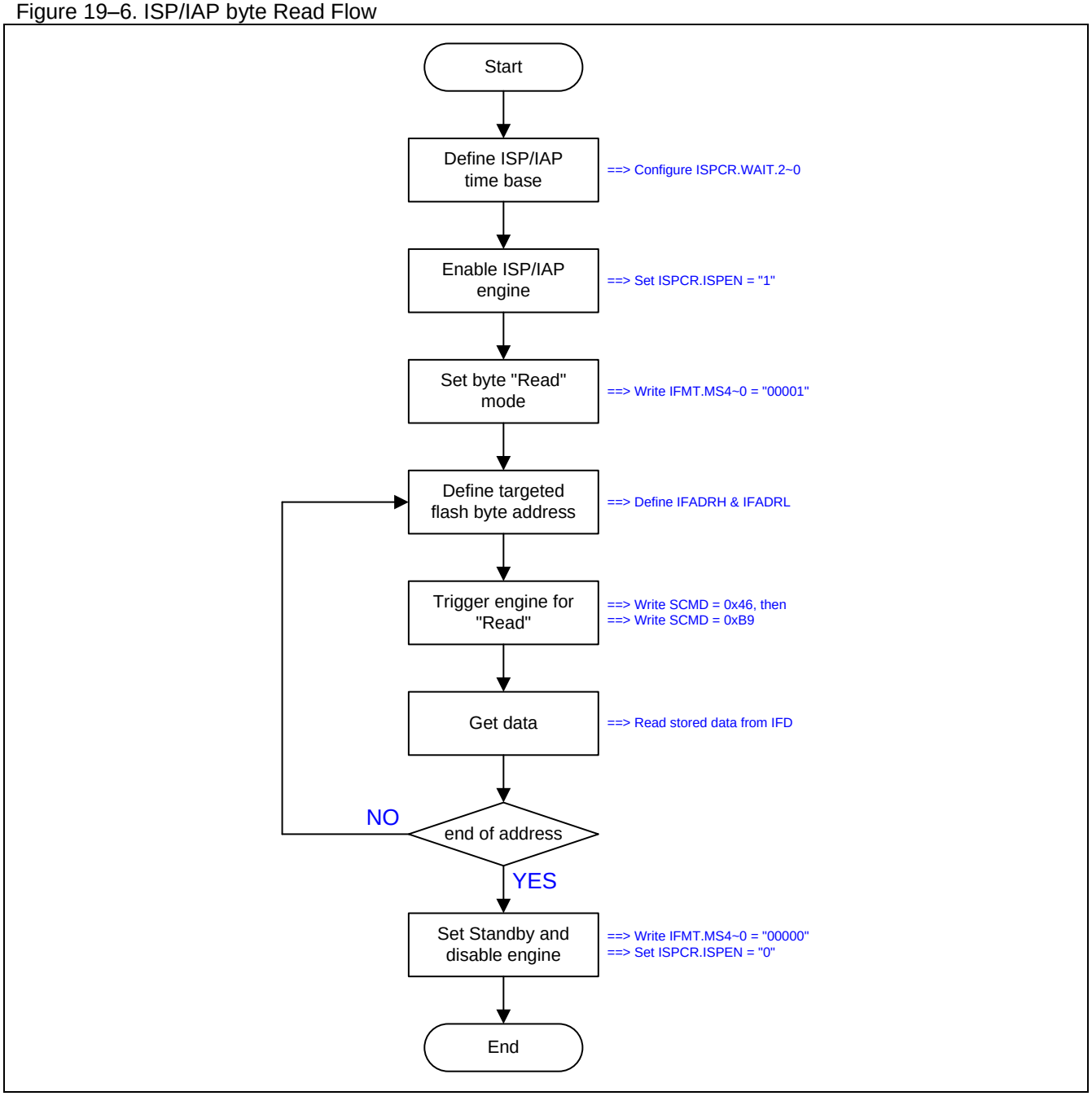


Figure 19–7 shows the demo code of the ISP/IAP byte read operation.

Figure 19–7. Assemble code example for ISP/IAP byte Read

```
MOV    ISPCR,#00010111b ; XCKS4~0 = decimal 23 when OSCin = 24MHz

MOV    ISPCR,#10000011b ; ISPCR.7=1, enable ISP

MOV    IFMT,#01h        ; select Read Mode

MOV    IFADRH,??        ; fill [IFADRH,IFADRL] with byte address
MOV    IFADRL,??        ;

MOV    SCMD,#46h        ; trigger ISP/IAP processing
MOV    SCMD,#0B9h       ;

;Now, MCU will halt here until processing completed

MOV    A,IFD            ; now, the read data exists in IFD

MOV    IFMT,#00h        ; select Standby Mode
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, disable ISP
```

19.3. ISP Operation

ISP means In-System-Programming which makes it possible to update the user's application program (in AP-memory) and non-volatile application data (in IAP-memory) without removing the MCU chip from the actual end product. This useful capability makes a wide range of field-update applications possible. The ISP mode is used in the *loader program* to program both the AP-memory and IAP-memory.

Note:

(1) Before using the ISP feature, the user should configure an ISP-memory space and pre-program the ISP code (loader program) into the ISP-memory by a universal Writer/Programmer or Megawin proprietary Writer/Programmer.

(2) ISP code in the ISP-memory can only program the AP-memory and IAP-memory.

After ISP operation has been finished, software writes "001" on ISPCR.7 ~ ISPCR.5 which triggers an software RESET and makes CPU reboot into application program memory (AP-memory) on the address 0x0000.

As we have known, the purpose of the ISP code is to program both AP-memory and IAP-memory. Therefore, ***the MCU must boot from the ISP-memory in order to execute the ISP code***. There are two methods to implement In-System Programming according to how the MCU boots from the ISP-memory.

19.3.1. Hardware approached ISP

To make the MCU directly boot from the ISP-memory when it is just powered on, the MCU's hardware options *HWBS* and *ISP Memory* must be enabled. The ISP entrance method by hardware option is named hardware approached. Once *HWBS* and *ISP Memory* are enabled, the MCU will always boot from the ISP-memory to execute the ISP code (loader program) when it is just powered on. The first thing the ISP code should do is to check if there is an ISP request. If there is no ISP requested, the ISP code should trigger a software reset (setting ISPCR.7~5 to "101" simultaneously) to make the MCU re-boot from the AP-memory to run the user's application program.

The following sample code describes how to leave hardware approach ISP memory to launch AP software:

Code example for ISP reset to AP or AP reset to AP

Assembly Code Example:

```
SWRST   EQU     20h
SWBS    EQU     40h

        ANL     ISPCR, #(0FFh - SWBS)   ;clear SWBS
        ORL     ISPCR, #SWRST                   ;trigger software reset
```

C Code Example:

```
#define SWRST   0x20
#define SWBS    0x40

        ISPCR &= ~SWBS;                 // clear SWBS
        ISPCR |= SWRST;                 // trigger software reset
```

19.3.2. Software approached ISP

The software approached ISP to make the MCU boot from the ISP-memory is to trigger a software reset while the MCU is running in the AP-memory. In this case, neither HWBS nor HWBS2 is enabled. The only way for the MCU to boot from the ISP-memory is to trigger a software reset, setting ISPCR.7~5 to “111” simultaneously, when running in the AP-memory. Note: the ISP memory must be configured a valid space by hardware option to reserve ISP mode for software approached ISP application.

The following sample code describes how to leave AP memory to launch software approached ISP:

Code example for AP reset to ISP

Assembly Code Example:

```
SWRST   EQU     20h
SWBS    EQU     40h

        ORL     ISPCR, #( SWBS + SWRST)     ;set SWBS and trigger software reset
```

C Code Example:

```
#define SWRST   0x20
#define SWBS    0x40

        ISPCR |=(SWBS + SWRST);     // set SWBS and trigger software reset
```

19.3.3. Notes for ISP

Developing of the ISP Code

Although the ISP code is programmed in the ISP-memory that has an *ISP Start Address* in the MCU's Flash (see Figure 19–1), it doesn't mean you need to put this offset (= *ISP Start Address*) in your source code. The code offset is automatically manipulated by the hardware. User just needs to develop it like an application program in the AP-memory.

Interrupts during ISP

After triggering the ISP/IAP flash processing, the MCU will halt for a while for internal ISP processing until the processing is completed. At this time, the interrupt will queue up for being serviced if the interrupt is enabled previously. Once the processing is completed, the MCU continues running and the interrupts in the queue will be serviced immediately if the interrupt flag is still active. The user, however, should be aware of the following:

(1) Any interrupt can not be in-time serviced when the MCU halts for ISP processing.
(2) The low/high-level triggered external interrupts, INTx, should keep activated until the ISP is completed, or they will be neglected.

ISP and Idle mode

MPC82E/L52 does not make use of idle-mode to perform ISP function. Instead, it freezes CPU running to release the flash memory for ISP/IAP engine operating. Once ISP/IAP operation finished, CPU will be resumed and advanced to the instruction which follows the previous instruction that invokes ISP/AP activity.

Accessing Destination of ISP

As mentioned previously, the ISP is used to program both the AP-memory and the IAP-memory. Once the accessing destination address is beyond that of the last byte of the IAP-memory, the hardware will automatically neglect the triggering of ISP processing. That is the triggering of ISP is invalid and the hardware does nothing.

Flash Endurance for ISP

The endurance of the embedded Flash is 20,000 erase/write cycles, that is to say, the erase-then-write cycles shouldn't exceed 20,000 times. Thus the user should pay attention to it in the application which needs to frequently update the AP-memory and IAP-memory.

19.3.4. Default ISP Code in MPC82E/L52

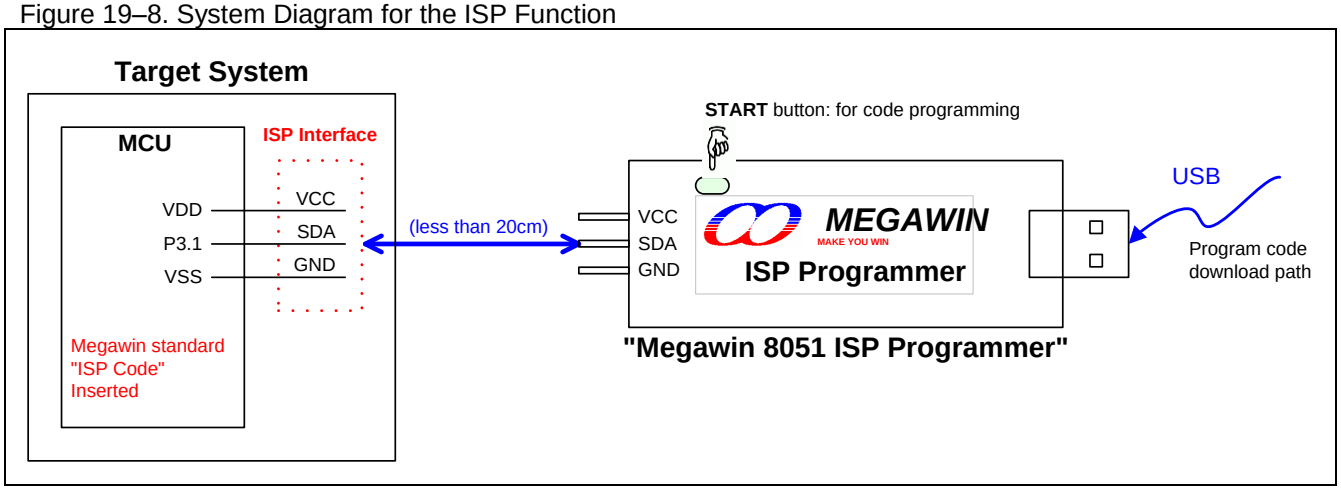
Although the user may design self ISP, MPC82E/L52 has been inserted the Megawin proprietary ISP code before shipping. Megawin provides a tool, Megawin 8051 ISP Programmer, to perform In-System-Programming for AP program code upgraded. This section shows a brief description for the Megawin ISP behavior.

Features

- The standard 'ISP code' is pre-programmed in factory before shipping.
- Only one port pin (P3.1) used for the ISP interface.
- Operation independent of the oscillator frequency.
- Capable of stand-alone working without host's intervention.

The above valuable features make the ISP Programmer very friendly to the user. Particularly, it is capable of stand-alone working after the programming data is downloaded. This is especially useful in the field without a PC. The system diagram of the "Megawin 8051 ISP Programmer" is shown in Figure 19–8. Only three pins are used for the ISP interface: the ***DTA*** line transmits the programming data from the ISP Programmer to the target MCU; the ***VCC*** & ***GND*** are the power supply entry of the ISP Programmer. Section "22.4 ISP Interface Circuit" describes the ISP interface circuit of the MCU applied in user system. The USB connector of the ISP tool can be directly plugged into the PC's USB port to download the programming data from PC to the ISP Programmer.

Figure 19–8. System Diagram for the ISP Function



19.4. In-Application-Programming (IAP)

The **MPC82E/L52** has built a function as *In Application Programmable* (IAP), which allows some region in the Flash memory to be used as non-volatile data storage while the application program is running. This useful feature can be applied to the application where the data must be kept after power off. Thus, there is no need to use an external serial EEPROM (such as 93C46, 24C01, .., and so on) for saving the non-volatile data.

In fact, the operating of IAP is the same as that of ISP except the Flash range to be programmed is different. The programmable Flash range for ISP operating is located within the AP and IAP memory, while the range for IAP operating is only located within the configured IAP-memory.

Note:

*(1) For **MPC82E/L52** IAP feature, the software should specify an IAP-memory space by writing IAPLB in Page-P SFR space. The IAP-memory space can be also configured by a universal Writer/Programmer or Megawin proprietary Writer/Programmer which configuration is corresponding to IAPLB initial value.*

*(2) The program code to execute IAP is located in the AP-memory and **just only** program IAP-memory **not** ISP-memory.*

19.4.1. IAP-memory Boundary/Range

If ISP-memory is specified, the range of the IAP-memory is determined by IAP and the ISP starts address as listed below.

IAP high boundary = ISP start address –1.
IAP low boundary = ISP start address - IAP.

If ISP-memory is not specified, the range of the IAP-memory is determined by the following formula.

IAP high boundary = 0x1FFF.
IAP low boundary = 0x1FFFF – IAP + 1.

For example, if ISP-memory is **1K**, so that ISP start address is **0x1C00**, and IAP-memory is **1K**, then the IAP-memory range is located at **0x1800 ~ 0x1BFF**. The IAP low boundary in **MPC82E/L52** is defined by IAPLB register which can only be modified by writer.

19.4.2. Update data in IAP-memory

The special function registers are related to ISP/IAP would be shown in Section “19.5 ISP/IAP Register“.

Because the IAP-memory is a part of Flash memory, only ***Page Erase, no Byte Erase***, is provided for Flash erasing. To update “one byte” in the IAP-memory, users can not directly program the new datum into that byte. The following steps show the proper procedure:

Step 1: Save the whole page flash data (with 512 bytes) into XRAM buffer which contains the data to be updated.
Step 2: Erase this page ***(using ISP/IAP Flash Page Erase mode)***.
Step 3: Modify the new data on the byte(s) in the XRAM buffer.
Step 4: Program the updated data out of the XRAM buffer into this page ***(using ISP/IAP Flash Program mode)***.

To read the data in the IAP-memory, users can use the ***ISP/IAP Flash Read mode*** to get the targeted data.

19.4.3. Notes for IAP

Interrupts during IAP

After triggering the ISP/IAP flash processing for In-Application Programming, the MCU will halt for a while for internal IAP processing until the processing is completed. At this time, the interrupt will queue up for being serviced if the interrupt is enabled previously. Once the processing is completed, the MCU continues running and the interrupts in the queue will be serviced immediately if the interrupt flag is still active. Users, however, should be aware of the following:

(1) Any interrupt cannot be in-time serviced during the MCU halts for IAP processing.
(2) The low/high-level triggered external interrupts, INTx, should keep activated until the IAP is completed, or they will be neglected.

IAP and Idle mode

MPC82E/L52 does not make use of idle-mode to perform IAP function. Instead, it freezes CPU running to release the flash memory for ISP/IAP engine operating. Once ISP/IAP operation finished, CPU will be resumed and advanced to the instruction which follows the previous instruction that invokes ISP/AP activity.

Accessing Destination of IAP

As mentioned previously, the IAP is used to program only the IAP-memory. Once the accessing destination is not within the IAP-memory, the hardware will automatically neglect the triggering of IAP processing. That is the triggering of IAP is invalid and the hardware does nothing.

An Alternative Method to Read IAP Data

To read the Flash data in the IAP-memory, in addition to using the Flash Read Mode, the alternative method is using the instruction “MOVC A,@A+DPTR”. Where, DPTR and ACC are filled with the wanted address and the offset, respectively. And, the accessing destination must be within the IAP-memory, or the read data will be indeterminate. Note that using ‘MOVC’ instruction is much faster than using the Flash Read Mode.

Flash Endurance for IAP

The endurance of the embedded Flash is 20,000 erase/write cycles, that is to say, the erase-then-write cycles shouldn’t exceed 20,000 times. Thus the user should pay attention to it in the application which needs to frequently update the IAP-memory.

19.5. ISP/IAP Register

The following special function registers are related to the access of ISP and IAP operation:

IFD: ISP/IAP Flash Data Register

SFR Address = 0xE2 RESET = 1111-1111

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

IFD is the data port register for ISP/IAP operation. The data in IFD will be written into the desired address in operating ISP/IAP write and it is the data window of readout in operating ISP/IAP read.

IFADRH: ISP/IAP Address for High-byte addressing

SFR Address = 0xE3 RESET = 0000-0000

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

IFADRH is the high-byte address port for all ISP/IAP modes.

IFADRL: ISP/IAP Address for Low-byte addressing

SFR Address = 0xE4 RESET = 0000-0000

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

IFADRL is the low byte address port for all ISP/IAP modes.

IFMT: ISP/IAP Flash Mode Table

SFR Page = Normal

SFR Address = 0xE5 RESET = xxxx-xx00

7	6	5	4	3	2	1	0
--	--	--	MS.4	MS.3	MS.2	MS.1	MS.0
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7~5: Reserved. Software must write “0000_0” on these bits when IFMT is written.

Bit 3~0: ISP/IAP/Page-P operating mode selection

MS[4:0]	Mode
0 0 0 0 0	Standby
0 0 0 0 1	Flash byte read of AP/IAP-memory
0 0 0 1 0	Flash byte program of AP/IAP-memory
0 0 0 1 1	Flash page erase of AP/IAP-memory
Others	Reserved

IFMT is used to select the flash mode for performing numerous ISP/IAP function.

SCMD: Sequential Command Data register

SFR Address = 0xE6 RESET = xxxx-xxxx

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

SCMD is the command port for triggering ISP/IAP activity. If SCMD is filled with sequential 0x46h, 0xB9h and if ISPCR.7 = 1, ISP/IAP activity will be triggered.

ISPCR: ISP Control Register

SFR Address = 0xE7 RESET = 0000-0xxx

7	6	5	4	3	2	1	0
ISPEN	SWBS	SWRST	CFAIL	--	WAIT.2	WAIT.1	WAIT.0
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: ISPEN, ISP/IAP/Page-P operation enable.
0: Global disable all ISP/IAP/Page-P program/ read function.
1: Enable ISP/IAP/Page-P program/ read function.

Bit 6: SWBS, software boot selection control.
0: Boot from main-memory after reset.
1: Boot from ISP memory after reset.

Bit 5: SWRST, software reset trigger control.
0: No operation
1: Generate software system reset. It will be cleared by hardware automatically.

Bit 4: CFAIL, Command Fail indication for ISP/IAP operation.
0: The last ISP/IAP command has finished successfully.
1: The last ISP/IAP command fails. It could be caused since the access of flash memory was inhibited.

Bit 3: Reserved. Software must write “0” on this bit when ISPCR is written.

Bit 2~0: WAIT.2~0, Configure ISP timing according to the oscillator frequency.

WAIT[2:0]	OSCin Frequency (MHz)
0 0 0	> 24
0 0 1	20 ~ 24
0 1 0	12 ~ 20
0 1 1	6 ~ 12
1 0 0	3 ~ 6
1 0 1	2 ~ 3
1 1 0	1 ~ 2
1 1 1	< 1

19.6. ISP/IAP Sample Code

(1). Required Function: General function call for ISP/IAP flash read

Assembly Code Example:

```
IxP_Flash_Read  EQU                     01h
ISPEN           EQU                     80h

_ixp_read:
ixp_read:

   MOV    ISPCR,#ISPEN              ; Enable Function
   MOV    IFMT,# IxP_Flash_Read         ; ixp_read=0x01

   MOV    IFADRH,??                 ; fill [IFADRH,IFADRL] with byte address
   MOV    IFADRL,??

   MOV    SCMD,#046h                ;
   MOV    SCMD,#0B9h                ;

     MOV    A,IFD                   ; now, the read data exists in IFD

   MOV    IFMT,#000h                ; Flash_Standby=0x00
   ANL    ISPCR,#(0FFh – ISPEN)         ; Disable Function

   RET
```

C Code Example:

```
#define  Flash_Standby                  0x00
#define  IxP_Flash_Read                 0x01
#define  ISPEN                          0x80


unsigned char ixp_read (void)
{
   unsigned char arg;
     ISPCR = ISPEN;                  // Enable Function
   IFMT = IxP_Flash_Read;              // IxP_Read=0x01

   IFADRH = ??
   IFADRL = ??

   SCMD = 0x46;                    //
   SCMD = 0xB9;                    //

   arg = IFD;

   IFMT = Flash_Standby;               // Flash_Standby=0x00
   ISPCR &= ~ISPEN;

   return arg;
}
```

(2). Required Function: General function call for ISP/IAP flash Erase

Assembly Code Example:

```
IxP_Flash_ Erase EQU                    03h
ISPEN            EQU                    80h

_ixp_erase:
ixp_erase:

   MOV    ISPCR,#ISPEN              ; Enable Function
   MOV    IFMT,# IxP_Flash_Erase        ; ixp_erase=0x03

   MOV    IFADRH,??                 ; fill [IFADRH,IFADRL] with byte address
   MOV    IFADRL,??

   MOV    SCMD,#046h                ;
   MOV    SCMD,#0B9h                ;

   MOV    IFMT,#000h                ; Flash_Standby=0x00
   ANL    ISPCR,#(0FFh – ISPEN)         ; Disable Function

   RET
```

C Code Example:

```
#define  Flash_Standby                  0x00
#define  IxP_Flash_Erase                0x03
#define  ISPEN                          0x80


void ixp_erase (unsigned char Addr_H, unsigned char Addr_L)
{
   ISPCR = ISPEN;                  // Enable Function
   IFMT = IxP_Flash_Erase;             // IxP_Erase=0x03

   IFADRH = Addr_H;
   IFADRL = Addr_L;

   SCMD = 0x46;                    //
   SCMD = 0xB9;                    //

   IFMT = Flash_Standby;               // Flash_Standby=0x00
   ISPCR &= ~ISPEN;
}
```

(3). Required Function: General function call for ISP/IAP flash program

Assembly Code Example:

```
IxP_Flash_Program       EQU                 02h
ISPEN                   EQU                 80h

_ixp_program:
ixp_program:

   MOV    ISPCR,#ISPEN              ; Enable Function
   MOV    IFMT,# IxP_Flash_Program      ; ixp_program=0x03

   MOV    IFADRH,??                 ; fill [IFADRH,IFADRL] with byte address
   MOV    IFADRL,??
     MOV    IFD, A                  ; now, the program data exists in Accumulator

   MOV    SCMD,#046h                ;
   MOV    SCMD,#0B9h                ;

   MOV    IFMT,#000h                ; Flash_Standby=0x00
   ANL    ISPCR,#(0FFh – ISPEN)         ; Disable Function

   RET
```

C Code Example:

```
#define  Flash_Standby                  0x00
#define  IxP_Flash_Program              0x02
#define  ISPEN                          0x80

void ixp_program(unsigned char Addr_H, unsigned char Addr_L, unsigned char dta)
{
   ISPCR = ISPEN;                  // Enable Function
   IFMT = IxP_Flash_Program;           // IxP_Program=0x02

   IFADRH = Addr_H;
   IFADRL = Addr_L;
   IFD = dta;

   SCMD = 0x46;                    //
   SCMD = 0xB9;                    //

   IFMT = Flash_Standby;               // Flash_Standby=0x00
   ISPCR &= ~ISPEN;


}
```

20. Auxiliary SFRs

AUXR: Auxiliary Register

SFR Address = 0x8E RESET = 0000-00xx

7	6	5	4	3	2	1	0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	--	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 7: T0X12, Timer 0 clock source selector while C/T=0.
0: Clear to select SYSCLK/12.
1: Set to select SYSCLK as the clock source.

Bit 6: T1X12, Timer 1 clock source selector while C/T=0.
0: Clear to select SYSCLK/12.
1: Set to select SYSCLK as the clock source.

Bit 5: URM0X6, Serial Port mode 0 baud rate selector.
0: Clear to select SYSCLK/12 as the baud rate for UART Mode 0.
1: Set to select SYSCLK/2 as the baud rate for UART Mode 0.

Bit 4: EADCI, ADC interrupt enable register.
0: Disable ADC interrupt.
1: Enable ADC interrupt.

Bit 3: ESPI, Enable SPI interrupt.
0: Disable SPI interrupt.
1: Enable SPI interrupt.

Bit 2: ENLVFI, Enable LVD Interrupt.
0: Disable LVD (LVF) interrupt.
1: Enable LVD (LVF) interrupt.

Bit 1~0: Reserved. Software must write “0” on these bits when AUXR is written.

21. Hardware Option

The MCU's Hardware Option defines the device behavior which cannot be programmed or controlled by software. The hardware options can only be programmed by a Universal Programmer or the "Megawin 8051 Writer U1". After whole-chip erased, all the hardware options are left in "disabled" state and there is no ISP-memory and IAP-memory configured. The **MPC82E/L52** has the following Hardware Options:

LOCK:
☑: Enabled. Code dumped on a universal Writer or Programmer is locked to 0xFF for security.
☐: Disabled. Not locked.

SB:
☑: Enabled. Code dumped on a universal Writer or Programmer is scrambled for security.
☐: Disabled. Not scrambled.

ISP-memory Space:
The ISP-memory space is specified by its starting address. And, its higher boundary is limited by the Flash end address, i.e., 0x1FFF. The following table lists the ISP space option in this chip. In default setting, **MPC82E/L52** ISP space is configured to **1K** that had been embedded Megawin proprietary ISP code to perform In-System-Programming through Megawin 1-Line ISP protocol.

ISP-memory Size	ISP Start Address
3K bytes	0x1400
2K bytes	0x1800
1K bytes	0x1C00
No ISP Space	--

HWBS:
☑: Enabled. When powered up, MCU will boot from ISP-memory if ISP-memory is configured.
☐: Disabled. MCU always boots from AP-memory.

IAP-memory Space:
The IAP-memory space specifies the user defined IAP space. The IAP-memory Space can be configured by hardware option. In default, it is configured to 1K bytes.

ENLVR:
☑: Enabled. LVD will trigger a RESET event to CPU on AP program start address. (3.7V for 5V device and 2.3V for 3.3V device)
☐: Disabled. LVD can not trigger a RESET to CPU.

LVFWP:
☑: Enabled. LVF inhibits the flash write action in ISP/IAP operation. (3.7V for 5V device and 2.3V for 3.3V device)
☐: Disabled. No inhibition on the flash-writing action.

OSCDN:
☑: Enabled. Low gain option on crystal oscillating circuit to reduce power consumption and EMI radiation. It is applied for crystal < 12MHz.
☐: Disabled. Typical gain setting is up to 25MHz crystal supporting.

ENROSC:
☑: Enabled. Set MCU running internal 6MHz RC-oscillator after power-on.
☐: Disabled. Set MCU running crystal mode after power-on.

HWENW: Hardware loaded for "ENW" of WDTCR.
☑: Enabled. Enable WDT and load the content of HWWIDL and HWPS2~0 to WDTCR after power-on.
☐: Disabled. WDT is not enabled automatically after power-on.

HWWIDL, HWPS2, HWPS1, HWPS0:
When HWENW is enabled, the content on these four fused bits will be loaded to WDTCR SFR after power-on.

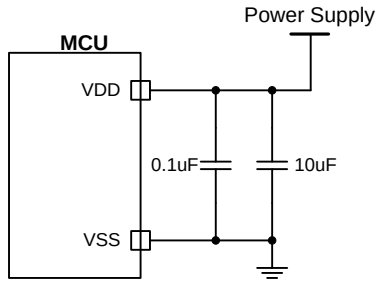
WDSFWP:
☑: Enabled. The WDT SFRs, WIDL, PS2, PS1 and PS0 in WDTCR, will be write-protected.
☐: Disabled. The WDT SFRs, WIDL, PS2, PS1 and PS0 in WDTCR, are free for writing of software.

22. Application Notes

22.1. Power Supply Circuit

To have the **MPC82E/L52** work with power supply varying from 4.5V to 5.5V for E-type and from 2.4V to 3.6V for L-type, adding some external decoupling and bypass capacitors is necessary, as shown in Figure 22–1.

Figure 22–1. Power Supplied Circuit



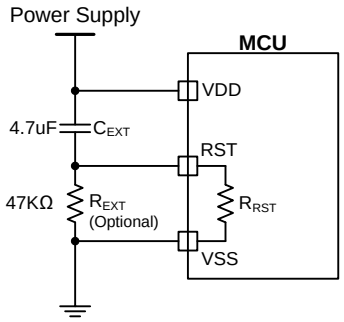
22.2. Reset Circuit

Normally, the power-on reset can be successfully generated during power-up. However, to further ensure the MCU a reliable reset during power-up, the external reset is necessary. Figure 22–2 shows the external reset circuit, which consists of a capacitor C_{EXT} connected to VDD (power supply) and a resistor R_{EXT} connected to VSS (ground).

In general, R_{EXT} is optional because the RST pin has an internal pull-down resistor (R_{RST}). This internal diffused resistor to VSS permits a power-up reset using only an external capacitor C_{EXT} to VDD.

See Section "23.2 DC Characteristics" for R_{RST} value.

Figure 22–2. Reset Circuit



22.3. XTAL Oscillating Circuit

To achieve successful and exact oscillating (up to 25MHz), the capacitors C1 and C2 are necessary, as shown in Figure 22–3. Normally, C1 and C2 have the same value. Table 22–1 lists the C1 & C2 value for the different frequency crystal application.

Figure 22–3. XTAL Oscillating Circuit

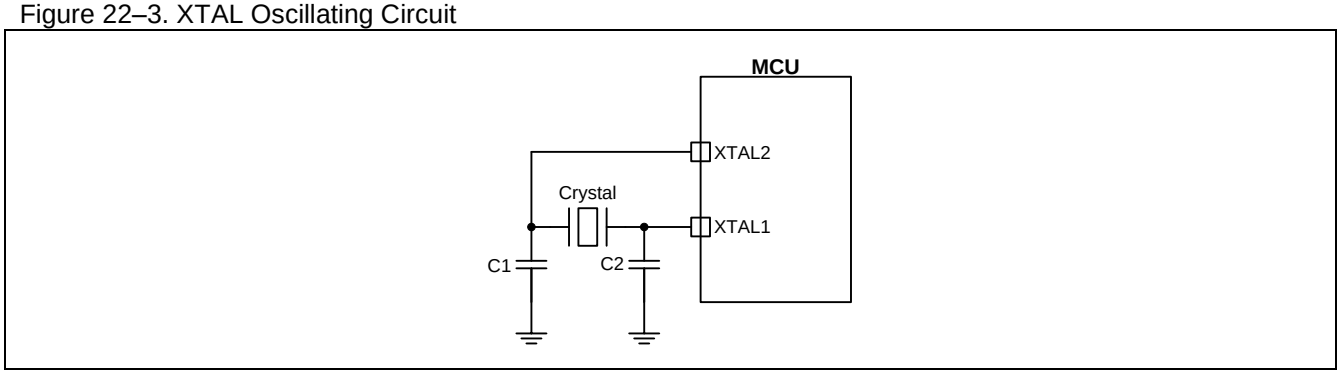


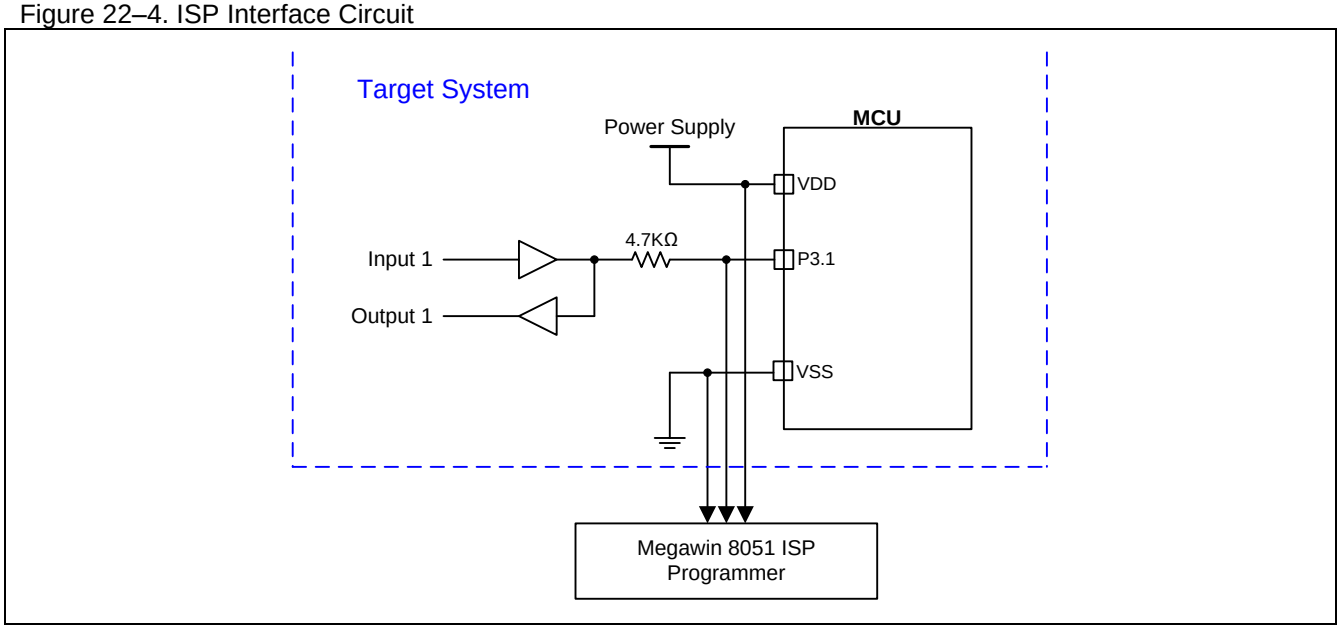
Table 22–1. Reference Capacitance of C1 & C2 for crystal oscillating circuit

Crystal	C1, C2 Capacitance
16MHz ~ 25MHz	10pF
6MHz ~ 16MHz	15pF
2MHz ~ 6MHz	33pF

22.4. ISP Interface Circuit

MPC82E/L52 has been inserted Megawin standard ISP code before shipping which embeds Megawin proprietary 1-Line ISP protocol. The ISP interface allows the P3.1 pin to be shared with user functions so that In-System Flash Programming function could be performed. This is practicable because ISP communication is performed when the device is running in the ISP memory, where the user software (AP memory) is stalled. In this state, the Megawin ISP firmware can safely 'borrow' the P3.1 pin. In most applications, external resistors are required to isolate ISP interface traffic from the user application. A typical isolation configuration is shown in Figure 22–4.

Figure 22–4. ISP Interface Circuit



23. Electrical Characteristics

23.1. Absolute Maximum Rating

For MPC82E52:

Parameter	Rating	Unit
Ambient temperature under bias	-40 ~ +85	°C
Storage temperature	-65 ~ + 150	°C
Voltage on any Port I/O Pin or RST with respect to VSS	-0.5 ~ VDD + 0.5	V
Voltage on VDD with respect to VSS	-0.5 ~ +6.0	V
Maximum total current through VDD and VSS	400	mA
Maximum output current sunk by any Port pin	40	mA

*Note: stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the devices at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

For MPC82L52:

Parameter	Rating	Unit
Ambient temperature under bias	-40 ~ +85	°C
Storage temperature	-65 ~ + 150	°C
Voltage on any Port I/O Pin or RESET with respect to Ground	-0.3 ~ VDD + 0.3	V
Voltage on VDD with respect to Ground	-0.3 ~ +4.2	V
Maximum total current through VDD and Ground	400	mA
Maximum output current sunk by any Port pin	40	mA

*Note: stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the devices at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

23.2. DC Characteristics

For MPC82E52:

VDD = 5.0V±10%, VSS = 0V, T_A = 25 ℃ and execute NOP for each machine cycle, unless otherwise specified

Symbol	Parameter	Test Condition	Limits			Unit
			min	typ	max	
Input/Output Characteristics						
V_{IH1}	Input High voltage (All I/O Ports)	P0, P1, P2 and P3	2.0			V
V_{IH2}	Input High voltage (RST)		3.5			
V_{IL1}	Input Low voltage (All I/O Ports)	P0, P1, P2 and P3			0.8	V
V_{IL2}	Input Low voltage (RST)				0.8	V
I_{IH}	Input High Leakage current (All Input only or open-drain Ports)	V_{PIN} = VDD		0	10	uA
I_{IL1}	Logic 0 input current (Quasi-mode)	V_{PIN} = 0.45V		17	50	uA
I_{IL2}	Logic 0 input current (All Input only or open-drain Ports)	V_{PIN} = 0.45V		0	10	uA
I_{H2L}	Logic 1 to 0 input transition current (Quasi-mode)	V_{PIN} =1.8V		230	500	uA
I_{OH1}	Output High current (Quasi-Mode)	V_{PIN} =2.4V		220		uA
I_{OH2}	Output High current (All push-pull output ports)	V_{PIN} =2.4V	12	20		mA
I_{OL1}	Output Low current (All I/O Ports)	V_{PIN} =0.45V	12	20		mA
R_{RST}	Internal reset pull-down resistance			100		Kohm
Power Consumption						
I_{OP1}	Normal mode operating current	SYSCLK = 12MHz @ XTAL		12	30	mA
I_{IDLE1}	Idle mode operating current	SYSCLK = 12MHz @ XTAL		6	15	mA
I_{PD1}	Power down mode current			0.1	50	uA
LVD Characteristics						
V_{LVD}	LVD detection level	SYSCLK = 12MHz @ XTAL		3.7(1)		V
Operating Condition						
V_{PSR}	Power-on Slop Rate	T_A = -40℃ to +85℃	0.05			V/ms
V_{OP1}	Operating Speed 0–25MHz	T_A = -40℃ to +85℃	4.5		5.5	V
V_{OP2}	Operating Speed 0-12MHz	T_A = -40℃ to +85℃	4.2		5.5	V

(1) Data based on characterization results, not tested in production.

For MPC82L52:

VDD = 3.3V±10%, VSS = 0V, T_A = 25 ℃ and execute NOP for each machine cycle, unless otherwise specified

Symbol	Parameter	Test Condition	Limits			Unit
			min	typ	max	
Input/Output Characteristics						
V_{IH1}	Input High voltage (All I/O Ports)	P0, P1, P2 and P3	2.0			V
V_{IH3}	Input High voltage (RST)		2.8			V
V_{IL1}	Input Low voltage (All I/O Ports)	P0, P1, P2 and P3			0.8	V
V_{IL3}	Input Low voltage (RST)				0.8	V
I_{IH}	Input High Leakage current (All Input only or open-drain Ports)	V_{PIN} = VDD		0	10	uA
I_{IL1}	Logic 0 input current (Quasi-mode)	V_{PIN} = 0.45V		7	50	uA
I_{IL2}	Logic 0 input current (All Input only or open-drain Ports)	V_{PIN} = 0.45V		0	10	uA
I_{H2L}	Logic 1 to 0 input transition current (Quasi-mode)	V_{PIN} =1.4V		100	600	uA
I_{OH1}	Output High current (Quasi-Mode)	V_{PIN} =2.4V		64		uA
I_{OH2}	Output High current (All push-pull output ports)	V_{PIN} =2.4V	4	8		mA
I_{OL1}	Output Low current (All I/O Ports)	V_{PIN} =0.45V	8	14		mA
R_{RST}	Internal reset pull-down resistance			100		Kohm
Power Consumption						
I_{OP1}	Normal mode operating current	SYSCLK = 12MHz @ XTAL		9	15	mA
I_{IDLE1}	Idle mode operating current	SYSCLK = 12MHz @ XTAL		3.5	6	mA
I_{PD1}	Power down mode current			0.1	50	uA
LVD Characteristics						
V_{LVD}	LVD detection level	SYSCLK = 12MHz @ XTAL		2.3(1)		V
Operating Condition						
V_{PSR}	Power-on Slop Rate	T_A = -40℃ to +85℃	0.05			V/ms
V_{OP1}	Operating Speed 0–25MHz	T_A = -40℃ to +85℃	2.7		3.6	V
V_{OP2}	Operating Speed 0-12MHz	T_A = -40℃ to +85℃	2.4		3.6	V

(1) Data based on characterization results, not tested in production.

23.3. External Clock Characteristics

For MPC82E52:

VDD = 4.5V ~ 5.5V, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

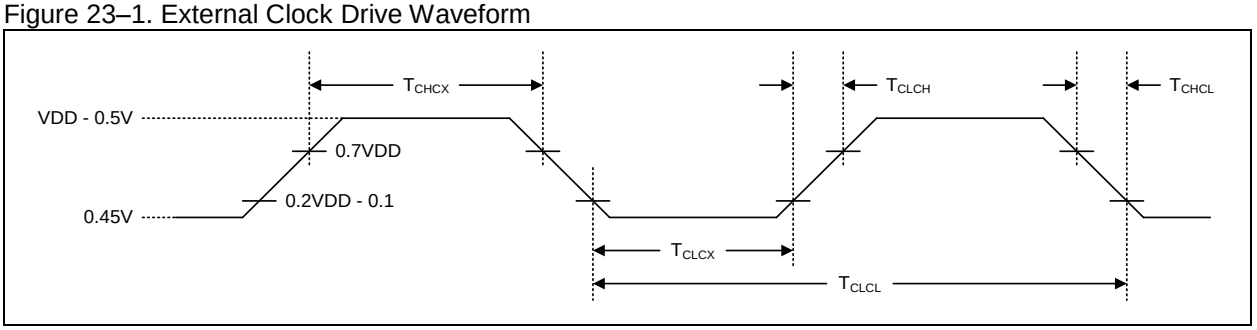
Symbol	Parameter	Oscillator Crystal Mode		Unit
		Min.	Max	
$1/t_{CLCL}$	Oscillator Frequency	2	25	MHz
$1/t_{CLCL}$	Oscillator Frequency (VDD = 4.2V ~ 5.5V)	2	12	MHz
t_{CLCL}	Clock Period	40		ns
t_{CHCX}	High Time	0.4T	0.6T	t_{CLCL}
t_{CLCX}	Low Time	0.4T	0.6T	t_{CLCL}
t_{CLCH}	Rise Time		5	ns
t_{CHCL}	Fall Time		5	ns

For MPC82L52:

VDD = 2.7V ~ 3.6V, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

Symbol	Parameter	Oscillator Crystal Mode		Unit
		Min.	Max	
$1/t_{CLCL}$	Oscillator Frequency	2	25	MHz
$1/t_{CLCL}$	Oscillator Frequency (VDD = 2.4V ~ 3.6V)	2	12	MHz
t_{CLCL}	Clock Period	40		ns
t_{CHCX}	High Time	0.4T	0.6T	t_{CLCL}
t_{CLCX}	Low Time	0.4T	0.6T	t_{CLCL}
t_{CLCH}	Rise Time		5	ns
t_{CHCL}	Fall Time		5	ns

Figure 23–1. External Clock Drive Waveform



23.4. IRCO Characteristics

For MPC82E52:

Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Voltage		4.5		5.5	V
IRCO Frequency	TA = +25℃	4.2(1)	6(1)	7.8(1)	MHz

(1) Data based on characterization results, not tested in production.

For MPC82L52:

Parameter	Test Condition	Limits			Unit
		min	Typ	max	
Supply Voltage		2.7		3.6	V
IRCO Frequency	TA = +25℃	4.2(1)	6(1)	7.8(1)	MHz

(1) Data based on characterization results, not tested in production.

23.5. Flash Characteristics

For MPC82E52:

Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Voltage	TA = -40℃ to +85℃	4.2		5.5	V
Flash Write (Erase/Program) Voltage	TA = -40℃ to +85℃	4.5		5.5	V
Flash Erase/Program Cycle	TA = -40℃ to +85℃	20,000			times
Flash Data Retention	TA = +25℃	100			year

For MPC82L52:

Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Voltage	TA = -40℃ to +85℃	2.4		3.6	V
Flash Write (Erase/Program) Voltage	TA = -40℃ to +85℃	2.7		3.6	V
Flash Erase/Program Cycle	TA = -40℃ to +85℃	20,000			times
Flash Data Retention	TA = +25℃	100			year

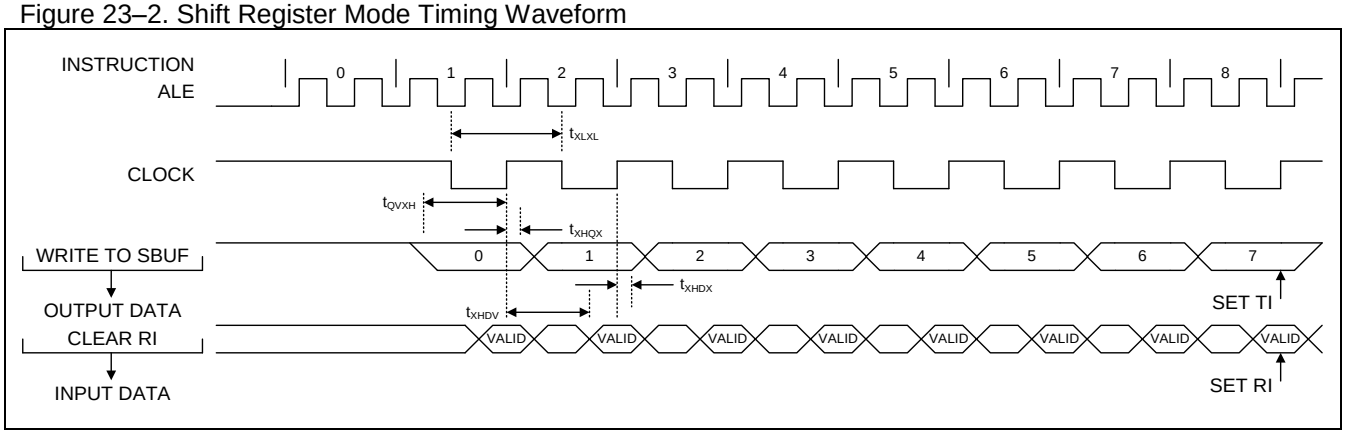
23.6. Serial Port Timing Characteristics

For MPC82E52: VDD = 5.0V±10%, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

For MPC82L52: VDD = 3.3V±10%, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

Symbol	Parameter	URM0X3 = 0		URM0X3 = 1		Unit
		Min.	Max	Min.	Max	
t_{XLXL}	Serial Port Clock Cycle Time	12T		4T		T_{SYSCLK}
t_{QVXH}	Output Data Setup to Clock Rising Edge	10T-20		T-20		ns
t_{XHQX}	Output Data Hold after Clock Rising Edge	T-10		T-10		ns
t_{XHDX}	Input Data Hold after Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		10T-20		4T-20	ns

Figure 23–2. Shift Register Mode Timing Waveform



23.7. SPI Timing Characteristics

For MPC82E52: VDD = 5.0V±10%, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified
For MPC82L52: VDD = 3.3V±10%, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

Symbol	Parameter	Min	Max	Units
Master Mode Timing				
t_{MCKH}	SPICLK High Time	2T		T_{SYSCLK}
t_{MCKL}	SPICLK Low Time	2T		T_{SYSCLK}
t_{MIS}	MISO Valid to SPICLK Shift Edge	2T+20		ns
t_{MIH}	SPICLK Shift Edge to MISO Change	0		ns
t_{MOH}	SPICLK Shift Edge to MOSI Change		10	ns
Slave Mode Timing				
t_{SE}	nSS Falling to First SPICLK Edge	2T		T_{SYSCLK}
t_{SD}	Last SPICLK Edge to nSS Rising	2T		T_{SYSCLK}
t_{SEZ}	nSS Falling to MISO Valid		4T	T_{SYSCLK}
t_{SDZ}	nSS Rising to MISO High-Z		4T	T_{SYSCLK}
t_{CKH}	SPICLK High Time	4T		T_{SYSCLK}
t_{CKL}	SPICLK Low Time	4T		T_{SYSCLK}
t_{SIS}	MOSI Valid to SPICLK Sample Edge	2T		T_{SYSCLK}
t_{SIH}	SPICLK Sample Edge to MOSI Change	2T		T_{SYSCLK}
t_{SOH}	SPICLK Shift Edge to MISO Change		4T	T_{SYSCLK}
t_{SLH}	Last SPICLK Edge to MISO Change (CPHA = 1 ONLY)	1T	2T	T_{SYSCLK}

Figure 23–3. SPI Master Transfer Waveform with CPHA=0

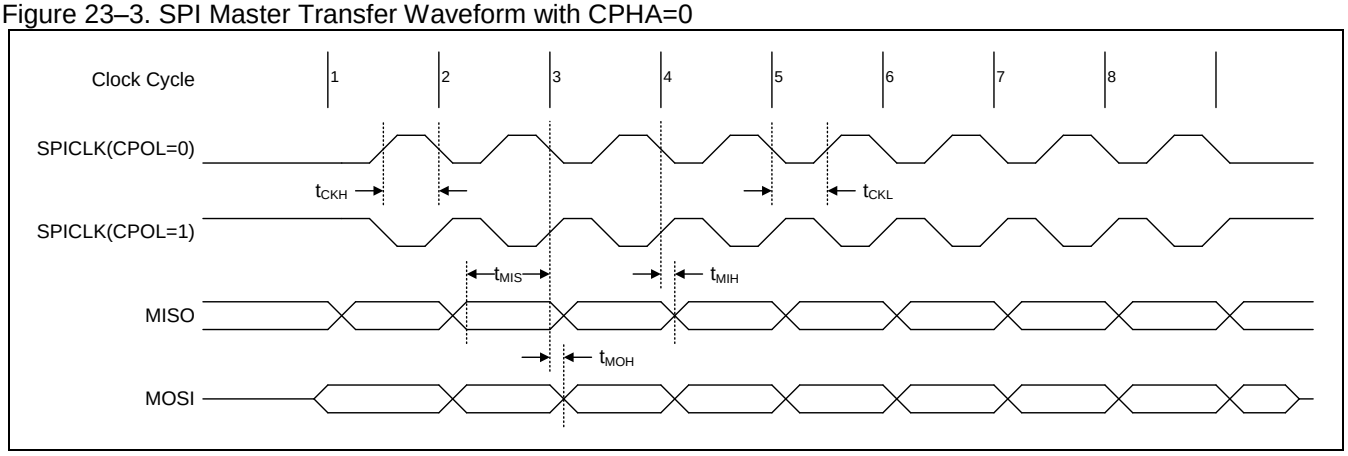


Figure 23–4. SPI Master Transfer Waveform with CPHA=1

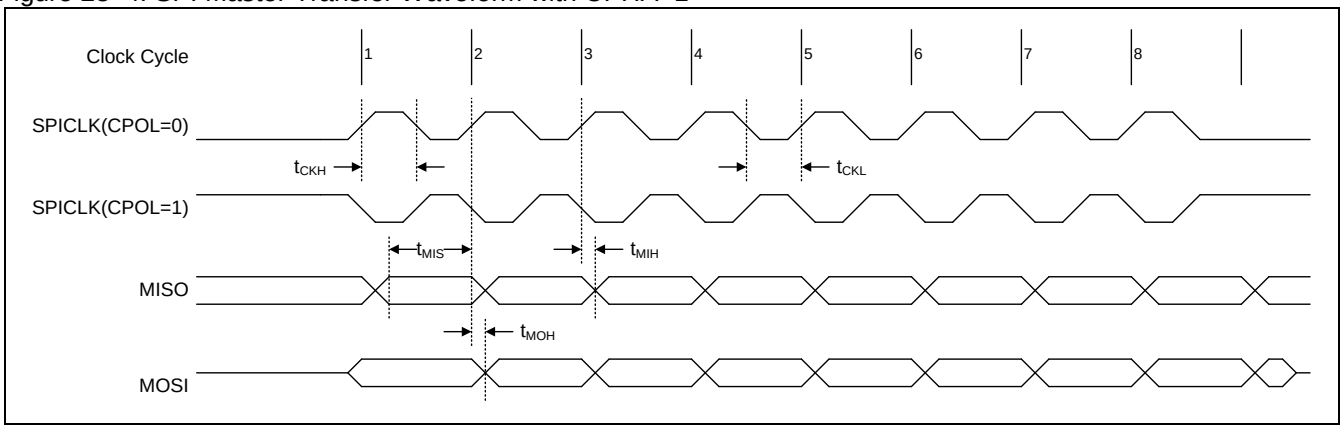


Figure 23–5. SPI Slave Transfer Waveform with CPHA=0

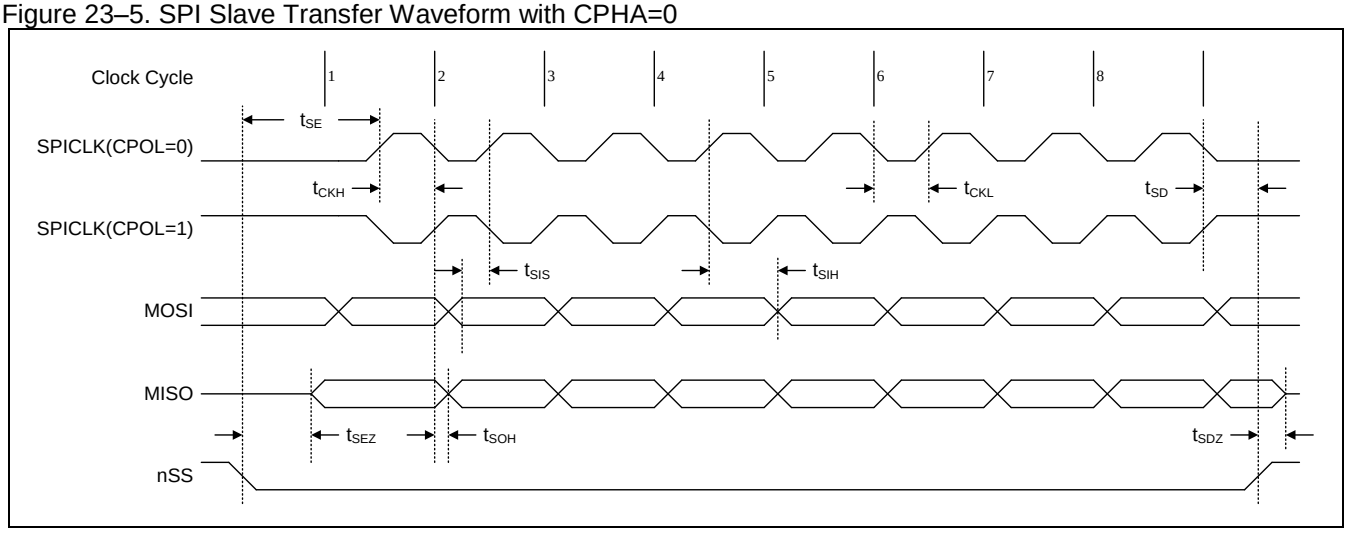
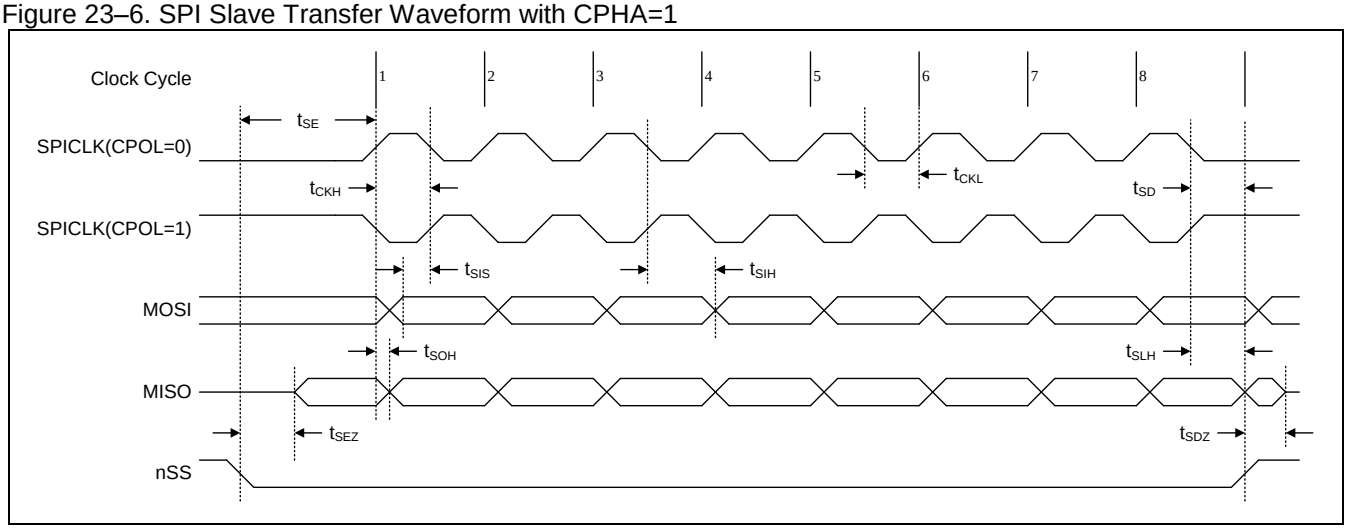


Figure 23–6. SPI Slave Transfer Waveform with CPHA=1



24. Instruction Set

Table 24–1. Instruction Set

MNEMONIC	DESCRIPTION	BYTE	EXECUTION Cycles
DATA TRASFER			
MOV A,Rn	Move register to Acc	1	1
MOV A,direct	Move direct byte o Acc	2	2
MOV A,@Ri	Move indirect RAM to Acc	1	2
MOV A,#data	Move immediate data to Acc	2	2
MOV Rn,A	Move Acc to register	1	2
MOV Rn,direct	Move direct byte to register	2	4
MOV Rn,#data	Move immediate data to register	2	2
MOV direct,A	Move Acc to direct byte	2	3
MOV direct,Rn	Move register to direct byte	2	3
MOV direct,direct	Move direct byte to direct byte	3	4
MOV direct,@Ri	Move indirect RAM to direct byte	2	4
MOV direct,#data	Move immediate data to direct byte	3	3
MOV @Ri,A	Move Acc to indirect RAM	1	3
MOV @Ri,direct	Move direct byte to indirect RAM	2	3
MOV @Ri,#data	Move immediate data to indirect RAM	2	3
MOV DPTR,#data16	Load DPTR with a 16-bit constant	3	3
MOVC A,@A+DPTR	Move code byte relative to DPTR to Acc	1	4
MOVC A,@A+PC	Move code byte relative to PC to Acc	1	4
MOVX A,@Ri	Move on-chip auxiliary RAM(8-bit address) to Acc	1	3
MOVX A,@DPTR	Move on-chip auxiliary RAM(16-bit address) to Acc	1	3
MOVX @Ri,A	Move Acc to on-chip auxiliary RAM(8-bit address)	1	4
MOVX @DPTR,A	Move Acc to on-chip auxiliary RAM(16-bit address)	1	3
MOVX A,@Ri	Move external RAM(8-bit address) to Acc	1	Not Support
MOVX A,@DPTR	Move external RAM(16-bit address) to Acc	1	Not Support
MOVX @Ri,A	Move Acc to external RAM(8-bit address)	1	Not Support
MOVX @DPTR,A	Move Acc to external RAM(16-bit address)	1	Not Support
PUSH direct	Push direct byte onto Stack	2	4
POP direct	Pop direct byte from Stack	2	3
XCH A,Rn	Exchange register with Acc	1	3
XCH A,direct	Exchange direct byte with Acc	2	4
XCH A,@Ri	Exchange indirect RAM with Acc	1	4
XCHD A,@Ri	Exchange low-order digit indirect RAM with Acc	1	4
ARITHEMATIC OPERATIONS			
ADD A,Rn	Add register to Acc	1	2
ADD A,direct	Add direct byte to Acc	2	3
ADD A,@Ri	Add indirect RAM to Acc	1	3
ADD A,#data	Add immediate data to Acc	2	2
ADDC A,Rn	Add register to Acc with Carry	1	2
ADDC A,direct	Add direct byte to Acc with Carry	2	3
ADDC A,@Ri	Add indirect RAM to Acc with Carry	1	3
ADDC A,#data	Add immediate data to Acc with Carry	2	2
SUBB A,Rn	Subtract register from Acc with borrow	1	2
SUBB A,direct	Subtract direct byte from Acc with borrow	2	3
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	3

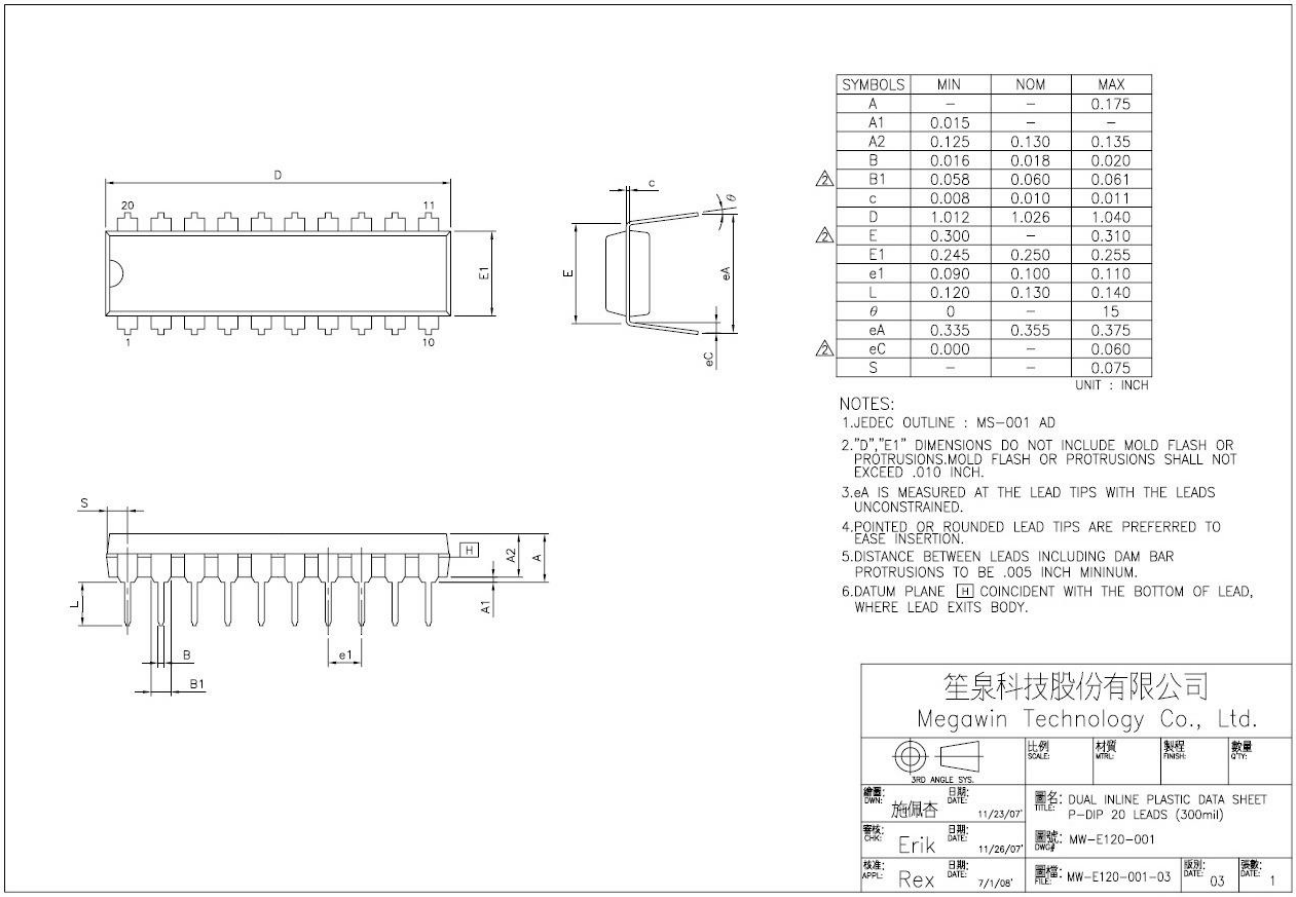
SUBB A,#data	Subtract immediate data from Acc with borrow	2	2
INC A	Increment Acc	1	2
INC Rn	Increment register	1	3
INC direct	Increment direct byte	2	4
INC @Ri	Increment indirect RAM	1	4
DEC A	Decrement Acc	1	2
DEC Rn	Decrement register	1	3
DEC direct	Decrement direct byte	2	4
DEC @Ri	Decrement indirect RAM	1	4
INC DPTR	Increment DPTR	1	1
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	5
DA A	Decimal Adjust Acc	1	4
LOGIC OPERATION			
ANL A,Rn	AND register to Acc	1	2
ANL A,direct	AND direct byte to Acc	2	3
ANL A,@Ri	AND indirect RAM to Acc	1	3
ANL A,#data	AND immediate data to Acc	2	2
ANL direct,A	AND Acc to direct byte	2	4
ANL direct,#data	AND immediate data to direct byte	3	4
ORL A,Rn	OR register to Acc	1	2
ORL A,direct	OR direct byte to Acc	2	3
ORL A,@Ri	OR indirect RAM to Acc	1	3
ORL A,#data	OR immediate data to Acc	2	2
ORL direct,A	OR Acc to direct byte	2	4
ORL direct,#data	OR immediate data to direct byte	3	4
XRL A,Rn	Exclusive-OR register to Acc	1	2
XRL A,direct	Exclusive-OR direct byte to Acc	2	3
XRL A,@Ri	Exclusive-OR indirect RAM to Acc	1	3
XRL A,#data	Exclusive-OR immediate data to Acc	2	2
XRL direct,A	Exclusive-OR Acc to direct byte	2	4
XRL direct,#data	Exclusive-OR immediate data to direct byte	3	4
CLR A	Clear Acc	1	1
CPL A	Complement Acc	1	2
RL A	Rotate Acc Left	1	1
RLC A	Rotate Acc Left through the Carry	1	1
RR A	Rotate Acc Right	1	1
RRC A	Rotate Acc Right through the Carry	1	1
SWAP A	Swap nibbles within the Acc	1	1
BOOLEAN VARIABLE MANIPULATION			
CLR C	Clear Carry	1	1
CLR bit	Clear direct bit	2	4
SETB C	Set Carry	1	1
SETB bit	Set direct bit	2	4
CPL C	Complement Carry	1	1
CPL bit	Complement direct bit	2	4
ANL C,bit	AND direct bit to Carry	2	3
ANL C,/bit	AND complement of direct bit to Carry	2	3
ORL C,bit	OR direct bit to Carry	2	3
ORL C,/bit	OR complement of direct bit to Carry	2	3

MOV C,bit	Move direct bit to Carry	2	3
MOV bit,C	Move Carry to direct bit	2	4
BOOLEAN VARIABLE MANIPULATION			
JC rel	Jump if Carry is set	2	3
JNC rel	Jump if Carry not set	2	3
JB bit,rel	Jump if direct bit is set	3	4
JNB bit,rel	Jump if direct bit not set	3	4
JBC bit,rel	Jump if direct bit is set and then clear bit	3	5
PROAGRAM BRACHING			
ACALL addr11	Absolute subroutine call	2	6
LCALL addr16	Long subroutine call	3	6
RET	Return from subroutine	1	4
RETI	Return from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump	2	3
JMP @A+DPTR	Jump indirect relative to DPTR	1	3
JZ rel	Jump if Acc is zero	2	3
JNZ rel	Jump if Acc not zero	2	3
CJNE A,direct,rel	Compare direct byte to Acc and jump if not equal	3	5
CJNE A,#data,rel	Compare immediate data to Acc and jump if not equal	3	4
CJNE Rn,#data,rel	Compare immediate data to register and jump if not equal	3	4
CJNE @Ri,#data,rel	Compare immediate data to indirect RAM and jump if not equal	3	5
DJNZ Rn,rel	Decrement register and jump if not equal	2	4
DJNZ direct,rel	Decrement direct byte and jump if not equal	3	5
NOP	No Operation	1	1

25. Package Dimension

25.1. DIP-20

Figure 25–1. DIP-20



SYMBOLS	MIN	NOM	MAX
A	–	–	0.175
A1	0.015	–	–
A2	0.125	0.130	0.135
B	0.016	0.018	0.020
⚠ B1	0.058	0.060	0.061
c	0.008	0.010	0.011
D	1.012	1.026	1.040
⚠ E	0.300	–	0.310
E1	0.245	0.250	0.255
e1	0.090	0.100	0.110
L	0.120	0.130	0.140
θ	0	–	15
eA	0.335	0.355	0.375
⚠ eC	0.000	–	0.060
S	–	–	0.075

UNIT : INCH

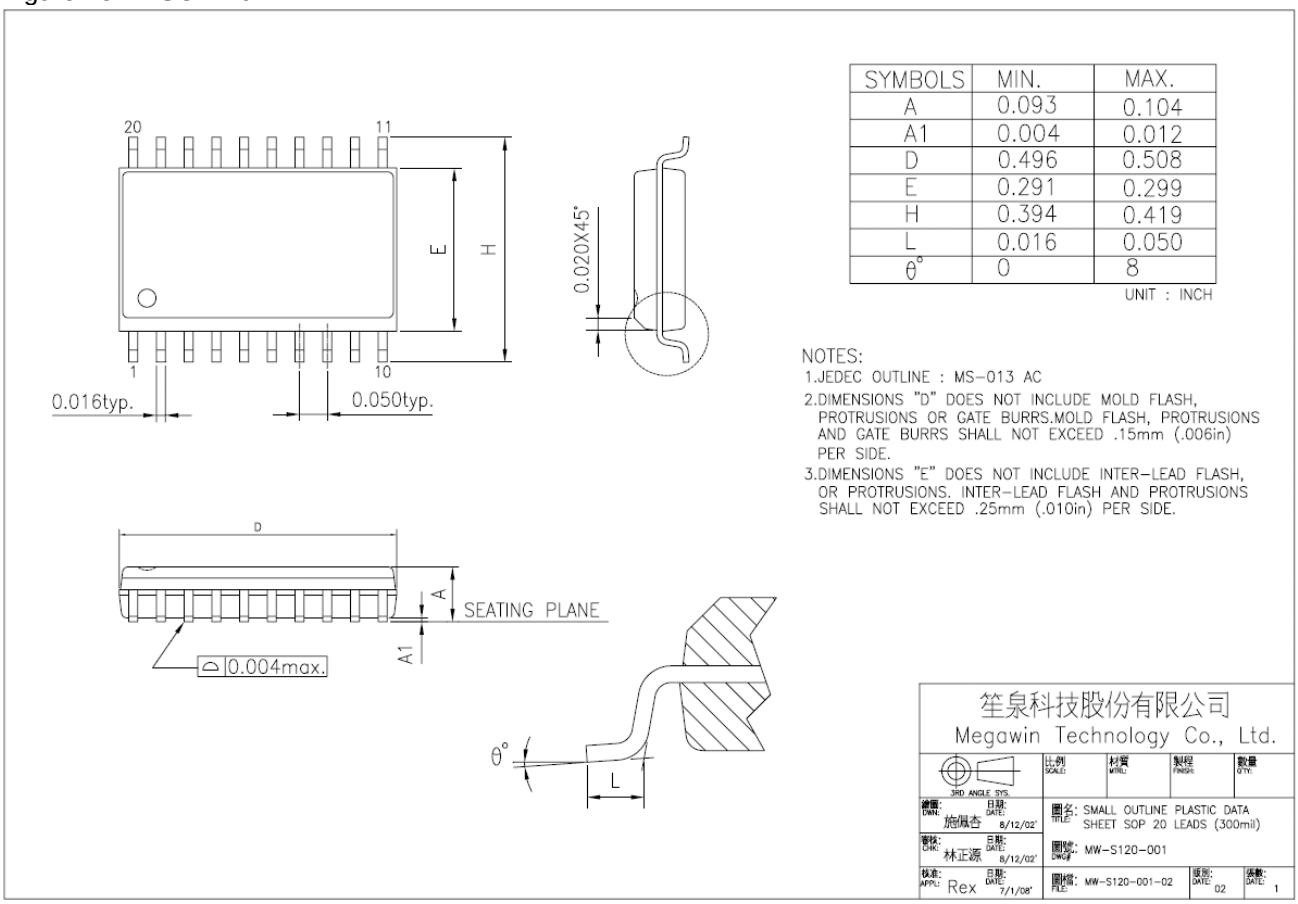
NOTES:
1. JEDEC OUTLINE : MS-001 AD
2. "D","E1" DIMENSIONS DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 INCH.
3. eA IS MEASURED AT THE LEAD TIPS WITH THE LEADS UNCONSTRAINED.
4. POINTED OR ROUNDED LEAD TIPS ARE PREFERRED TO EASE INSERTION.
5. DISTANCE BETWEEN LEADS INCLUDING DAM BAR PROTRUSIONS TO BE .005 INCH MININUM.
6. DATUM PLANE [H] COINCIDENT WITH THE BOTTOM OF LEAD, WHERE LEAD EXITS BODY.

笙泉科技股份有限公司
Megawin Technology Co., Ltd.

3RD ANGLE SYS.	比例 SCALE:	材質 MTRL:	製程 FINISH:	數量 QTY:
繪圖 DWN: 施佩杏 日期 DATE: 11/23/07'	圖名 TITLE: DUAL INLINE PLASTIC DATA SHEET P-DIP 20 LEADS (300mil)			
審核 CHK: Erik 日期 DATE: 11/26/07'	圖號 DWG#: MW-E120-001			
核准 APPL: Rex 日期 DATE: 7/1/08'	圖檔 FILE: MW-E120-001-03	版別 DATE: 03	張數 DATE: 1	

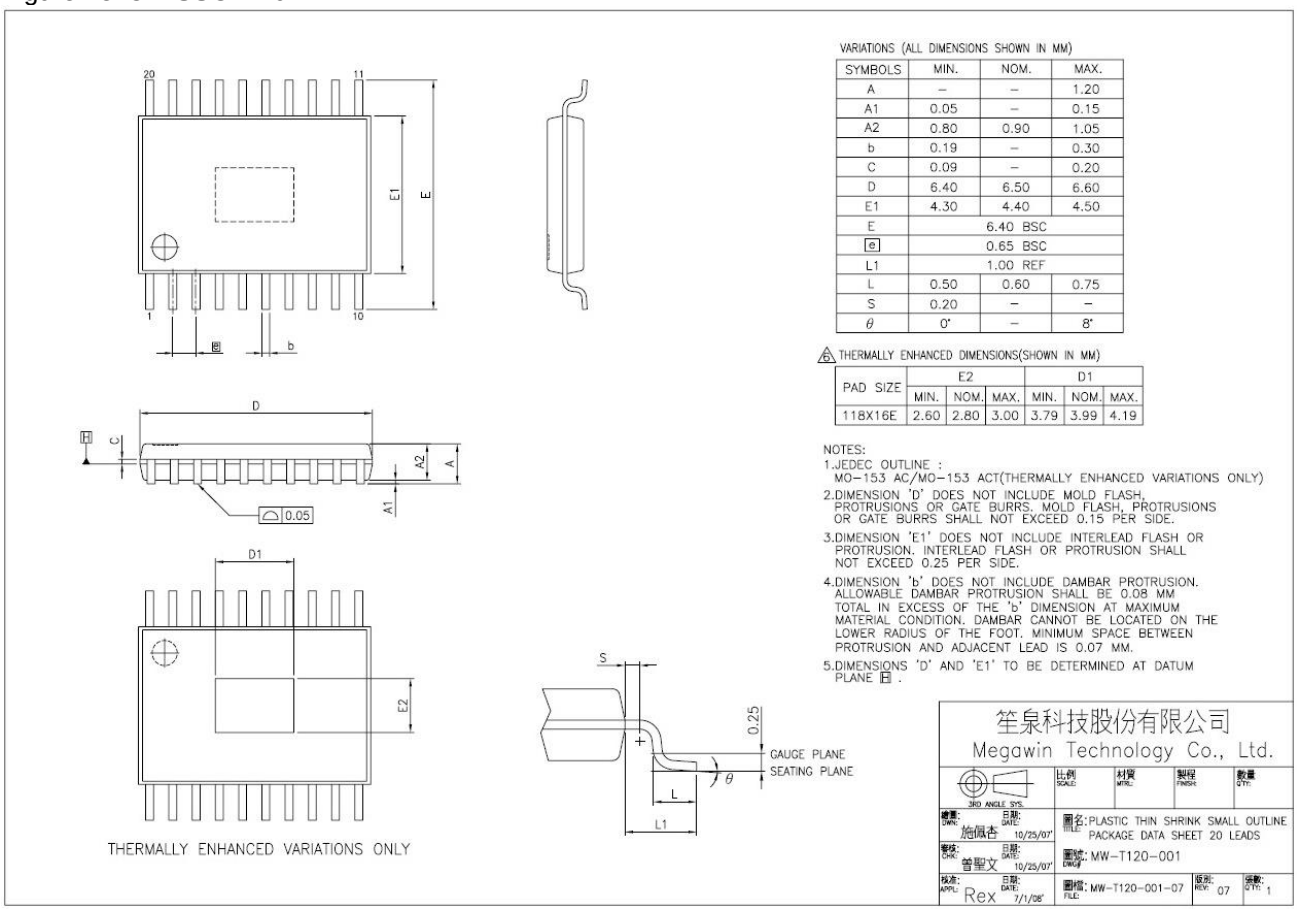
25.2. SOP-20

Figure 25–2. SOP-20



25.3. TSSOP-20

Figure 25–3. TSSOP-20



VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

SYMBOLS	MIN.	NOM.	MAX.
A	–	–	1.20
A1	0.05	–	0.15
A2	0.80	0.90	1.05
b	0.19	–	0.30
C	0.09	–	0.20
D	6.40	6.50	6.60
E1	4.30	4.40	4.50
E	6.40 BSC		
e	0.65 BSC		
L1	1.00 REF		
L	0.50	0.60	0.75
S	0.20	–	–
θ	0°	–	8°

⚠6 THERMALLY ENHANCED DIMENSIONS(SHOWN IN MM)

PAD SIZE	E2			D1		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
118X16E	2.60	2.80	3.00	3.79	3.99	4.19

NOTES:
1. JEDEC OUTLINE : MO-153 AC/MO-153 ACT(THERMALLY ENHANCED VARIATIONS ONLY)
2. DIMENSION 'D' DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS. MOLD FLASH, PROTRUSIONS OR GATE BURRS SHALL NOT EXCEED 0.15 PER SIDE.
3. DIMENSION 'E1' DOES NOT INCLUDE INTERLEAD FLASH OR PROTRUSION. INTERLEAD FLASH OR PROTRUSION SHALL NOT EXCEED 0.25 PER SIDE.
4. DIMENSION 'b' DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 MM TOTAL IN EXCESS OF THE 'b' DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OF THE FOOT. MINIMUM SPACE BETWEEN PROTRUSION AND ADJACENT LEAD IS 0.07 MM.
5. DIMENSIONS 'D' AND 'E1' TO BE DETERMINED AT DATUM PLANE H .

笙泉科技股份有限公司
Megawin Technology Co., Ltd.

3RD ANGLE SYS.	比例 SCALE:	材質 MTRL:	製程 FINISH:	數量 QTY:
繪圖 DWN: 施佩杏 日期 DATE: 10/25/07'	圖名: PLASTIC THIN SHRINK SMALL OUTLINE PACKAGE DATA SHEET 20 LEADS			
審核 CHK: 曾聖文 日期 DATE: 10/25/07'	圖號 DWG#: MW-T120-001			
核准 APPL: Rex 日期 DATE: 7/1/08'	圖檔 FILE: MW-T120-001-07	版次 REV: 07	張數 QTY: 1	

26. Revision History

Table 26–1. Revision History

Rev	Descriptions	Date
A1	1. Initial issue.	2005/09
A2	1. Pin description for MISO/SCLK.	2006/01
	2. Operation Temperature.	2006/01
	3. Correct CCAPPn to CAPPn.	2006/01
	4. Correct SFR PWMMSBn to PCAPWMn.	2006/01
	5. Correct PWM diagram.	2006/01
A3	1. Revises possible operating temperature.	2006/08
A4	1. Add special note on low voltage detector.	2006/12
A5	1. Modify the storage temperature.	2007/03
A6	1. Operation frequency range up to 24 MHz.	2007/11
A7	1. Add 2.7V requirement in flash write operation.	2007/12
	2. Modify Absolute Maximum Rating.	
A8	1. Format modification.	2008/12
A9	1. Format modification.	2012/11
	2. Add sample code.	2012/11
	3. Operation frequency range up to 25 MHz.	2012/11
	3. Add description for SMOD0.	2012/11

Disclaimers

Herein, Megawin stands for "***Megawin Technology Co., Ltd.***"

Life Support — This product is not designed for use in medical, life-saving or life-sustaining applications, or systems where malfunction of this product can reasonably be expected to result in personal injury. Customers using or selling this product for use in such applications do so at their own risk and agree to fully indemnify Megawin for any damages resulting from such improper use or sale.

Right to Make Changes — Megawin reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in mass production, relevant changes will be communicated via an Engineering Change Notification (ECN).