MEGAWIN
MAKE YOU WIN

8051-Based MCU

MG86FE/L508
Data Sheet

Version: A1.3

Features

- 1-T 80C51 Central Processing Unit
- **MG86FE/L508** with **8K** Bytes flash ROM
  - ISP memory zone could be optioned as 0.5KB/1KB/1.5KB……
  - Flexible IAP size by software configured
  - Code protection for flash memory access
  - Flash write/erase cycle
    - For 0.5K IAP, the MTP of IAP write cycle is 2,000 times.
    - For 1.0K IAP, the MTP of IAP write cycle is 1,000 times
  - Flash data retention: 100 years at 25℃
  - MG86FE/L508 Flash space mapping (Default)
    - AP Flash（6KB, 0000h~17FFh）
    - IAP Flash（1KB, 1800h~1BFFh）
    - ISP Flash（1KB, 1C00h~1FFFh）（ISP Boot code）
- On-chip 256 bytes scratch-pad RAM
- Interrupt controller
  - 8 sources, four-level-priority interrupt capability
  - Two external interrupt inputs, nINT0 and nINT1
  - The external interrupts support High/Low level or Rising/Falling edge trigger.
- Two 16-bit timer/counters, Timer 0 and Timer 1.
  - T0CKO on P34 and T1CKO on P35
  - X12 mode enabled for Timer 0/1
  - Support PWM mode
  - Programmable prescaler for Timer 0 clock source
- Programmable 16-bit counter/timer Array (PCA) with 4 channels PWM
  - Auto reload 16-bit base timer
  - 2 compare/capture modules and 2 PWM modules
  - Capture mode
  - 16-bit software timer mode
  - High speed output mode
  - Variable resolution PWM mode, up to 8-bit
- Enhanced UART (S0)
  - Framing Error Detection
  - Automatic Address Recognition
  - Selectable clock polarity in mode 0
  - SPI master support in mode 4
- Keypad Interrupt on 16 GPIOs
- Two wire interface Start/Stop detection
- 8-bit ADC
  - Programmable throughput up to 200 ksps
  - 8 channel single-ended inputs
- Programmable Watchdog Timer, clock sourced from ILRCO
  - One time enabled by CPU or power-on
  - Interrupt CPU or Reset CPU on WDT overflow
  - Support WDT function in power down mode (watch mode)
- Real-Time-Clock
  - 0.5S ~ 64S programmable interrupt period

- 21-bit length system timer

- Beeper function
- Maximum 26 GPIOs in 28-pin package.
  - P3 can be configured to quasi-bidirectional, push-pull output, open-drain output and input only.
  - P1, P2, P4.0 and P4.1 can be configured to push-pull output or open-drain output.
  - P4.0, P4.1 and P3.6 are shared with XTAL2, XTAL1 and RST.
- Multiple power control modes: idle mode, power-down mode, slow mode, sub-clock mode, RTC mode, watch mode and monitor mode.
  - All interrupts can wake up IDLE mode
  - **7** sources to wake up Power-Down mode
  - Slow mode and sub-clock mode support low speed MCU operation
  - RTC mode supports RTC to resume CPU in power down
  - Watch mode supports WDT to resume CPU in power down
  - Monitor mode supports BOD0 to resume CPU in power down (L-series only)
  - Fast wakeup from Power-Down mode by IHRCO (typical 1uS)
- Brown-Out Detector: VDD 4.0V for E-series and VDD 2.6V for L-series
  - Interrupt CPU or reset CPU
  - Wake up CPU in Power-Down mode (L-series only)
- Operating voltage range:
  - MG86FE508: 4.2V~5.5V, minimum 4.5V requirement in flash write operation (ISP/IAP/ICP)
  - MG86FL508: 2.4V~3.6V, minimum 2.7V requirement in flash write operation (ISP/IAP/ICP)
- Operation frequency range: 25MHz(max)
  - MG86FE508: 0 – 12MHz @ 4.2V – 5.5V and 0 – 25MHz @ 4.5V – 5.5V
  - MG86FL508: 0 – 12MHz @ 2.4V – 3.6V and 0 – 25MHz @ 2.7V – 3.6V
- Clock Source
  - Internal 24MHz/22.118MHz oscillator (IHRCO): factory calibrated to ±1%, typical
  - External crystal mode, 32.768KHz oscillating support
  - Internal Low power 64KHz RC Oscillator (ILRCO)
  - External clock input (ECKI) on P4.0/XTAL2
  - IHRCO output on P4.0/XTAL2
- Operating Temperature:
  - Industrial (-40℃ to +85℃)*
- Package Types:
  - SOP28: MG86FE/L508AS28
  - SOP20: MG86FE/L508AS20
  - SOP16: MG86FE/L508AS16

*: Tested by sampling.

Content

1. General Description

The **MG86FE/L508** 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 **MG86FE/L508** can operate at a much lower speed and thereby greatly reduce the power consumption.

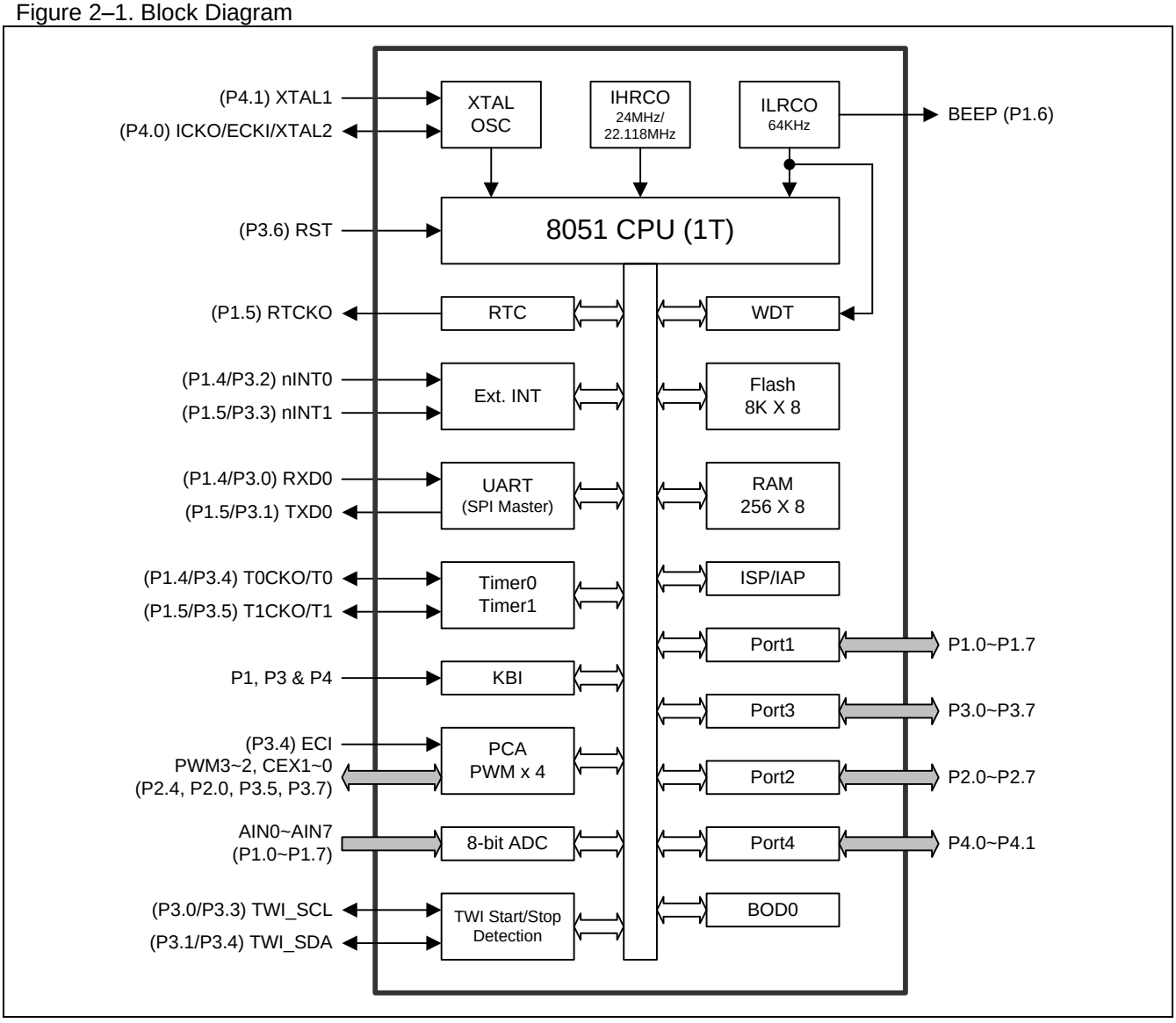
The **MG86FE/L508** has **8K** bytes of embedded Flash memory for code and data. The Flash memory can be programmed either in serial writer mode (via ICP, In-Circuit-Programming) or in ISP (In-System Programming) mode. And, it also provides the In-Application Programming (IAP) capability. ICP and ISP allow 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 **MG86FE/L508** retains all features of the standard 80C52 with 256 bytes of scratch-pad RAM, three 8bit I/O ports, two external interrupts, a multi-source 4-level interrupt controller and two 16-bits timer/counters. In addition, the **MG86FE/L508** has two extra I/O pins (P4.0 and P4.1), keypad interrupt, 8-btis ADC, a 4-channel PCA, Watchdog Timer, Real-Time-Clock module, a Brown-out Detector, an on-chip crystal oscillator (shared with P4.0 and P4.1), a high precision internal oscillator (IHRCO), an internal low speed RC oscillator (ILRCO) and a more versatile serial channel that facilitates multiprocessor communication (EUART).

The **MG86FE/L508** has multiple operating modes to reduce the power consumption: idle mode, power down mode, slow mode, sub-clock mode, RTC mode, watch mode and monitor 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. In slow mode, the user can further reduce the power consumption by using the 8-bit system clock pre-scaler to slow down the operating speed. Or select sub-clock mode which clock source is derived from internal low speed oscillator (ILRCO) for CPU to perform an ultra low speed operation. The RTC mode supports Real-Time-Clock function in all modes. In watch mode, it keeps WDT running in power-down or idle mode and resumes CPU when WDT overflows. Monitor mode provides the Brown-Out detection in power down mode and resumes CPU when chip VDD reaches the specific detection level.

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	CCAP2H	CCAP3H	--	--
F0	B	PAOE	PCAPWM0	PCAPWM1	PCAPWM2	PCAPWM3	--	--
E8	P4	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	--	--
E0	ACC	WDTCR	IFD	IFADRH	IFADRL	IFMT	SCMD	ISPCR
D8	CCON	CMOD	CCAPM0	CCAPM1	--	--	--	--
D0	PSW	--	--	--	--	--	P3KBIE	P1KBIE
C8	--	--	--	--	--	--	**CLRL**	**CHRL**
C0	--	--	--	--	ADCON0	--	ADCDH	CKCON0
B8	IP0L	SADEN	--	--	--	--	**RTCCR**	--
B0	P3	P3M0	P3M1	P4M0	PUCON0	--	**RTCTM**	IP0H
A8	IE	SADDR	--	--		EIE1	EIP1L	EIP1H
A0	P2	AUXR0	AUXR1	AUXR2	--	--	--	--
98	SCON	SBUF	--	--	--	--	--	--
90	P1	P1M0	P1AIO	--	--	P2M0	**BOREV**	PCON1
88	TCON	TMOD	TL0	TL1	TH0	TH1	SFIE	--
80	--	SP	DPL	DPH	--	--	--	PCON0
	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
PCON0	Power Control 0	87H	SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL	00010000B
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
SFIE	System Flag INT En.	8EH	UTIE	**SDIFIE**	--	**RTCFIE**	KBIFIE	--	BOF0IE	WDTFIE	00x00x00B
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
P1AIO	P1 Analog Input Only	92H	P17AIO	P16AIO	P15AIO	P14AIO	P13AIO	P12AIO	P11AIO	P10AIO	00000000B
P2M0	P2 Mode Register 0	95H	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0	00000000B
BOREV	Bit Order Reversed	96H	BOREV.7	BOREV.6	BOREV.5	BOREV.4	BOREV.3	BOREV.2	BOREV.1	BOREV.0	00000000B
PCON1	Power Control 1	97H	SWRF	EXRF	--	**RTCF**	KBIF	--	BOF0	WDTF	00x00x00B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	Ti	RI	00000000B
SBUF	Serial Buffer	99H									xxxxxxxxB
P2	*Port 2*	*A0H*	*P2.7*	*P2.6*	*P2.5*	*P2.4*	*P2.3*	*P2.2*	*P2.1*	*P2.0*	*11111111B*
AUXR0	Auxiliary Register 0	A1H	P40OC1	P40OC0	P40FD	**T0XL**	P1FS1	P1FS0	INT1H	INT0H	00000000B
AUXR1	Auxiliary Register 1	A2H	P3TWI	P4S0MI	P2PCA	**XTOR**	**STAF**	**STOF**	**BPOC1**	**BPOC0**	00000000B
AUXR2	Auxiliary Register 2	A3H	--	BTI	**URM0X3**	**SM3**	T1X12	T0X12	T1CKOE	T0CKOE	00000000B
IE	Interrupt Enable	A8H	EA	GF4	--	ES	ET1	EX1	ET0	EX0	00000000B
SADDR	Slave Address	A9H									00000000B
EIE1	Extended INT Enable 1	ADH	--	--	--	--	ESF	EPCA	EADC	--	xxxx000xB
EIP1L	Ext. INT Priority 1 Low	AEH	--	--	--	--	PSFL	PPCAL	PADCL	--	xxxx000xB
EIP1H	Ext. INT Priority 1 High	AFH	--	--	--	--	PSFH	PPCAH	PADCH	--	xxxx000xB
P3	*Port 3*	*B0H*	*P3.7*	*P3.6*	*P3.5*	*P3.4*	*P3.3*	*P3.2*	*P3.1*	*P3.0*	*11111111B*
P3M0	P3 Mode Register 0	B1H	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0	00000000B
P3M1	P3 Mode Register 1	B2H	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0	00000000B
P4M0	P4 Mode Register 0	B3H	--	--	--	--	--	--	P4M0.1	P4M0.0	xxxxxx00B
PUCON0	Pull-Up Control 0	B4H	--	PU40	PU21	PU20	PU11	PU10	--	--	00000000B
RTCTM	RTC Timer Register	B6H	RTCCS1	RTCCS0	RTCCT5	RTCCT4	RTCCT3	RTCCT2	RTCCT1	RTCCT0	01111111B
IP0H	Interrupt Priority 0 High	B7H	--	--	--	PSH	PT1H	PX1H	PT0H	PX0H	00000000B
IP0L	*Interrupt Priority Low*	*B8H*	--	--	--	*PSL*	*PT1L*	*PX1L*	*PT0L*	*PX0L*	*00000000B*
SADEN	Slave Address Mask	B9H									00000000B
RTCCR	RTC Control Register	BEH	RTCE	RTCOE	RTCRL5	RTCRL4	RTCRL3	RTCRL2	RTCRL1	RTCRL0	0x111111B
ADCON0	ADC Control 0	C4H	ADCEN	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0	00000000B
ADCDH	ADC Data High	C6H	ADCV.7	ADCV.6	ADCV.5	ADCV.4	ADCV.3	ADCV.2	ADCV.1	ADCV.0	00000000B
CKCON0	Clock Control 0	C7H	AFS	--	--	--	--	SCKS2	SCKS1	SCKS0	0xxxx001B
CLRL	PCA base timer Low Reload register	CEH	CLRL.7	CLRL.6	CLRL.5	CLRL.4	CLRL.3	CLRL.2	CLRL.1	CLRL.0	00000000B
CHRL	PCA base timer High Reload register	CFH	CHRL.7	CHRL.6	CHRL.5	CHRL.4	CHRL.3	CHRL.2	CHRL.1	CHRL.0	00000000B
PSW	*Program Status Word*	*D0H*	*CY*	*AC*	*F0*	*RS1*	*RS0*	*OV*	*F1*	*P*	*00000000B*
P3KBIE	P3 KBI Enable	D6H	P37KBIE	P36KBIE	P35KBIE	P34KBIE	**P41KBIE**	**P40KBIE**	P31KBIE	P30KBIE	00000000B
P1KBIE	P1 KBI Enable	D7H	P17KBIE	P16KBIE	P15KBIE	P14KBIE	P13KBIE	P12KBIE	P11KBIE	P10KBIE	00000000B
CCON	*PCA Control Reg.*	*D8H*	*CF*	*CR*	--	--	--	--	*CCF1*	*CCF0*	*00xxx000B*
CMOD	PCA Mode Reg.	D9H	CIDL	--	--	--	CPS2	CPS1	CPS0	ECF	0xxx0000B
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	WREN	NSW	ENW	CLW	WIDL	PS2	PS1	PS0	00000000B 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	--	--	--	--	--	MS2	MS1	MS0	xxxx0000B
SCMD	ISP Serial Command	E6H									xxxxxxxxB

ISPCR	ISP Control Register	E7H	ISPEN	SWBS	SWRST	CFAIL	--	--	--	--	0000xxxxB
P4	*Port 4*	*E8H*	--	--	--	--	--	--	*P4.1*	*P4.0*	*xxxxxx11B*
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
CCAP2L	PCA module2 capture Low	ECH									00000000B
CCAP3L	PCA module3 capture Low	EDH									00000000B
B	*B Register*	*F0H*	*F7H*	*F6H*	*F5H*	*F4H*	*F3H*	*F2H*	*F1H*	*F0H*	*00000000B*
PAOE		F1H									00011001B
PCAPWM0	PCA PWM0 Mode	F2H	--	--	--	--	--	P0INV	EPC0H	EPC0L	xxxxx000B
PCAPWM1	PCA PWM1 Mode	F3H	--	--	--	--	--	P1INV	EPC1H	EPC1L	xxxxx000B
PCAPWM2	PCA PWM2 Mode	F4H	PWM2	--	--	--	--	P2INV	EPC2H	EPC2L	0xxxx000B
PCAPWM3	PCA PWM3 Mode	F5H	PWM3	--	--	--	--	P3INV	EPC3H	EPC3L	0xxxx000B
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
CCAP2H	PCA Module2 capture High	FCH									00000000B
CCAP3H	PCA Module3 capture High	FDH									00000000B

3.3. Auxiliary SFR Map (Page P)

MG86FE/L508 has an auxiliary SFR page which is indexed by page P and the SFRs' access is a different way from standard 8051 SFR page. The registers in auxiliary SFR map are addressed by IFMT and SCMD like ISP/IAP access flow. Page P has 256 bytes space that can target to **5 physical bytes** and **5 logical bytes**. The 5 physical bytes include IAPLB, CKCON2, PCON2, SPCON0 and DCON0. The 5 logical bytes include PCON0, PCON1, CKCON0, RTCCR and WDTCR. Access on the 5 logical bytes gets the coherence content with the same SFR in Normal Page. Please refer Section "23 Page P SFR Access" for more detail information.

Table 3–3. Auxiliary SFR Map (Page P)

		0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8		--	--	--	--	--	--	--	--
F0		--	--	--	--	--	--	--	--
E8		--	--	--	--	--	--	--	--
E0		--	WDTCR	--	--	--	--	--	--
D8		--	--	--	--	--	--	--	--
D0		--	--	--	--	--	--	--	--
C8		--	--	--	--	--	--	--	--
C0		--	--	--	--	--	--	--	CKCON0
B8		--	--	--	--	--	--	RTCCR	--
B0		--	--	--	--	--	--	--	--
A8		--	--	--	--	--	--	--	--
A0		--	--	--	--	--	--	--	--
98		--	--	--	--	--	--	--	--
90		--	--	--	--	--	--	--	PCON1
88		--	--	--	--	--	--	--	--
80		--	--	--	--	--	--	--	PCON0
78		--	--	--	--	--	--	--	--
70		--	--	--	--	--	--	--	--
68		--	--	--	--	--	--	--	--
60		--	--	--	--	--	--	--	--
58		--	--	--	--	--	--	--	--
50		--	--	--	--	--	--	--	--
48		SPCON0	--	--	--	DCON0	--	--	--
40		CKCON2	--	--	--	PCON2	--	--	--
38		--	--	--	--	--	--	--	--
30		--	--	--	--	--	--	--	--
28		--	--	--	--	--	--	--	--
20		--	--	--	--	--	--	--	--
18		--	--	--	--	--	--	--	--
10		--	--	--	--	--	--	--	--
08		--	--	--	--	--	--	--	--
00		--	--	--	IAPLB	--	--	--	--
		0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F

3.4. Auxiliary SFR Bit Assignment (Page P)

Table 3–4. Auxiliary SFR Bit Assignment (Page P)

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	
Physical Bytes											
IAPLB	IAP Low Boundary	03H	IAPLB6	IAPLB5	IAPLB4	IAPLB3	IAPLB2	IAPLB1	IAPLB0	--	00011000B
CKCON2	Clock Control 2	40H	XTGS1	XTGS0	XTALE	IHRCOE	0	0	OSCS1	OSCS0	11010000B
PCON2	Power Control 2	44H	0	AWBOD0	0	0	0	0	BO0RE	**1**	00000001B
SPCON0	SFR Page Control 0	48H	RTCCTL	0	0	WRCTL	0	CKCTL0	PWCTL1	PWCTL0	00000000B
DCON0	Device Control 0	4CH	HSE	IAPO	0	0	0	0	RSTIO	0	10000010B
Logical Bytes											
PCON0	Power Control 0	87H	SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL	00010000B
PCON1	Power Control 1	97H	SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF	00x00x00B
RTCCR	RTC Control Register	BEH	RTCE	RTCOE	RTCRL5	RTCRL4	RTCRL3	RTCRL2	RTCRL1	RTCRL0	0x111111B
CKCON0	Clock Control 0	C7H	AFS	--	--	--	--	SCKS2	SCKS1	SCKS0	0xxxx001B
WDTCR	Watch-dog-timer Control register	E1H	WREN	NSW	ENW	CLW	WIDL	PS2	PS1	PS0	00000000B xxx0xxxxB

Sample Code of Page-P SFR write：

```
IFADRH = 0x00;
ISPCR = ISPEN;                    //enable IAP/ISP
IFMT = MS2;                       // Page-P write，IFMT =0x04
IFADRL = SPCON0;                  //Set Page-P SFR address
IFD |= CKCTL0;                    // set CKCTL0
SCMD = 0x46;                      //
SCMD = 0xB9;                      //
IFMT = Flash_Standby;             // IAP/ISP standby，IFMT =0x00
ISPCR &= ~ISPEN;
```

4. Pin Configurations

4.1. Package Instruction

Figure 4–1. MG86FE/L508AS28 Top View

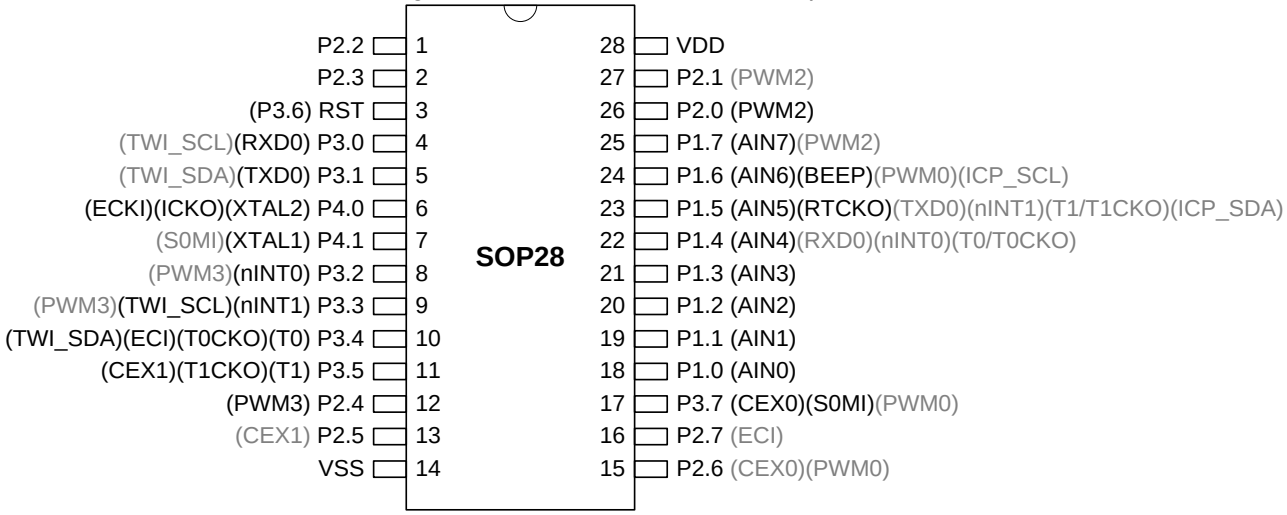


Figure 4–2. MG86FE/L508AS20 Top View

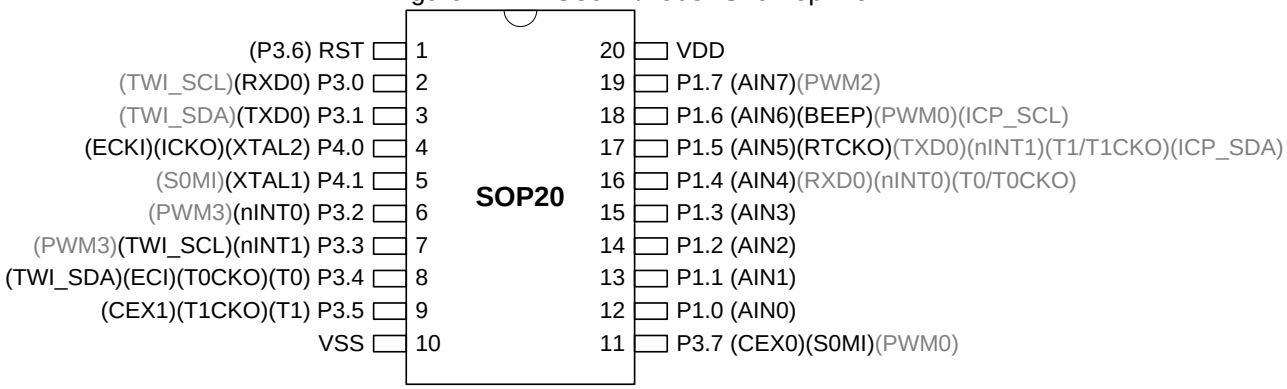
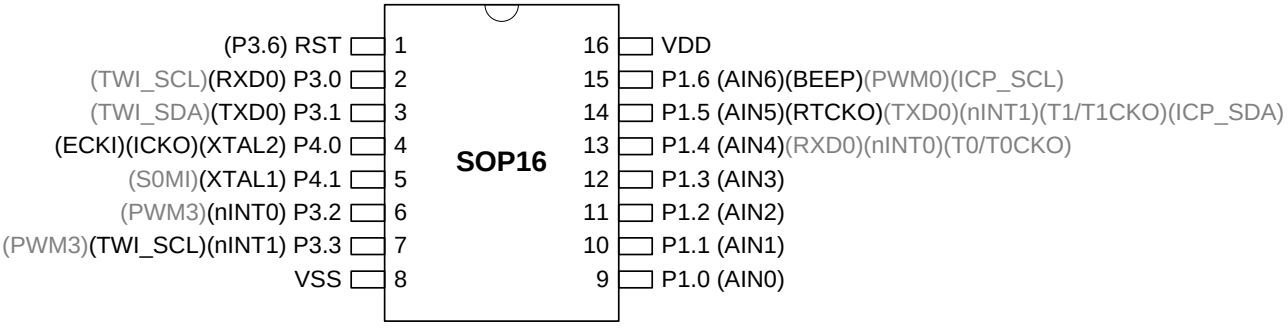


Figure 4–3. MG86FE/L508AS16 Top View



4.2. Pin Description

Table 4–1. Pin Description

MNEMONIC	PIN NUMBER			I/O TYPE	DESCRIPTION
	28-Pin SOP	20-Pin SOP	16-Pin SOP		
P1.0 (AIN0)	18	12	9	I/O	* Port 1.0. * AIN0: ADC channel-0 analog input.
P1.1 (AIN1)	19	13	10	I/O	* Port 1.1. * AIN1: ADC channel-1 analog input.
P1.2 (AIN2)	20	14	11	I/O	* Port 1.2. * AIN2: ADC channel-2 analog input.
P1.3 (AIN3)	21	15	12	I/O	* Port 1.3. * AIN3: ADC channel-3 analog input.
P1.4 (AIN4) (RXD0) (nINT0) (T0/T0CKO)	22	16	13	I/O	* Port 1.4. * AIN4: ADC channel-4 analog input. * RXD0: Alternate function for UART0 RXD0. * nINT0: Alternate function for nINT0 input. * T0/T0CKO: Alternate function for T0 input and T0CKO output.
P1.5 (AIN5) (RTCKO) (TXD3) (nINT1) (T1/T1CKO) **(ICP_SDA)**	23	17	14	I/O	* Port 1.5. * AIN5: ADC channel-5 analog input. * RTCKO: RTC clock output. * TXD3: Alternate function for UART0 TXD0. * nINT1: Alternate function for nINT1 input. * T1/T1CKO: Alternate function for T1 input and T1CKO output. * ICP_SDA: Serial data of ICP interface.
P1.6 (AIN6) (BEEP) (PWM0) **(ICP_SCL)**	24	18	15	I/O	* Port 1.6. * AIN6: ADC channel-6 analog input. * BEEP: Beeper output. * PWM0: Secondary output of CEX0 PWM output. * ICP_SCL: Serial clock of ICP interface.
P1.7 (AIN7) (PWM2)	25	19	--	I/O	* Port 1.7. * AIN7: ADC channel-7 analog input. * PWM2: Secondary output of PWM2 output.
P2.0 (PWM2)	26	--	--	I/O	* Port 2.0. * PWM2: PCA module-2 PWM2 output.
P2.1 (PWM2)	27	--	--	I/O	* Port 2.1. * PWM2: Third output of PWM2 output.
P2.2	1	--	--	I/O	* Port 2.2.
P2.3	2	--	--	I/O	* Port 2.3.
P2.4 (PWM3)	12	--	--	I/O	* Port 2.4. * PWM3: PCA module-3 PWM output.
P2.5 (CEX1)	13	--	--	I/O	* Port 2.5. * CEX1: Alternate CEX1 I/O for PCA module-1.
P2.6 (CEX0) (PWM0)	15	--	--	I/O	* Port 2.6. * CEX0: Alternate CEX0 I/O for PCA module 0. * PWM0: Secondary output of CEX0 PWM output.
P2.7 (ECI)	16	--	--	I/O	* Port 2.7. * ECI: Alternate ECI input for PCA.
P3.0 (RXD0) (TWI_SCL)	4	2	2	I/O	* Port 3.0. * RXD0: UART0 serial input port. * TWI_SCL: Alternate TWI SCL input.
P3.1 (TXD0) (TWI_SDA)	5	3	3	I/O	* Port 3.1. * TXD0: UART0 serial output port. * TWI_SDA: Alternate TWI SDA input.
P3.2 (nINT0) (PWM3)	8	6	6	I/O	* Port 3.2. * nINT0: external interrupt 0 input. * PWM3: Secondary output of PWM3 output.
P3.3 (nINT1) (TWI_SCL) (PWM3)	9	7	7	I/O	* Port 3.3. * nINT1: external interrupt 1 input. * TWI_SCL: SCL input for TWI Start/Stop detection. * PWM3: Third output of PWM3 output.
P3.4 (T0)	10	8	--	I/O	* Port 3.4. * T0: Timer/Counter 0 external input.

(T0CKO) (ECI) (TWI_SDA)					* T0CKO: programmable clock-out from Timer 0. * ECI: PCA external clock input. * TWI_SDA: SDA input for TWI Start/Stop detection.
P3.5 (T1) (T1CKO) (CEX1)	11	9	--	I/O	* Port 3.5. * T1: Timer/Counter 1 external input. * T1CKO: programmable clock-out from Timer 1. * CEX1: PCA module-1 external I/O.
P3.7 (CEX0) (S0MI) (PWM0)	17	11	--	I/O	* Port 3 bit-7. * CEX0: PCA module-0 external I/O. * S0MI: SPI Master Input on UART0. * PWM0: Secondary output of CEX0 PWM output.
P4.0 (XTAL2) (ECKI) (ICKO)	6	4	4	I/O O I O	* Port 4.0. * XTAL2: Output of on-chip crystal oscillating circuit. * ECKI: In external clock input mode, this is clock input pin. * ICKO: IHRCO clock output.
P4.1 (XTAL1) (S0MI)	7	5	5	I/O I I/O	* Port 4.1. * XTAL1: Input of on-chip crystal oscillating circuit. * S0MI: Alternate S0MI input.
RST (P3.6)	3	1	1	I I/O	* RST: External RESET input, high active. * Port 3.6. RST has the alternate function for P3.6.
VDD	28	20	16	P	Power supply input. 5V input for E-type. 3.3V input for L-type.
VSS	14	10	8	G	Ground, 0 V reference.

4.3. Alternate Function Redirection

Many I/O pins, in addition to their normal I/O function, also serve the alternate function for internal peripherals. For the peripherals UART, Timer 0, Timer1, nINT0, nINT1, S0MI and TWSI detection, Port 1, Port 2 and Port 3 serve the alternate function in the default state. However, the user may select other Port to serve their alternate function by setting the corresponding control bits P3TWI, P4S0MI and P2PCA in AUXR1 register. P1FS1~0 in AUXR0 register select the function swapped to other function. It is especially useful by software programming.

AUXR0: Auxiliary Register 0

SFR Page = Normal
SFR Address = 0xA1 RESET = 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P4.0 function configured control bit 1 and 0. The two bits only act when IHRCO is selected for system clock source. In crystal mode, XTAL2 and XTAL1 are the alternated function of P4.0 and P4.1. In external clock input mode, P4.0 is the dedicated clock input pin. In IHRCO operating, P4.0 provides the following selections for GPIO or clock source generator. When P40OC[1:0] index to non-P4.0 GPIO function, P4.0 will drive the IHRCO output to provide the clock source for other devices.

P40OC[1:0]	P4.0 function	I/O mode
00	P4.0	By P4M0.0
01	IHRCO	By P4M0.0
10	IHRCO/2	By P4M0.0
11	IHRCO/4	By P4M0.0

For clock-out on P4.0 function, it is recommended to set P4M0.0 to “1” which selects P4.0 as push-push output mode.

Bit 3~2: P1.4 and P1.5 alternated function selection.

P1FS[1:0]	P1.4	P1.5
00	P1.4	P1.5
01	Input for RXD0	Output for TXD0
10	Input for nINT0	Input for nINT1
11	Input for T0 or Output for T0CKO	Input for T1 or Output for T1CKO

AUXR1: Auxiliary Control Register 1

SFR Page = Normal
SFR Address = 0xA2 RESET = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	BPOC1	BPOC1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P3TWI, TWI interface in P3.0~P3.1.
0: Disable TWI function moved to P30 & P31.
1: Set TWI function in P3 as following definition.
‘TWI_SCL’ function in P3.3 is moved to P3.0.
‘TWI_SDA’ function in P3.4 is moved to P3.1.

Bit 6: P4S0MI, S0MI of Serial Port SPI master mode input in P4.1.
0: Disable S0MI function moved to P4.
1: Set S0MI in P4.1 as following definition.
‘S0MI’ function in P3.7 is moved to P4.1.

Bit 5: P2PCA, all PCA signals Port 2.
0: Disable PCA function moved to P4.

1: Set PCA in Port 2 as following definition.
‘ECI’ function in P3.4 is moved to P2.7.
‘CEX0’ function in P3.7 is moved to P2.6.
‘CEX1’ function in P3.5 is moved to P2.5.
‘CEX2’ function in P2.0 is kept in P2.0.
‘CEX3’ function in P2.4 is kept in P2.4.

5. 8051 CPU Function Description

5.1. CPU Register

PSW: Program Status Word

SFR Page = Normal
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 Page = Normal
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 Page = Normal
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 Page = Normal
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 Page = Normal
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 Page = Normal
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 **MG86FE/L508** 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 “28 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 **MG86FE/L508** 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 **MG86FE/L508**, 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.

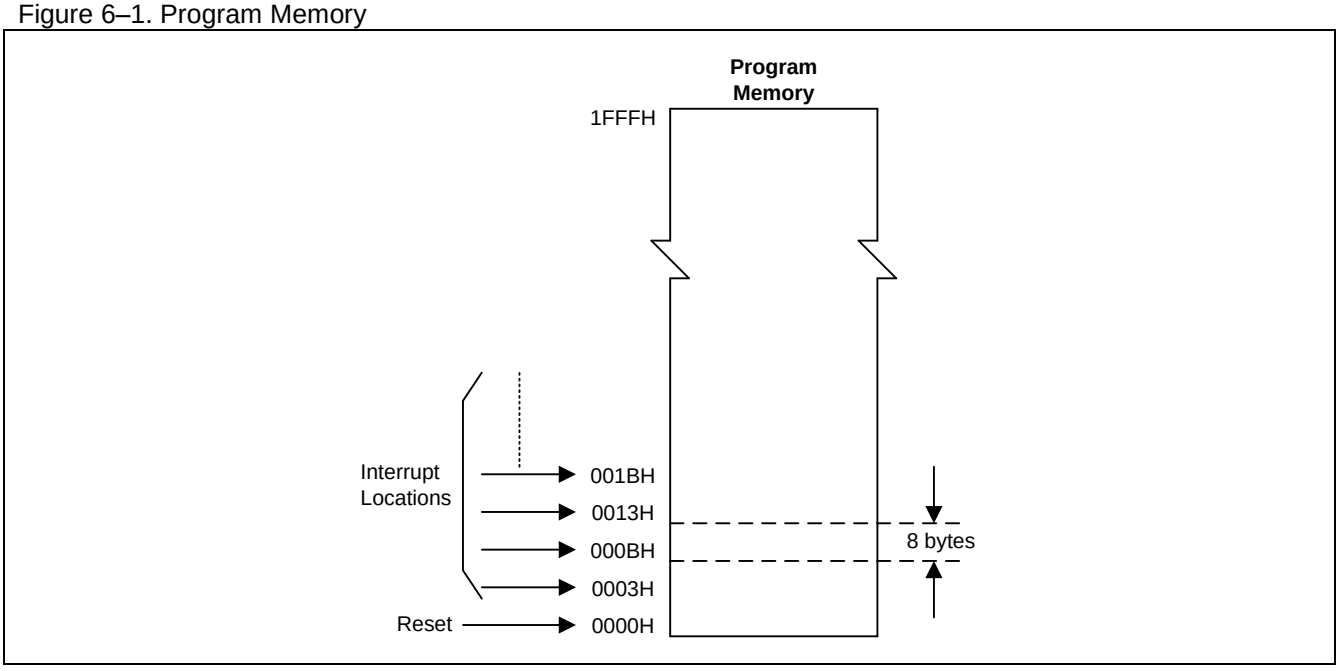
Data memory occupies a separate address space from program memory. In the **MG86FE/L508**, there is only a 256 bytes of internal scratch-pad RAM and has no any expanded RAM(XRAM).

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 and external data memory spaces available to the **MG86FE/L508** 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

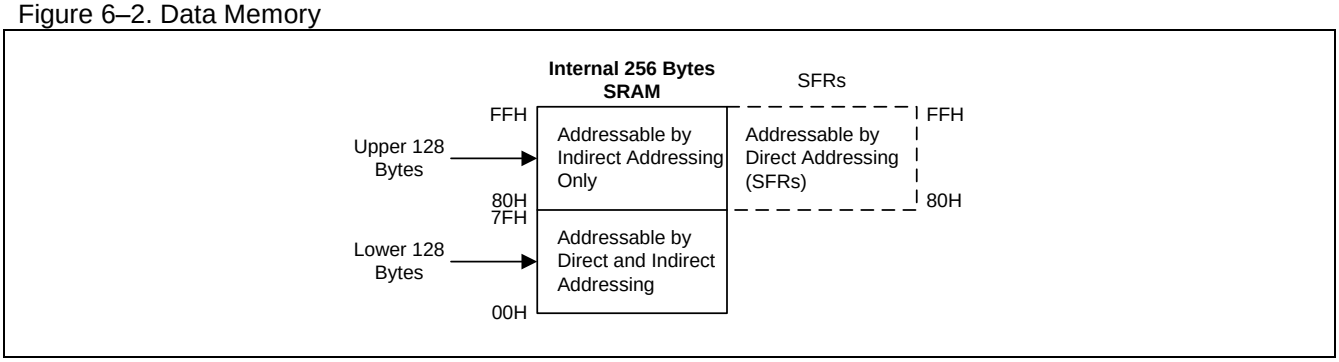


Figure 6–3. Lower 128 Bytes of Internal RAM

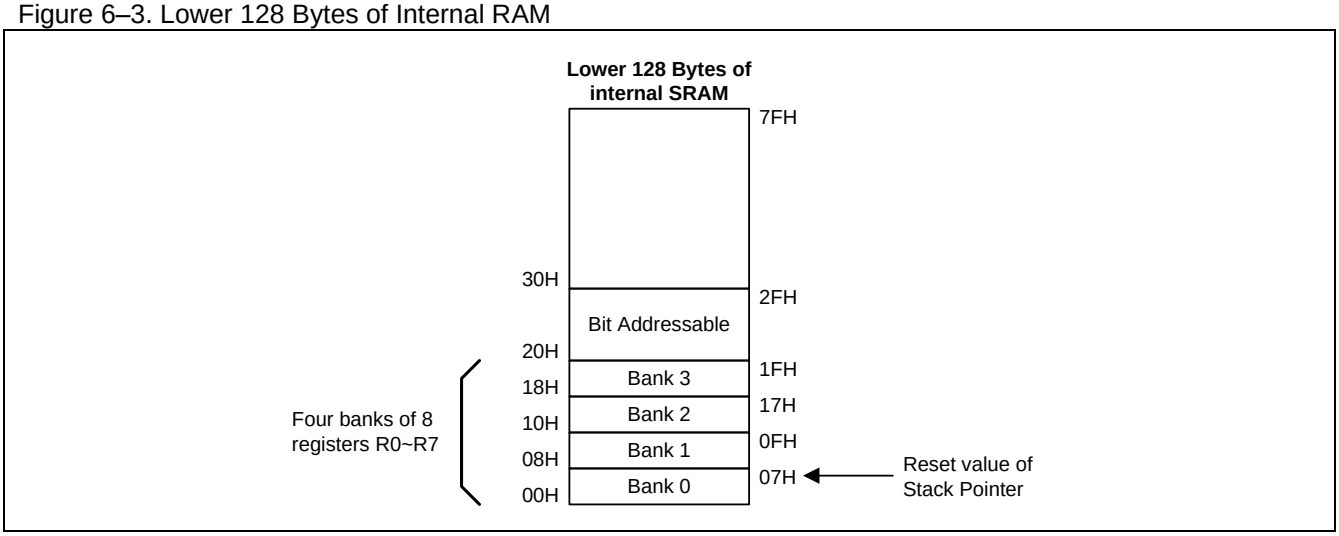
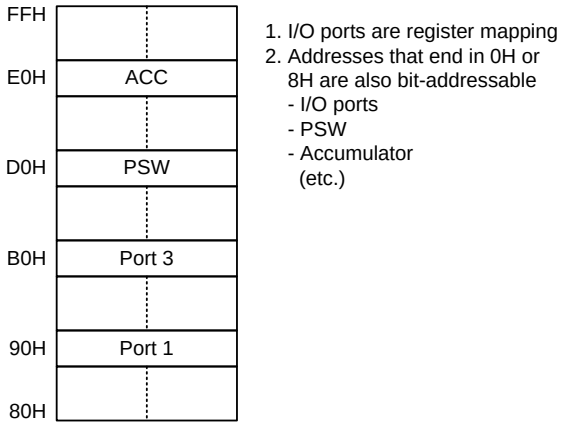


Figure 6–4. SFR Space



6.3. Declaration Identifiers in a C51-Compiler

The declaration identifiers in a C51-compiler for the various **MG86FE/L508** 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 **MG86FE/L508**. **MG86FE/L508** does not support external memory access and MOVX instruction.

8. System Clock

There are four clock sources for the system clock: Internal High-frequency RC Oscillator (IHRCO), external crystal oscillator, Internal Low-frequency RC Oscillator (ILRCO) and External Clock Input. Figure 8–1 shows the structure of the system clock in **MG86FE/L508**.

The **MG86FE/L508** always boots from IHRCO on 24MHz with divided 2 on system clock and reserves crystal pads as P4.0/P4.1 GPIO function. Software can select the one of the four clock sources by application required and switches them on the fly. But software needs to settle the clock source stably before clock switching. If software selects external crystal mode, port pin of P4.0 and P4.1 will be assigned to XTAL2 and XTAL1. And P4.0/P4.1 GPIO function will be inhibited. In external clock input mode (ECKI), the clock source comes from P4.0 input and P4.1 still serves the GPIO function.

After set XTALE (CKCON2.4) to enable external crystal oscillating, XTOR (AUXR1.4) will be set by hardware to indicate the crystal oscillating is stable for software to switch the OSCin on it. XTOR is read only. MCU must poll this bit before switching the crystal oscillator as system clock source.

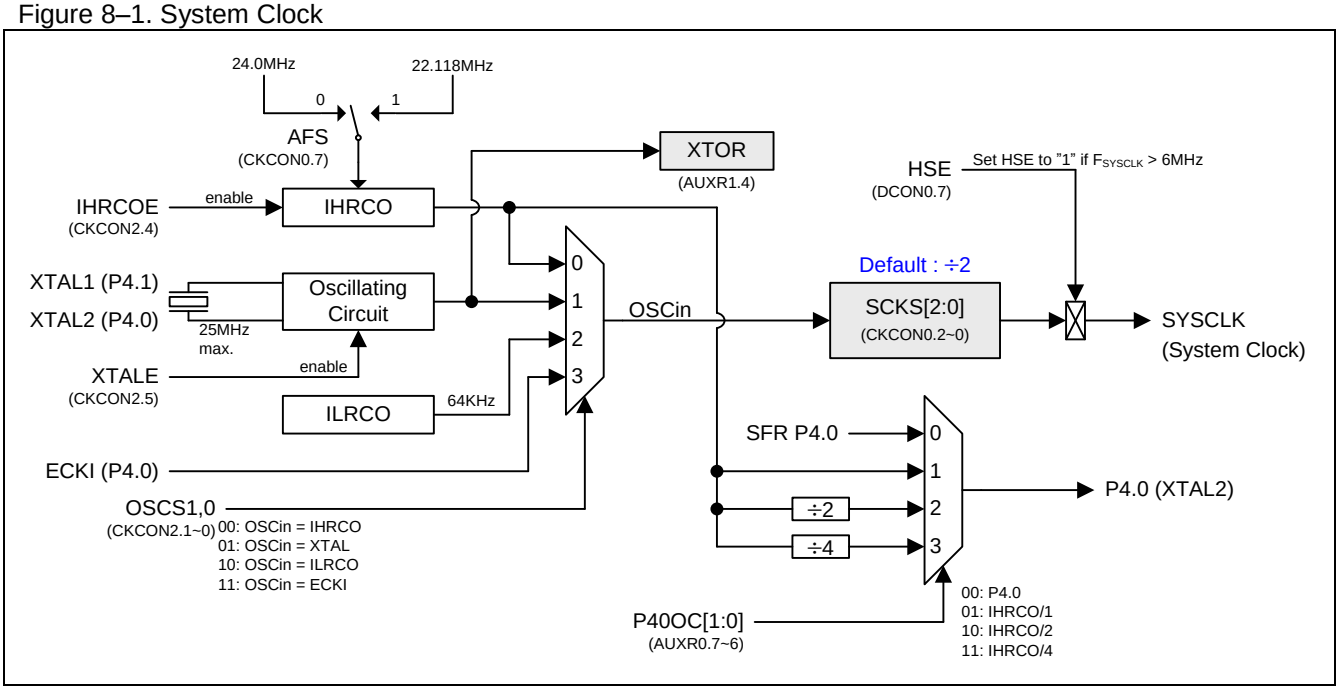
The built-in IHRCO provides two kinds of frequency for software selected. Another frequency is 22.118MHz by software setting AFS on CKCON0.7. Both of 24MHz and 22.118 MHz in IHRCO provide high precision frequency for system clock source. To find the detailed IHRCO performance, please refer Section "27.4 IHRCO Characteristics"). In IHRCO mode, P4.0 can be configured to *IHRCO* output or *IHRCO*/2 and *IHRCO*/4 for system application.

The system clock, *SYSCLK*, is obtained from one of these four clock sources through the clock divider, as shown in Figure 8–1. The user can program the divider control bits SCKS2~SCKS0 (in CKCON0 register) to get the desired system clock. The default system clock divider is set to "**/2**" in **MG86FE/L508** after power on or reset.

8.1. Clock Structure

Figure 8–1 presents the principal clock systems in the **MG86FE/L508**. The system clock can be sourced by the external oscillator circuit or either internal oscillator.

Figure 8–1. System Clock



8.2. Clock Register

CKCON0: Clock Control Register 0

SFR Page = Normal & Page P

SFR Address = 0xC7 RESET = 0xxx-x001

7	6	5	4	3	2	1	0
AFS	0	0	0	0	SCKS2	SCKS1	SCKS0
R/W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: AFS, Alternated Frequency Selection.
0: Select IHRCO to output 24MHz.
1: Select IHRCO to output 22.118MHz.

Bit 6~3: Reserved. Software must write “0” on these bits when CKCON0 is written.

Bit 2~0: SCKS2 ~ SCKS0, programmable System Clock Selection. The default value of SCKS[2:0] is set to “001” to select system clock on OSCin/2.

SCKS[2:0]	System Clock
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

CKCON2: Clock Control Register 2

SFR Page = **Page P Only**

SFR Address = 0x40 RESET = 1101-xx00

7	6	5	4	3	2	1	0
XTGS1	XTGS0	XTALE	IHRCOE	**0**	**0**	OSCS1	OSCS0
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**R/W**	**R/W**

Bit 7~6: XTGS1~XTGS0, OSC Driving control Register.

XTGS1, XTGS0	Gain Define
0, 0	Gain for 32.768K
1, 1	Gain for 2MHz ~ 25MHz
Others	Reserved

Bit 5: XTALE, external Crystal (XTAL) Enable.
0: Disable XTAL oscillating circuit. In this case, XTAL2 and XTAL1 behave as Port 4.0 and Port 4.1.
1: Enable XTAL oscillating circuit. If this bit is set by CPU software, software polls the **XTOR** (AUXR1.4) **true** to indicate the crystal oscillator is ready for clock selected.

Bit 4: IHRCOE, Internal High frequency RC Oscillator Enable. The default value is set for MCU clock source.
0: Disable internal high frequency RC oscillator.
1: Enable internal high frequency RC oscillator. If this bit is set by CPU software, it needs **32 us** to have stable output after IHRCOE enabled.

Bit 3~2: Reserved. Software must write “0” on these bits when CKCON2 is written.

Bit 1~0: OSCS[1:0], OSCin source selection. The default selection of OSCin is IHRCO.

OSCS[1:0]	OSCin source Selection
0 0	IHRCO
0 1	XTAL
1 0	ILRCO
1 1	ECKI, External Clock Input (P4.0) as OSCin.

AUXR0: Auxiliary Register 0

SFR Page = Normal

SFR Address = 0xA1 RESET = 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P4.0 function configured control bit 1 and 0. The two bits only act when IHRCO is selected for system clock source. In crystal mode, XTAL2 and XTAL1 are the alternated function of P4.0 and P4.1. In external clock input mode, P4.0 is the dedicated clock input pin. In IHRCO operating, P4.0 provides the following selections for GPIO or clock source generator. When P40OC[1:0] index to non-P4.0 GPIO function, P4.0 will drive the IHRCO output to provide the clock source for other devices.

P40OC[1:0]	P4.0 function	I/O mode
00	P4.0	By P4M0.0
01	IHRCO/1	By P4M0.0
10	IHRCO/2	By P4M0.0
11	IHRCO/4	By P4M0.0

For clock-out on P4.0 function, it is recommended to set P4M0.0 to “1” which selects P4.0 as push-push output mode.

Bit 5: P40FD, P4.0 Fast Driving.
0: P4.0 output with default driving.
1: P4.0 output with fast driving enabled. If P4.0 is configured to clock output, enable this bit when P4.0 output frequency is more than 12MHz at 5V application or more than 6MHz at 3V application.

AUXR1: Auxiliary Control Register 1

SFR Page = Normal

SFR Address = 0xA2 RESET = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	BPOC1	BPOC0
R/W	**R/W**	**R/W**	**R**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1: XTOR, Crystal Oscillating Ready. Read Only.
0: Crystal Oscillating not Ready.
1: Crystal Oscillating Ready. When XTALE is enabled, XTOR reports the crystal oscillator reached start-up count.

DCON0: Device Control 0

SFR Page = Page P only

SFR Address = 0x4C RESET = 10xx-xx1x

7	6	5	4	3	2	1	0
HSE	IAPO	0	0	0	0	RSTIO	0
W	**W**	**W**	**W**	**W**	**W**	**W**	**W**

Bit 7: HSE, High Speed operation Enable.
0: Select MCU running in low speed mode which is slow down internal circuit to reduce power consumption.
1: Enable MCU full speed operation if F_{SYSCLK} > 6MHz.

8.3. Clock Sample Code

(1) Required function: Switch IHRCO from default 24MHz to 22.118MHz

Assembly Code Example:

```
ORL     CKCON0,#(AFS)                   ; Select IHRCO to output 22.118MHz
```

C Code Example:

```
CKCON0 |= AFS;                          // Select IHRCO to output 22.1184MHz.
```

(2). Required Function: Switch SYSCLK to OSCin/1 (default is OSCin/2)

Assembly Code Example:

```
ANL     CKCON0,#(AFS)                   ; Set SCKS[2:0] = 0 to select OSCin/1
```

C Code Example:

```
CKCON0 &= ~(SCKS2 | SCKS1 | SCKS0);     // System clock divider /1
                                        // SCKS[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
```

(3). Required Function: Select XTAL as OSCin source when MCU using IHRCO or ILRCO (default is IHRCO)

Assembly Code Example:

```
    MOV     IFADRL,#(CKCON2)            ; Index Page-P address to CKCON2
    CALL    _page_p_sfr_read            ; Read CKCON2 data

    ORL     IFD,#(XTGS1 | XTGS0 | XTALE)  ; Enable XTALE and set to high gain (for non-32768Hz application)
                                        ; For 32768Hz XTAL, set XTGS1 = XTGS0 = 0
    CALL    _page_p_sfr_write           ; Write data to CKCON2, SYSCLK must be less than 25MHz

check_XTOR:                             ; Check XTAL oscillating ready
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR            ; Waiting for XTOR(AUXR1.4) true

    ANL     IFD,#~(OSCS1 | OSCS0)       ; Switch OSCin source to XTAL.
    ORL     IFD,#(OSCS0)
    CALL    _page_p_sfr_write           ; Write data to CKCON2

    ANL     IFD,#~(IHRCOE)              ; Disable IHRCO if MCU is switched from IHRCO
    CALL    _page_p_sfr_write           ; Write data to CKCON2
```

C Code Example:

```
    IFADRL = CKCON2;                    // Index Page-P address to CKCON2
    page_p_sfr_read();                  // Read CKCON2 data

    IFD |= XTGS1 | XTGS0 | XTALE;       // Enable XTALE and set to high gain (for non-32768Hz application)
                                        // For 32768Hz XTAL, set XTGS1 = XTGS0 = 0
    page_p_sfr_write ();                // Write data to CKCON2, SYSCLK must be less than 25MHz

    while(AUXR1 & XTOR == 0x00);        // Check XTAL oscillating ready
                                        // Waiting for XTOR(AUXR1.4) true

    IFD &= ~(OSCS1 | OSCS0);            // Switch OSCin source to XTAL.
```

```
IFD |= OSCS0;
page_p_sfr_write ();                    // Write data to CKCON2

IFD &= ~IHRCOE;                         // Disable IHRCO if MCU is switched from IHRCO
page_p_sfr_write();                     // Write data to CKCON2
```

(4). Required Function: Select ILRCO as OSCin source when MCU using IHRCO, ECKI or XTAL (default is IHRCO)

Assembly Code Example:

```
MOV     IFADRL,#(CKCON2)            ; Index Page-P address to CKCON2
CALL    _page_p_sfr_read            ; Read CKCON2 data

ANL     IFD,#~(OSCS1 | OSCS0)       ; Switch OSCin source to ILRCO
ORL     IFD,#(OSCS1)
CALL    _page_p_sfr_write           ; Write data to CKCON2

ANL     IFD,#~(XTALE | IHRCOE)      ; Disable XTAL and IHRCO
CALL    _page_p_sfr_write           ; Write data to CKCON2

MOV     IFADRL,#(DCON0)             ; Index Page-P address to DCON0
CALL    _page_p_sfr_read            ; Read DCON0 data

ANL     IFD,#~(HSE)                 ; Disable HSE when SYSCLK ≤ 6MHz for power saving
CALL    _page_p_sfr_write           ; Write data to DCON0
```

C Code Example:

```
IFADRL = CKCON2;                    // Index Page-P address to CKCON2
page_p_sfr_read();                  // Read CKCON2 data.

IFD = ~(OSCS1 | OSCS0);             // Switch OSCin source to ILRCO
IFD |= OSCS1;
page_p_sfr_write();                 // Write data to CKCON2

IFD &= ~(XTALE | IHRCOE);           // Disable XTAL and IHRCO
page_p_sfr_write();                 // Write data to CKCON2

IFADRL = DCON0;                     // Index Page-P address to DCON0
page_p_sfr_read();                  // Read DCON0 data

IFD &= ~HSE;                        // Disable HSE when SYSCLK ≤ 6MHz for power saving
page_p_sfr_write();                 // Write data to DCON0
```

(5). Required Function: Select ECKI as OSCin source when MCU using IHRCO or ILRCO (default is IHRCO)

Assembly Code Example:

```
MOV     IFADRL,#(CKCON2)            ; Index Page-P address to CKCON2
CALL    _page_p_sfr_read            ; Read CKCON2 data

ORL     IFD,#(OSCS1 | OSCS0)        ; Switch OSCin source to ECKI
CALL    _page_p_sfr_write           ; Write data to CKCON2, SYSCLK must be less than 25MHz

ANL     IFD,#~(XTALE | IHRCOE)      ; Disable IHRCO & XTAL
CALL    _page_p_sfr_write           ; Write data to CKON2
```

C Code Example:

```
IFADRL = CKCON2;                    // Index Page-P address to CKCON2
page_p_sfr_read();                  // Read CKCON2 data.

IFD |= OSCS1 | OSCS0;               // Switch OSCin source to ECKI
page_p_sfr_write ();                // Write data to CKCON2, SYSCLK must be less than 25MHz
```

```
IFD &= ~(XTALE | IHRCOE);                  //Disable IHRCO and XTAL
page_p_sfr_write ();                       // Write data to CKCON2
```

(6). Required Function: Select IHRCO as OSCin source when MCU using ILRCO, ECKI or XTAL

Assembly Code Example:

```
MOV      IFADRL,#(CKCON2)          ; Index Page-P address to CKCON2
CALL     _page_p_sfr_read          ; Read CKCON2 data

ORL      IFD,#(IHRCOE)             ; Enable IHRCO
CALL     _page_p_sfr_write         ; Write data to CKCON2

Delay_32us

ANL      IFD,#~(OSCS1 | OSCS0)     ; Switch OSCin source to IHRCO
CALL     _page_p_sfr_write         ; Write data to CKCON2

ANL      IFD,#~(XTALE)             ; Disable XTAL
CALL     _page_p_sfr_write         ; Write data to CKCON2
```

C Code Example:

```
IFADRL = CKCON2;                   // Index Page-P address to CKCON2
page_p_sfr_read();                 // Read CKCON2 data.

IFD |= IHRCOE;                     // Enable IHRCO
page_p_sfr_write();                // Write data to CKCON2

Delay 32us

IFD &= ~(OSCS1 | OSCS0);           // Switch OSCin source to IHRCO
page_p_sfr_write();                // Write data to CKCON2

IFD &= ~ XTALE;                    // Disable XTAL
page_p_sfr_write();                // Write data to CKCON2
```

(7). Required Function: Output IHRCO frequency on P4.0

Assembly Code Example:

```
MOV      P4M0,#P4M00               ; Set P4.0 to push-pull output mode
ANL      AUXR0,#~(P40OC1|P40OC0)    ; Switch P4.0 to GPIO function
ORL      AUXR0,#(P40OC0|P4FD)      ; P4.0 = IHRCO Frequency + Pin fast driving
                                   ; P40OC[1:0]   |   P4.0
                                   ; 00           |   GPIO
                                   ; 01           |   IHRCO/1
                                   ; 10           |   IHRCO/2
                                   ; 11           |   IHRCO/4
```

C Code Example:

```
P4M0 |= P4M00;                     // P4.0 select push-pull output mode.
AUXR0 &= ~(P40OC0 | P40OC1);        // Switch P4.0 to GPIO function
AUXR0 |= (P40OC0 | P4FD);           // P4.0 output IHRCO/1
//  AUXR0 = P40OC1|P4FD;            // P4.0 output IHROC/2
//  AUXR0 = P40OC1|P40OC0|P4FD;     // P4.0 output IHRCO/4
```

9. Watch Dog Timer (WDT)

9.1. WDT Structure

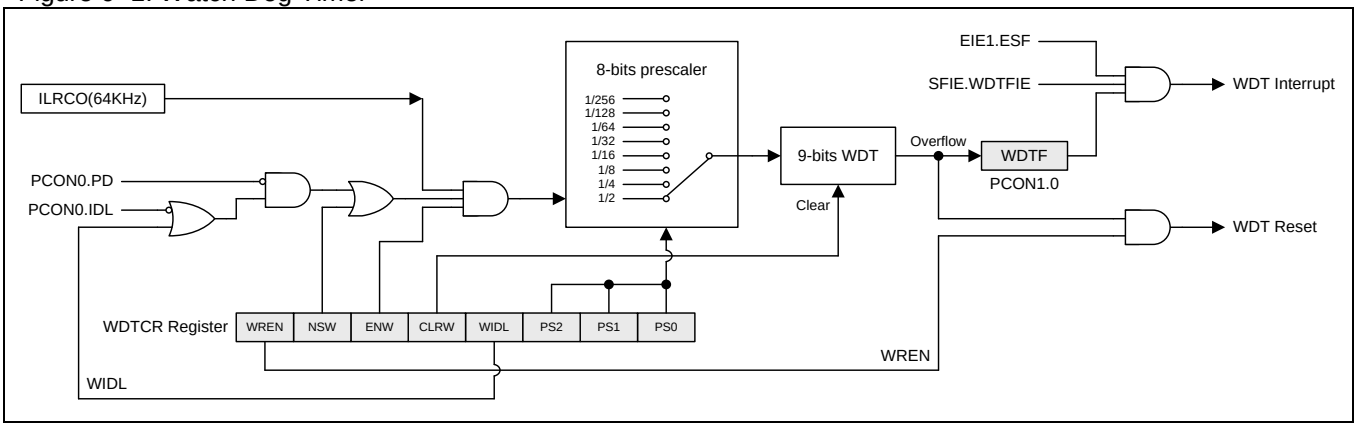
The Watch-dog Timer (WDT) is intended as a recovery method in situations where the CPU may be subjected to software upset. The WDT consists of a 9-bit free-running counter, a 8-bit prescaler and a control register (WDTCR). Figure 9–1 shows the WDT structure in **MG86FE/L508**.

When WDT is enabled, it derives its time base from the 64KHz ILRCO. The WDT overflow will set the WDTF on PCON1.0 which can be configured to generate an interrupt by enabled WDTFIE (SFIE.0) and enabled ESF (EIE1.3). The overflow can also trigger a system reset when WREN (WDTCR.7) is set. 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 or page-p SFR over-write on ENW, which will clear the ENW bit. The WDTCR register will keep the previous programmed value unchanged after external reset (RST-pin), software reset and WDT reset.

WREN, NSW and ENW are implemented to one-time-enabled function, only writing “1” valid in general SFR page. Page-P SFR Access on WDTCR can disable WREN, NSW and ENW, writing “0” on WDTCR.7~5. Please refer Section “9.3 WDT Register” and Section “23 Page P SFR Access” 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, the ILRCO won’t stop if the NSW (WDTCR.6) is enabled. That lets WDT keep counting even in Power down mode (Watch Mode). After WDT overflows, it will wake up the CPU from interrupt or reset by software configured.

9.3. WDT Register

WDTCR: Watch-Dog-Timer Control Register

SFR Page = Normal & Page P

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

7	6	5	4	3	2	1	0
WREN	NSW	ENW	CLRW	WIDL	PS2	PS1	PS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: WREN, WDT Reset Enable. The initial value can be changed by hardware option, WRENO.
0: The overflow of WDT does not set the WDT reset. The WDT overflow flag, WDTF, may be polled by software or trigger an interrupt.
1: The overflow of WDT will cause a system reset. Once WREN has been set, it can not be cleared by software in normal page. **In page P, software can modify it to "0" or "1".**

Bit 6: NSW. Non-Stopped WDT. The initial value can be changed by hardware option, NSWDT.
0: WDT stop counting while the MCU is in power-down mode.
1: WDT always keeps counting while the MCU is in power-down mode (Watch Mode) or idle mode. Once NSW has been set, it can not be cleared by software in normal page. **In page P, software can modify it to "0" or "1".**

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 in normal page. **In Page P, software can modify it as "0" or "1".**

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 9-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
0 0 0	2	15 ms
0 0 1	4	31 ms
0 1 0	8	62 ms
0 1 1	16	124 ms
1 0 0	32	248 ms
1 0 1	64	496 ms
1 1 0	128	992 ms
1 1 1	256	1.984 S

PCON1: Power Control Register 1

SFR Page = Normal & Page P

SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF
R/W	R/W	W	R/W	R/W	W	R/W	R/W

Bit 0: WDTF, WDT overflow flag.
0: This bit must be cleared by software writing **"1"** on it. Software writing ":0" is no operation.
1: This bit is only set by hardware when WDT overflows. Writing **"1"** on this bit will clear WDTF.

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 WRENO, NSWDT, HWENW, HWWIDL and HWPS[2:0], 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 WRENO into WREN bit, (3) load NSWDT into NSW bit, (4) load HWWIDL into WIDL bit, and (5) 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, even though access through Page-P SFR mechanism.

WRENO:
- ☑: Enabled. Set WDTCR.WREN to enable a system reset function by WDTF.
- ☐: Disabled. Clear WDTCR.WREN to disable the system reset function by WDTF.

NSWDT: Non-Stopped WDT
- ☑: Enabled. Set WDTCR.NSW to enable the WDT running in power down mode (watch mode).
- ☐: Disabled. Clear WDTCR.NSW to disable the WDT running in power down mode (disable Watch mode).

HWENW: Hardware loaded for “ENW” of WDTCR.
- ☑: Enabled. Enable WDT and load the content of WRENO, NSWDT, 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 period to 248ms

Assembly Code Example:

```
ANL     PCON1,#(WDTF)                   ; Clear WDTF flag (write “1”)
MOV     WDTCR,#(ENW | CLRW | PS2)       ; Enable WDT counter and set WDT period to 248ms
```

C Code Example:

```
PCON1 &= WDTF;                          // Clear WDT flag (write “1”)
WDTCR = (ENW | CLRW | PS2);             // Enable WDT counter and set WDT period to 248ms
                                        // PS[2:0]   | WDT period selection
                                        //      0    | 15ms
                                        //      1    | 31ms
                                        //      2    | 62ms
                                        //      3    | 124ms
                                        //      4    | 248ms
                                        //      5    | 496ms
                                        //      6    | 992ms
                                        //      7    | 1.984s
```

(2) Required function: How to Disable WDT

Assembly Code Example:

```
MOV     IFD,WDTCR                       ; Read WDTCR data
ANL     IFD,#~(ENW)                     ; Clear ENW to disable WDT

MOV     IFADRL,#(WDTCR_P)               ; Index Page-P address to WDTCR_P
CALL    _page_p_sfr_write               ; Write data to WDTCR
```

C Code Example:

```
IFD = WDTCR;                            // Read WDTCR data
IFD &= ~ENW;                            // Clear ENW to disable WDT

IFADRL = WDTCR_P;                       // Index Page-P address to WDTCR_P
page_p_sfr_write();                     // Write data to WDTCR
```

(3). Required Function: Enable WDT reset function and select WDT period to 62ms

Assembly Code Example:

```
ANL     PCON1,#(WDTF)                   ; Clear WDTF flag (write “1”)
MOV     WDTCR,#(WREN | CLRW | PS1)      ; Enable WDT reset function and set WDT period to 62ms

ORL     WDTCR,#(ENW)                    ; Enable WDT counter, WDT running
```

C Code Example:

```
PCON1 &= WDTF;                          // Clear WDTF flag (write “1”)
WDTCR = WREN | CLRW | PS1;              // Enable WDT reset function and set WDT period to 62ms

WDTCR |= ENW;                           // Enable WDT counter, WDT running.
```

(4). Required Function: Enable protected write for WDTCR

Assembly Code Example:

```
ANL     PCON1,#(WDTF)                ; Clear WDTF flag (write “1”)
MOV     WDTCR,#(ENW | CLRW | PS2)    ; Enable WDT counter and set WDT period to 248ms

MOV     IFADRL,#(SPCON0)             ; Index Page-P address to SPCON0
CALL    _page_p_sfr_read             ; Read SPCON0 data

ORL     IFD,#(WRCTL)                 ; Enable protected write for WDTCR
CALL    _page_p_sfr_write            ; Write data to SPCON0

MOV     IFD,WDTCR                    ; Read WDTCR data
ORL     IFD,#(CLRW)                  ; Enable CLRW

MOV     IFADRL,#(WDTCR_P)            ; Index Page-P address to WDTCR_P
CALL    _page_p_sfr_write            ; Write data to WDTCR to clear WDT counter
```

C Code Example:

```
PCON1 &= WDTF;                       // Clear WDTF flag (write “1”)
WDTCR = ENW | CLRW | PS2;            // Enable WDT counter and set WDT period to 248ms

IFADRL = SPCON0;                     // Index Page-P address to SPCON0
page_p_sfr_read();                   // Read SPCON0 data

IFD |= WRCTL;                        // Enable protected write for WDTCR
page_p_sfr_write();                  // Write data to SPCON0

IFD = WDTCR;                         // Read WDTCR data
IFD |= CLRW;                         // Enable CLRW

IFADRL = WDTCR_P;                    // Index Page-P address to WDTCR_P
page_p_sfr_write();                  // Write data to WDTCR to clear WDT counter
```

10. Real-Time-Clock(RTC)/System-Timer

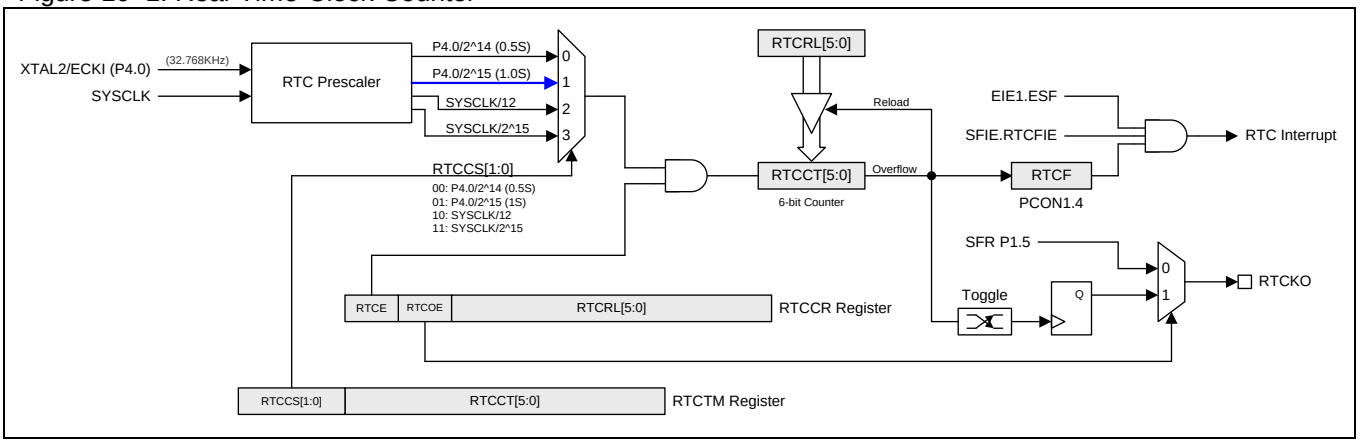
10.1. RTC Structure

The **MG86FE/L508** has a simple Real-Time clock that allows a user to continue running an accurate timer while the rest of the device is powered-down. The Real-Time clock can be a wake-up or an interrupt source. The Real-Time clock is a 21-bit up counter comprised of a 14/15-bit prescaler and a 6-bit loadable up counter. When it overflows, the counter will be reloaded again and the RTCF flag will be set. The clock source for this prescaler can be either the system clock (SYSCLK) or the XTAL oscillator, provided that the XTAL oscillator is not being used as the system clock. Figure 10–1 shows the RTC structure in **MG86FE/L508**.

The 32.768KHz crystal for the RTC module input will provide a programmable overflow period for 0.5S to 64S. The counter also provides a timer function with the clock derived from SYSCLK/12 or SYSCLK/2^15 for a short timer function or a long system timer function. The maximum overflow period for the system timer function is SYSCLK/2^21.

If the XTAL oscillator is used as the system clock, then the RTC still uses P4.0 input as its clock source. Only power-on reset will reset the Real-Time clock and its associated SFRs to the default state.

Figure 10–1. Real-Time-Clock Counter



10.2. RTC Register

RTCCR: Real-Time-Clock Control Register

SFR Page = Normal & Page P

SFR Address = 0xBE POR = 0011-1111

7	6	5	4	3	2	1	0
RTCE	RTCOE	RTCRL.5	RTCRL.4	RTCRL.3	RTCRL.2	RTCRL.1	RTCRL.0
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: RTCE, RTC Enable.
0: Stop RTC Counter, RTCCT.
1: Enable RTC Counter and set RTCF when RTCCT overflows. When RTCE is set, CPU can not access RTCTM. RTCTM must be accessed in RTCE cleared.

Bit 6: RTCOE, RTC Output Enabled. The frequency of RTCKO is (RTC overflow rate)/2.
0: Disable the RTCKO output.
1: Enable the RTCKO output.

Bit 5~0: RTCRL[5:0], RTC counter reload value register. This register is accessed by CPU and the content in the register is reloaded to RTCCT when RTCCT overflows.

RTCTM: Real-Time-Clock Timer Register

SFR Page = Normal

SFR Address = 0xB6 POR = 0111-1111

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

Bit 7~6: RTCCS.1~0, RTC Clock Selection. Default is “01”.

RTCCS[1:0]	Clock Source	RTC Interrupt Duration	Min. Step
0 0	P4.0/2^14	0.5S ~ 32S when P4.0 = 32768Hz	0.5S
0 1	P4.0/2^15	1S ~ 64S when P4.0 = 32768Hz	1S
1 0	SYSCLK/12	1us ~ 64us when SYSCLK = 12MHz	1us
1 1	SYSCLK/2^15	2.73ms ~ 174.72ms when SYSCLK = 12MHz	2.73ms

Bit 5~0: RTCCT[5:0], RTC counter register. It is a counter for RTC function or System Timer function by different clock source selection on RTCCS[1:0]. When the counter overflows, it sets the RTCF flag which shares the system flag interrupt when RTCFIE is enabled. The maximum RTC overflow period is 64 seconds.

PCON1: Power Control Register 1

SFR Page = Normal & Page P

SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	**RTCF**	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 4: RTCF, RTC overflow flag.
0: This bit must be cleared by software writing **“1”** on it. Software writing “:0” is no operation.
1: This bit is only set by hardware when RTCCT overflows. Writing **“1”** on this bit will clear RTCF.

SFIE: System Flag Interrupt Enable Register

SFR Page = Normal

SFR Address = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	**RTCFIE**	KBFIE	--	BOF0IE	WDTFIE
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 4: RTCFIE, Enable RTCF (PCON1.4) Interrupt.
0: Disable RTCF interrupt.
1: Enable RTCF interrupt. If enabled, RTCF will wake up CPU in Idle mode or power-down mode.

10.3. RTC Sample Code

(1). Required Function: Enable XTAL 32.768KHz oscillation for RTC application

Assembly Code Example:

```
    MOV     IFADRL,#(CKCON2)          ; Index Page-P address to CKCON2
    CALL    _page_p_sfr_read          ; Read CKCON2 data

    ANL     IFD,#~(XTGS1 | XTGS0)     ; Set XTAL to low gain for 32.768KHz
    ORL     IFD,#(XTALE)              ; Enable XTAL oscillating
    CALL    _page_p_sfr_write         ; Write data to CKCON2

check_XTOR_0:                         ; Check XTAL oscillating ready
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR_0        ; Waiting for XTOR(AUXR1.4) true
```

C Code Example:

```
    IFADRL = CKCON2;                  // Index Page-P address to CKCON2
    page_p_sfr_read();                // Read CKCON2 data

    IFD &= ~( XTGS1 | XTGS0 );        // Set XTAL to low gain for 32.768KHz
    IFD |= XTALE;                     // Enable XTAL oscillating
    page_p_sfr_write();               // Write data to CKCON2

    while( AUXR1&XTOR == 0x00 );      // Check XTAL oscillating ready
                                      // Waiting for XTOR(AUXR1.4) true
```

(2) Required function: Enable system timer interrupt with 174.72ms duration (when SYSCLK = IHRCO/2 = 12MHz in default)

Assembly Code Example:

```
    ORG     0003Bh
SystemFlag_ISR:
    ANL     PCON1,#(RTCF)             ; Clear RTC flag (write “1”)
    RETI

main:
    ANL     PCON1,#(RTCF)             ; Clear RTC flag (write “1”)

    MOV     RTCTM,#(RTCCS1 | RTCCS0)  ; Select SYSCLK/2^15 for RTC counter clock source
                                      ; RTCCT[5:0] = 0 for 174.72ms duration

    MOV     RTCCR,#(RTCE)             ; Set RTC reload count, RTCRL[5:0] = 0 for 174.72ms duration
                                      ; Enable RTC counter

    ORL     SFIE,#(RTCFIE)            ; Enable RTC interrupt
    ORL     EIE1,#(ESF)               ; Enable SystemFlag interrupt
    SETB    EA                        ; Enable global interrupt
```

C Code Example:

```
void SystemFlag_ISR (void) interrupt 7
{
    PCON1 &= RTCF;                    // Clear RTC flag (write “1”)
}

viod main (void)
{
    PCON1 &= RTCF;                    // Clear RTC flag (write “1”)

    RTCTM = RTCCS1 | RTCCS0;          // Select SYSCLK/2^15 for RTC counter clock source
                                      // RTCCT[5:0] = 0 for 174.72ms duration
    RTCCR = RTCE;                     // Set RTC reload count, RTCRL[5:0] = 0 for 174.72ms duration
                                      // Enable RTC counter
```

```
    SFIE |= RTCFIE;                         // Enable RTC interrupt
    EIE1 |= ESF;                            // Enable SystemFlag interrupt
    EA = 1;                                 // Enable global interrupt
}
```

(3). Required Function: Enable RTCKO to output SYSCLK/12/2

Assembly Code Example:

```
    ORL     P1M0,#20H                   ; Set RTCKO (P1.5) to push-pull output mode
    MOV     RTCTM,#0BFH                 ; RTC Clock select SYSCLK/12 and set RTCCT[5:0] = 3Fh
    MOV     RTCCR,#03FH                 ; Set RTCRL[5:0] = 3Fh
    ORL     RTCCR,#(RTCE|RTCOE)         ; Enable RTC counter and RTCKO output
```

C Code Example:

```
    P1M0 |= 0x20;                       // Set RTCKO (P1.5) to push-pull output mode
    RTCTM = 0xBF;                       // RTC Clock select SYSCLK/12 and set RTCCT[5:0] = 3Fh
    RTCCR |= 0x3F;                      // Set RTCRL[5:0] = 3Fh
    RTCCR |= (RTCE | RTCOE);            // Enable RTC counter and RTCKO output
```

11. System Reset

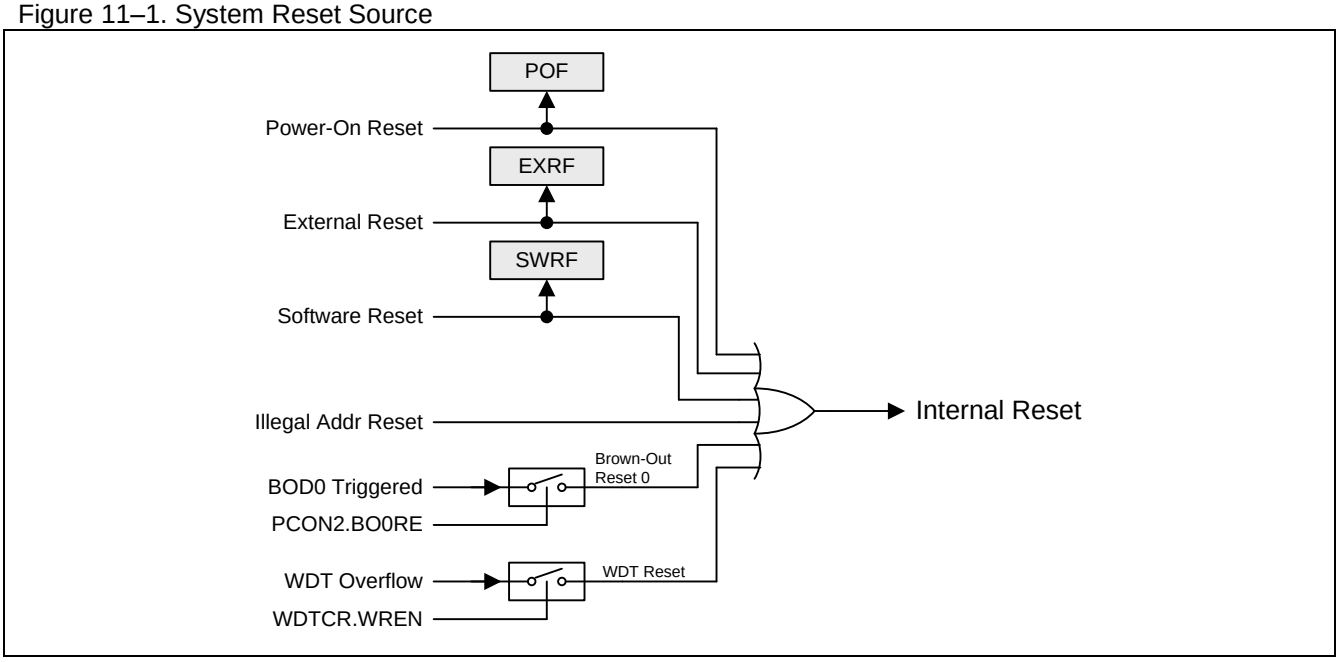
During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector, 0000H, or ISP start address by Hardware Option setting. The **MG86FE/L508** has six sources of reset: power-on reset, external reset, software reset, illegal address reset, WDT reset and brown-out reset. Figure 11–1 shows the system reset source in **MG86FE/L508**.

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

11.1. Reset Source

Figure 11–1 presents the reset systems in the **MG86FE/L508** and all of its reset sources.

Figure 11–1. System Reset Source



11.2. 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

PCON0: Power Control Register 0

SFR Page = Normal & Page P

SFR Address = 0x87 POR = 0001-0000, RESET = 000X-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	GF	**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, Brown-Out 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.

11.3. 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.

PCON1: Power Control Register 1

SFR Page = Normal & Page P
SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	**EXRF**	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 6: EXRF, External Reset Flag.
0: This bit must be cleared by software writing **“1”** on it. Software writing “:0” is no operation.
1: This bit is only set by hardware if an External Reset occurs. Writing **“1”** on this bit will clear EXRF.

11.4. Software Reset

Software can trigger the CPU to restart by software reset, writing “1” on SWRST (ISPCR.5), and set the SWRF flag (PCON1.7). SWBS decides the CPU is boot from ISP or AP region after the reset action

ISPCR: ISP Control Register

SFR Page = Normal
SFR Address = 0xE5 RESET = 0000-xxxx

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

PCON1: Power Control Register 1

SFR Page = Normal & Page P
SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: SWRF, Software Reset Flag.
0: This bit must be cleared by software writing “1” on it. Software writing “0” is no operation.
1: This bit is only set by hardware if a Software Reset occurs. Writing “1” on this bit will clear SWRF.

11.5. Brown-Out Reset

In **MG86FE/L508**, there is a Brown-Out Detectors (BOD0) to monitor VDD power. BOD0 services the fixed detection level at VDD=4.2V for E-series and 2.6V for L-series. If VDD power drops below BOD0 monitor level. Associated flag, BOF0, is set when BOD0 meets the detection level. If BO0RE (PCON2.1) is enabled, the BOD0 will trigger a system reset and a set BOF0 indicates a BOD0 Reset occurred.

PCON1: Power Control Register 1

SFR Page = Normal & Page P

SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	**BOF0**	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 1: BOF0, BOF0 (Reset) Flag.

0: This bit must be cleared by software writing **"1"** on it. Software writing ":0" is no operation.

1: This bit is only set by hardware when VDD meets BOD0 monitored level. Writing **"1"** on this bit will clear BOF0. If BO0RE (PCON2.1) is enabled, BOF0 indicates a BOD0 Reset occurred.

11.6. WDT Reset

When WDT is enabled to start the counter, WDTF will be set by WDT overflow. If WREN (WDTCR.7) is enabled, the WDT overflow will trigger a system reset that causes CPU to restart. Software can read the WDTF to recognize the WDT reset occurred.

PCON1: Power Control Register 1

SFR Page = Normal & Page P

SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	**WDTF**
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 0: WDTF, WDT Overflow/Reset Flag.

0: This bit must be cleared by software writing **"1"** on it. Software writing ":0" is no operation.

1: This bit is only set by hardware when WDT overflows. Writing **"1"** on this bit will clear WDTF. If WREN (WDTCR.7) is set, WDTF indicates a WDT Reset occurred.

11.7. Illegal Address Reset

In **MG86FE/L508**, if software program runs to illegal address such as over program ROM limitation, it triggers a RESET to CPU.

11.8. Reset Sample Code

(1) Required function: Trigger a software reset

Assembly Code Example:

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

C Code Example:

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

(2). Required Function: Enable BOD0 reset

Assembly Code Example:

```
MOV     IFADRL,#PCON2               ; Index Page-P address to PCON2
CALL    _page_p_sfr_read            ; Read PCON2 data

ORL     IFD,#BO0RE                  ; Enable BOD0 reset function
CALL    _page_p_sfr_write           ; Write data to PCON2
```

C Code Example:

```
IFADRL = PCON2;                     // Index Page-P address to PCON2
page_p_sfr_read();                  // Read PCON2 data

IFD |= BO0RE;                       // Enable BOD0 reset function
page_p_sfr_write();                 // Write data to PCON2
```

12. Power Management

The **MG86FE/L508** supports one power monitor module, Brown-Out Detector (BOD0), and **7** power-reducing modes: Idle mode, Power-down mode, Slow mode, Sub-Clock mode, RTC mode, Watch mode and Monitor mode.

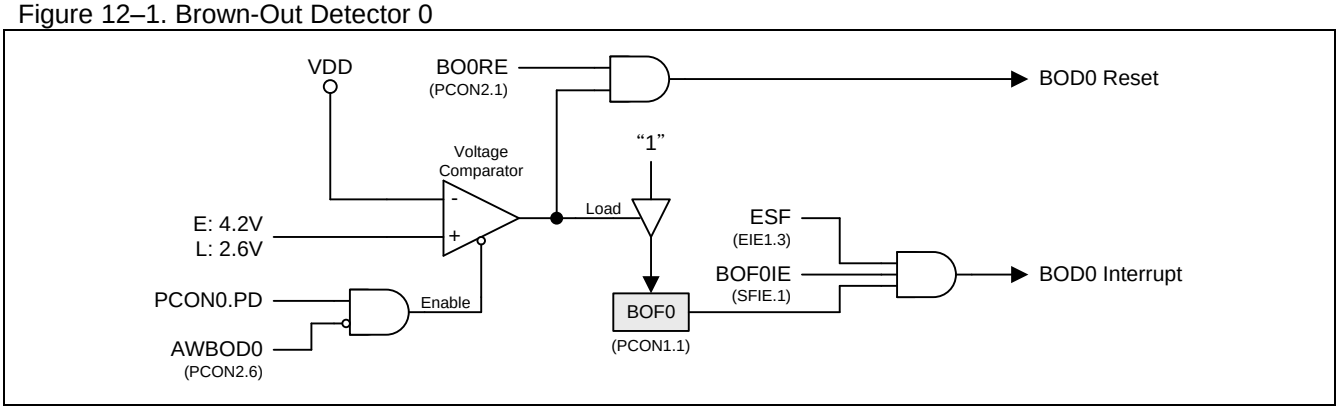
BOD0 reports the chip power status on the flag, BOF0, which provides the capability to interrupt CPU or to reset CPU by software configured. The seven power-reducing modes provide the different power-saving scheme for chip application. These modes are accessed through the CKCON0, CKCON2, RTCTR, PCON0, PCON1, PCON2 and WDTCR register.

12.1. Brown-Out Detector

In **MG86FE/L508**, there is a Brown-Out Detectors (BOD0) to monitor VDD power. Figure 12–1 shows the functional diagram of BOD0. BOD0 services the fixed detection level at VDD=**4.2V** for **5V** device and **2.6V** on **3.3V** device. Associated flag, BOF0 (PCON1.1), is set when BOD0 meets the detection level. If both of ESF (EIE1.3) and BOF0IE (SFIE.1) are enabled, a set BOF0 will generate a system flag interrupt. It can interrupt CPU either CPU in normal mode or idle mode. The interrupt also wakes up CPU in power down mode if AWBOD0 (PCON2.6) is enabled.

If BO0RE (PCON2.1) is enabled, the BOD0 event will trigger a system reset and a set BOF0 indicates a BOD0 Reset occurred. The BOD0 reset restart the CPU either CPU in normal mode or idle mode. The reset also restart CPU in power down mode if AWBOD0 (PCON2.6) is enabled in BOD0 reset operation.

Figure 12–1. Brown-Out Detector 0



12.2. Power Saving Mode

12.2.1. Slow Mode

The alternative to save the operating power is to slow the MCU's operating speed by programming SCKS2~SCKS0 bits (in CKCON0 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 in the other program segments.

12.2.2. Sub-Clock Mode

The alternative to slow down the MCU's operating speed by programming OSCS1~0 can select the ILRCO for system clock. The 64KHz ILRCO provides the MCU to operates in an ultra low speed and low power operation. Additional programming SCKS2~SCKS0 bits (in CKCON0 register, see Section "8 System Clock"), the user could put the MCU speed down to 500Hz slowest.

12.2.3. RTC Mode

The **MG86FE/L508** has a simple RTC module that allows a user to continue running an accurate timer while the rest of the device is powered-down. In RTC mode, the RTC module behaves a "Clock" function and can be a wake-up source from chip power down by RTC overflow rate. Please refer Section "10 Real-Time-Clock(RTC)/System-Timer" for more detail information.

12.2.4. Watch Mode

If Watch-Dog-Timer is enabled and NSW is set, Watch-Dog-Timer will keep running in power down mode, which named Watch Mode in **MG86FE/L508**. When WDT overflows, set WDTF and wakeup CPU from interrupt or system reset by software configured. The maximum wakeup period is about 2 seconds that is defined by WDT pre-scaler. Please refer Section "9 Watch Dog Timer (WDT)" and Section "14 Interrupt" for more detail information.

12.2.5. Monitor Mode (L-Series Only)

If AWBOD1 (PCON3.3) is set, BOD1 will keep VDD monitor in power down mode. It is the Monitor Mode in **MG86FE/L508**. When BOD1 meets the detection level, set BOF1 and wakeup CPU from interrupt or system reset by software configured. Please refer Section "12.1 Brown-Out Detector" and Section "14 Interrupt" for more detail information. ***This function is only valid in L-series device.***

12.2.6. 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, RTC, KBI and the BOD0 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.

The ADC input channels must be set to "***Analog Input Only***" in **P1AIO** SFR when MCU is in idle mode or power-down mode.

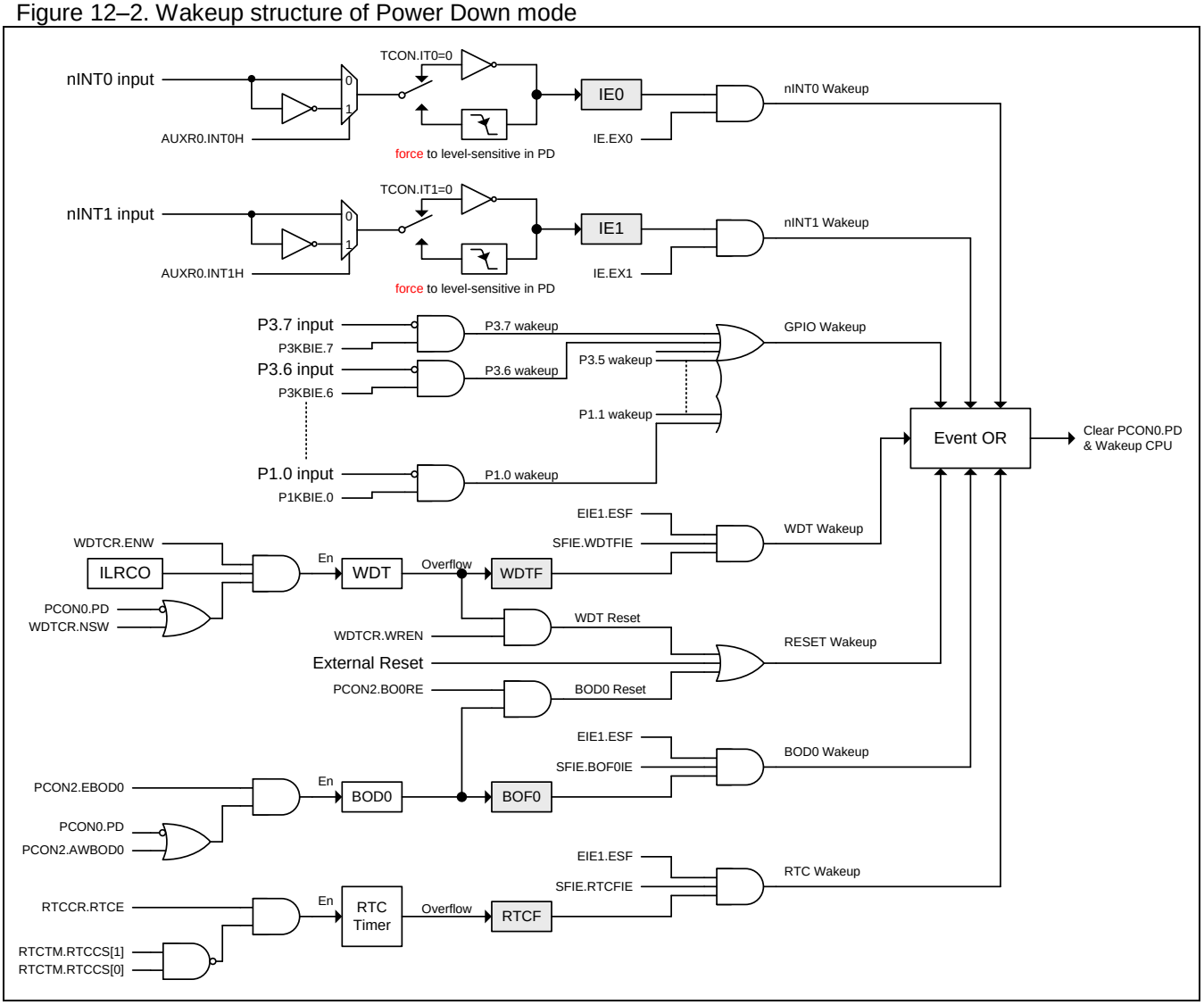
12.2.7. Power-Down Mode

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, enabled external interrupts, enabled KBI GPIOs, enabled BOD0 or enabled Non-Stop WDT.

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. To ensure minimum power consumption in power down mode, software must confirm all I/O not in floating state, including the port I/Os un-appearance on package pins. **For example, P2.7~P2.0 are un-appearance in MG86FE504AE20 (SOP20) package pins. Software may configure P2(A0H) corresponding bit SFR to "0" (output low) to avoid pin floating in power-down mode.**

Figure 12–2 shows the wakeup mechanism of power-down mode in **MG86FE/L508**.

Figure 12–2. Wakeup structure of Power Down mode



12.2.8. Interrupt Recovery from Power-down

Two external interrupts may be configured to terminate Power-down mode. External interrupts nINT0 (P3.2), nINT1 (P3.3) may be used to exit Power-down. To wake up by external interrupt nINT0, nINT1, the interrupt must be enabled and configured for level-sensitive operation. If the enabled external interrupts are configured to edge-sensitive operation (Falling or Rising), they will be forced to level-sensitive operation (Low level or High 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.

12.2.9. Reset Recovery from Power-down

If P3.6 is configured 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.

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.

12.2.10. KBI Recovery from Power-down

The GPIOs of **MG86FE/L508**, P1.7 ~ P1.0, P3.7~P3.4, P4.1, P4.0, P3.1 and P3.0 have wakeup CPU capability that are enabled by individual control bit in P1KBIE and P3KBIE. P3.2/nINT0 and P3.3/nINT1 have the wakeup capability from the interrupt function.

Wakeup from Power-down through an enabled KBI GPIO is similar to the interrupt. At the low-level of enabled KBI GPIO, 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. After the timeout period, CPU will meet a KBI interrupt and execute the interrupt service routine. Please refer Section “18 Keypad Interrupt (KBI)” for more detail information.

12.2.11. Safety and Fast wake-up for XTAL mode

If MCU is running crystal mode and needs power-down and wake-up procedure, **MG86FE/L508** provide a flow to get the system “Safety and Fast wake-up”. When MCU get the power-down stage in crystal mode, software can enable IHRCO and switch SYSCLK to IHRCO. Then MCU go to power-down mode and wait for wake-up event. After wake-up trigger, the MCU will wake up from IHRCO to handle the system immediately. (The general wake-up time is about **1us**, please refer Section “27.4 IHRCO Characteristics” for IHRCO wake-up information) And software polls the XTOR for crystal oscillating stable. If XTOR is set, software switches SYSCLK from IHRCO to XTAL mode for accuracy time base operation.

12.3. Power Control Register

PCON0: Power Control Register 0

SFR Page = Normal & Page P

SFR Address = 0x87 POR = 0001-0000, RESET = 000x-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	--	POF	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: POF0, Power-On Flag 0.
0: This bit must be cleared by software writing one to it.
1: This bit is set by hardware if a Power-On Reset occurs.

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.

PCON1: Power Control Register 1

SFR Page = Normal & Page P

SFR Address = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: SWRF, Software Reset Flag.
0: This bit must be cleared by software writing **"1"** to it.
1: This bit is set by hardware if a Software Reset occurs.

Bit 6: EXRF, External Reset Flag.
0: This bit must be cleared by software writing **"1"** to it.
1: This bit is set by hardware if an External Reset occurs.

Bit 5: Reserved. Software must write "0" on this bit when PCON1 is written.

Bit 4: RTCF, RTC overflow flag.
0: This bit must be cleared by software writing **"1"** on it. Software writing ":0" is no operation.
1: This bit is only set by hardware when RTCCT overflows. Writing **"1"** on this bit will clear RTCF.

Bit 3: KBIF, KBI Flag.
0: This bit must be cleared by software writing **"1"** to it.
1: This bit is set by hardware if a KBI input event occurs.

Bit 2: Reserved. Software must write "0" on this bit when PCON1 is written.

Bit 1: BOF0, Brown-Out Detection flag 0.
0: This bit must be cleared by software writing **"1"** to it.
1: This bit is set by hardware if the operating voltage matches the detection level of Brown-Out Detector 0 (E: 4.2V, L: 2.4V).

Bit 0: WDTF, WDT overflow flag.
0: This bit must be cleared by software writing **"1"** to it.
1: This bit is set by hardware if a WDT overflow occurs.

PCON2: Power Control Register 2

SFR Page = **Page P Only**

SFR Address = 0x44 POR = x0xx-xx01

7	6	5	4	3	2	1	0
--	**AWBOD0**	--	--	--	--	BO0RE	**1**
W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7: Reserved. Software must write “0” on this bit when PCON2 is written.

Bit 2: AWBOD0, Awaked BOD0 in PD mode.
0: BOD0 is disabled in power-down mode.
1: BOD0 keeps operation in power-down mode.

Bit 5~2: Reserved. Software must write “0” on these bits when PCON2 is written.

Bit 1: BO0RE, BOD0 Reset Enabled. Its initial is loaded from OR1.BO0REO inverted.
0: Disable BOD0 to trigger a system reset when BOF0 is set.
1: Enable BOD0 to trigger a system reset when BOF0 is set by VDD meets 4.2V(E) or 2.4V(L).

Bit 0: Reserved for test. Software must write “**1**” on this bit when PCON2 is written.

P1KBIE: Port 1 KBI Enable Control Register

SFR Page = Normal

SFR Address = 0xD7 RESET = 0000-0000

7	6	5	4	3	2	1	0
P17KBIE	P16KBIE	P15KBIE	P14KBIE	P13KBIE	P12KBIE	P11KBIE	P10KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: Keypad Input function enable bit for each Port 1 pins.
0: Disable associated port pin for keypad input function.
1: Enable associated port pin for keypad input function on low level.

P3KBIE: Port 3 KBI Enable Control Register

SFR Page = Normal

SFR Address = 0xD6 RESET = 0000-0000

7	6	5	4	3	2	1	0
P37KBIE	P36KBIE	P35KBIE	P34KBIE	**P41KBIE**	**P40KBIE**	P31KBIE	P30KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: Keypad Input function enable bit for P3.7 ~ P3.4, P4.1, P4.0, P3.1 and P3.0 pins.
0: Disable associated port pin for keypad input function.
1: Enable associated port pin for keypad input function on low level.

12.4. Power Control Sample Code

(1) Required function: Select Slow mode with OSCin/128 (default is OSCin/2)

Assembly Code Example:

```
ORL     CKCON0,#(SCKS0 | SCKS1 | SCKS2)     ; OSCin/128

MOV     IFADRL,#DCON0                       ; Index Page-P address to DCON0
CALL    _page_p_sfr_read                    ; Read DCON0 data

ANL     IFD,#~(HSE)                         ; Disable HSE when SYSCLK ≤ 6MHz for power saving
CALL    _page_p_sfr_write                   ; Write data to DCON0
```

C Code Example:

```
CKCON0 |= (SCKS2 | SCKS1 | SCKS0);       // Select system clock divider to OSCin/128.

IFADRL = DCON0;                          // Index Page-P address to DCON0
page_p_sfr_read();                       // Read DCON0 data.

IFD &= ~HSE;                             // Disable HSE when SYSCLK ≤ 6MHz for power saving
page_p_sfr_write();                      // Write data to DCON0
```

(2) Required function: Select Sub-Clock mode with OSCin/2 (default is OSCin/2=64KHz/2=32KHz)

Assembly Code Example:

```
MOV     IFADRL,#CKCON2                   ; Index Page-P address to CKCON2
CALL    _page_p_sfr_read                 ; Read CKCON2 data

ANL     IFD,#~(OSCS1|OSCS0)              ; Switch OSCin source to ILRCO
ORL     IFD,#OSCS1
CALL    _page_p_sfr_write                ; Write data to CKCON2

ANL     IFD,#~(IHRCOE|XTALE)             ; Disable IHRCO & XTAL
CALL    _page_p_sfr_write                ; Write data to CKCON2

MOV     IFADRL,#DCON0)                   ; Index Page-P address to DCON0
CALL    _page_p_sfr_read                 ; Read DCON0 data

ANL     IFD,#~(HSE)                      ; Disable HSE when SYSCLK ≤ 6MHz for power saving
CALL    _page_p_sfr_write                ; Write data to DCON0

MOV     A,CKCON0                         ; Select system clock = OSCin/2
ANL     A,#~(SCKS2|SCKS1|SCKS0)
ORL     A,#SCKS0
MOV     CKCON0,A
```

C Code Example:

```
IFADRL = CKCON2;                         // Index Page-P address to CKCON2
page_p_sfr_read();                       // Read CKCON2 data

IFD &= ~(OSCS1 | OSCS0);                 // Switch OSCin source to ILRCO
IFD |= OSCS1;
page_p_sfr_write();                      // Write data to CKCON2

IFD = IFD & ~(IHRCOE|XTALE);             // Disable IHRCO & XTAL
page_p_sfr_write();                      // Write data to CKCON2

IFADRL = DCON0;                          // Index Page-P address to DCON0
page_p_sfr_read();                       // Read DCON0 data

IFD = IFD & ~(HSE);                      // Disable HSE when SYSCLK ≤ 6MHz for power saving
page_p_sfr_write();                      // Write data to DCON0
```

```
    ACC = CKCON0;                          // Select system clock = OSCin/2
    ACC &= ~(SCKS2 | SCKS1 | SCKS0);
    ACC |= SCKS0;
    CKCON0 = ACC;
```

(3). Required Function: Switch MCU running with 32.768KHz XTAL mode

Assembly Code Example:

```
    MOV     IFADRL,#CKCON2              ; Index Page-P address to CKCON2
    CALL    _page_p_sfr_read            ; Read CKCON2 data

    ANL     IFD,#~(XTGS1|XTGS0)         ; Set XTAL to low gain for 32.768KHz
    ORL     IFD,#(XTALE)                ; Enable XTAL oscillating
    CALL    _page_p_sfr_write           ; Write data to CKCON2

check_XTOR_0:                           ; Check XTAL oscillating ready
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR_0          ; Waiting for XTOR(AUXR1.4) true

    ANL     IFD,#~(OSCS1|OSCS0)         ; Switch OSCin source to XTAL 32.768KHz
    ORL     IFD,#OSCS0
    CALL    _page_p_sfr_write           ; Write data to CKCON2

    ANL     IFD,#~(IHRCOE)              ; Disable IHRCO
    CALL    _page_p_sfr_write           ; Write data to CKCON2

    MOV     IFADRL,#DCON0               ; Index Page-P address to DCON0
    CALL    _page_p_sfr_read            ; Read DCON0 data

    ANL     IFD,#~(HSE)                 ; Disable HSE when SYSCLK ≤ 6MHz for power saving
    CALL    _page_p_sfr_write           ; Write data to DCON0

    ANL     CKCON0,#~(SCKS2|SCKS1|SCKS0) ; SYSCLK = OSCin/1 = 32.768KHz
```

C Code Example:

```
    IFADRL = CKCON2;                    // Index Page-P address to CKCON2
    page_p_sfr_read();                  // Read CKCON2 data

    IFD &= ~( XTGS1 | XTGS0 );          // Set XTAL to low gain for 32.768KHz
    IFD |= XTALE;                       // Enable XTAL oscillating
    page_p_sfr_write();                 // Write data to CKCON2

    while( AUXR1&XTOR == 0x00 );        // Check XTAL oscillating ready
                                        // Waiting for XTOR(AUXR1.4) true

    IFD &= ~(OSCS1 | OSCS0);            // Switch OSCin source to XTAL.
    IFD |= OSCS0;
    page_p_sfr_write ();                // Write data to CKCON2

    IFD &= ~IHRCOE;                     // Disable IHRCO if MCU is switched from IHRCO
    page_p_sfr_write();                 // Write data to CKCON2.

    IFADRL = DCON0;                     // Index Page-P address to DCON0
    page_p_sfr_read();                  // Read DCON0 data.

    IFD &= ~HSE;                        // Disable HSE when SYSCLK ≤ 6MHz for power saving
    page_p_sfr_write();                 // Write data to DCON0

    CKCON0 &= ~(SCKS2 | SCKS1 | SCKS0);  // SYSCLK = OSCin/1 = 32.768KHz
```

(4). Required Function: Enter Watch mode with 2S wake-up duration

Assembly Code Example:

```
    ORG     0003Bh
SystemFlag_ISR:
```

```
    ANL     PCON1,#(WDTF)                   ; Clear WDT flag (write “1”)
    RETI

main:
    ANL     PCON1,#WDTF                     ; Clear WDTF flag (write “1”)
    ORL     WDTCR,#(NSW|ENW|PS2|PS1|PS0)
                                            ;Enable WDT and NSW (for watch mode)
                                            ;Set PS[2:0] = 7 to select WDT period for 1.984s

    ORL     SFIE,#WDTFIE                    ; Enable WDT interrupt
    ORL     EIE1,#ESF                       ; Enable SystemFlag interrupt
    SETB    EA                              ; Enable Global interrupt

    ORL     PCON0,#PD                       ; Set MCU to power down

;   MCU wait for wake-up
```

C Code Example:

```
void SystemFlag_ISR (void) interrupt 7
{
    PCON1 &= WDTF;                          // Clear WDT flag (write “1”)
}

viod main (void)
{
    PCON1 &= WDTF;                          // Clear WDT flag (write “1”)
    WDTCR |= (NSW | ENW | PS2 | PS1 | PS0);  // Enable WDT and NSW (for watch mode)
                                            // Set PS[2:0] = 7 to select WDT period for 1.984s

    SFIE |= WDTFIE;                         // Enable WDT interrupt
    EIE1 |= ESF;                            // Enable SystemFlag interrupt
    EA = 1;                                 // Enable global interrupt

    PCON0 |= PD;                            // Set MCU to power down

//  MCU wait for wake-up
}
```

(5). Required Function: Monitor Mode (L-series only)

Assembly Code Example:

```
    ORG     0003Bh
SystemFlag_ISR:
    ANL     PCON1,#(BOF0)                   ; Clear BOD0 flag (write “1”)
    RETI

main:
    MOV     IFADRL,#PCON2                   ; Index Page-P address to PCON2
    CALL    _page_p_sfr_read                ; Read PCON2 data

    ORL     IFD,#AWBOD0                     ; Enable BOD0 operating in power-down mode
    CALL    _page_p_sfr_write               ; Write data to PCON2

    ORL     SFIE,#BOF0IE                    ; Enable BOF0 interrupt
    ORL     EIE1,#ESF                       ; Enable SystemFlag interrupt
    SETB    EA                              ; Enable global interrupt

    ORL     PCON0,#PD                       ; Set MCU to power down

;   MCU wait for wake-up
```

C Code Example:

```
void SystemFlag_ISR() interrupt 7
{
```

```
    PCON1 &= BOF0;                          // Clear BOD0 flag (write “1”)
}

void main()
{
    IFADRL = PCON2;                         // Index Page-P address to PCON2
    page_p_sfr_read();                      // Read PCON2 data

    IFD |= AWBOD0;                          // Enable BOD0 operating in power-down mode
    page_p_sfr_write();                     // Write data to PCON2


    SFIE |= BOF0IE;                         // Enable BOD0 interrupt
    EIE1 |= ESF;                            // Enable SystemFlag interrupt
    EA = 1;                                 // Enable global interrupt

    PCON0 |= PD;                            // Set MCU to power down

//  MCU wait for wake-up
}
```

(6). Required Function: Safety and Fast wakeup for XTAL mode in power-down

Assembly Code Example:

```
    MOV     IFADRL,#CKCON2              ; Index Page-P address to CKCON2
    CALL    _page_p_sfr_read            ; Read CKCON2 data

    ORL     IFD,#IHRCOE                 ; Enable IHRCO
    CALL    _page_p_sfr_write           ; Write data to CKCON2

    Delay_32us

    ANL     IFD,#~(OSCS1|OSCS0)         ; Switch OSCin source to IHRCO
    CALL    _page_p_sfr_write           ; Write data to CKCON2

    ORL     PCON0,#PD                   ; Set MCU to power down

;   MCU wait for wake-up

check_XTOR:                             ; Check XTAL oscillating ready
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR            ; Waiting for XTOR(AUXR1.4) true

    ANL     IFD,#~(OSCS1 | OSCS0)       ; Switch OSCin source to XTAL.
    ORL     IFD,#(OSCS0)
    CALL    _page_p_sfr_write           ; Write data to CKCON2

    ANL     IFD,#~(IHRCOE)              ; Disable IHRCO if MCU is switched from IHRCO
    CALL    _page_p_sfr_write           ; Write data to CKCON2
```

C Code Example:

```
    IFADRL = CKCON2;                    // Index Page-P address to CKCON2
    page_p_sfr_read();                  // Read CKCON2 data

    IFD |= IHRCOE;                      // Enable IHRCO
    page_p_sfr_write();                 // Write data to CKCON2

    Delay_32us();

    IFD &= ~(OSCS1 | OSCS0);            // Switch OSCin source to IHRCO
    page_p_sfr_write();                 // Write data to CKCON2

    PCON0 |= PD;                        // Set MCU to power down

//  MCU wait for wake-up
```

```
while(AUXR1 & XTOR == 0x00);          // Check XTAL oscillating ready
                                      // Waiting for XTOR(AUXR1.4) true

IFD &= ~(OSCS1 | OSCS0);              // Switch OSCin source to XTAL.
IFD |= OSCS0;
page_p_sfr_write ();                  // Write data to CKCON2

IFD &= ~IHRCOE;                       // Disable IHRCO if MCU is switched from IHRCO
page_p_sfr_write();                   // Write data to CKCON2.
```

13. Configurable I/O Ports

The **MG86FE/L508** has following I/O ports: P1.0~P1.7, P2.0~P2.7, P3.0~P3.7 and P4.0~P4.1. **RST** pin has a swapped function on **P3.6**. If select external crystal oscillator as system clock input, Port 4.0 and Port 4.1 are configured to XTAL2 and XTAL1. The exact number of I/O pins available depends upon the package types. See Table 13–1.

Table 13–1. Number of I/O Pins Available

Package Type	I/O Pins	Number of I/O ports
28-pin SOP	P1.0~P1.7, P2.0~P2.7, P3.0~P3.5, P3.7, P3.6(RST), P4.0 (ECKI/XTAL2), P4.1 (XTAL1)	26 or 25 (RST selected) or 24 (RST & ECKI selected) or 23 (RST & XTAL selected)
20-pin SOP	P1.0~P1.7, P3.0~P3.5, P3.7, P3.6(RST), P4.0 (ECKI/XTAL2), P4.1 (XTAL1)	18 or 17 (RST selected) or 16 (RST & ECKI selected) or 15 (RST & XTAL selected)

13.1. IO Structure

The I/O operating modes are distinguished two groups in **MG86FE/L508**. The first group is only for Port 3 to 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.

All other general port pins belong to the second group. They can be programmed to two output modes, push-pull output and open-drain output with pull-up resistor control. The default setting of this group I/O is open-drain mode with output high, which means input mode with high impedance state.

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

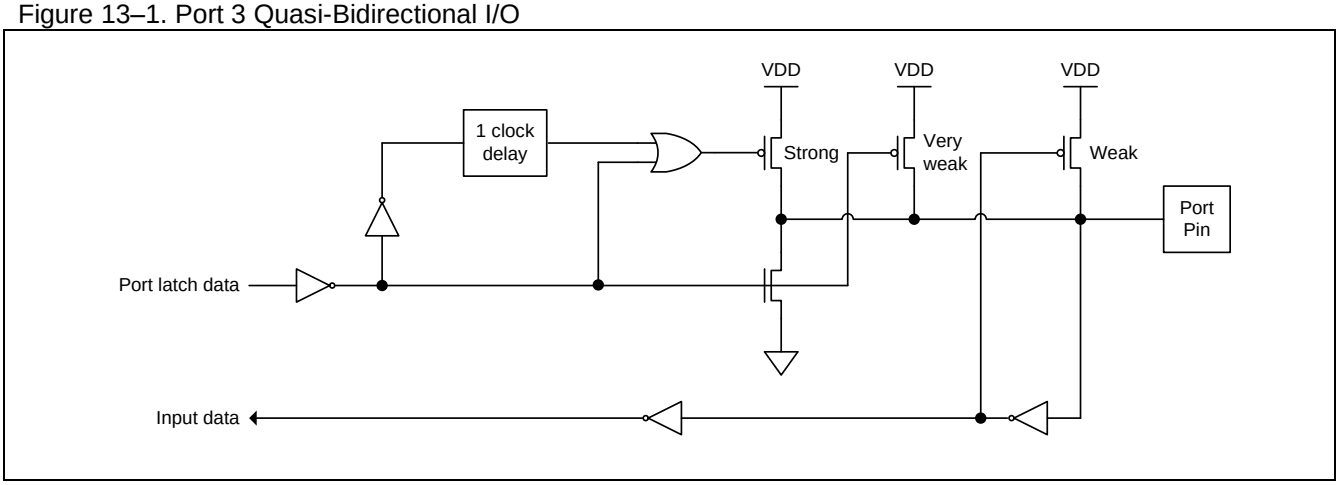
13.1.1. Port 3 Quasi-Bidirectional IO Structure

Port 3 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 **one** CPU clocks, quickly pulling the port pin high.

The quasi-bidirectional port configuration on Port 3 is shown in Figure 13–1.

Figure 13–1. Port 3 Quasi-Bidirectional I/O

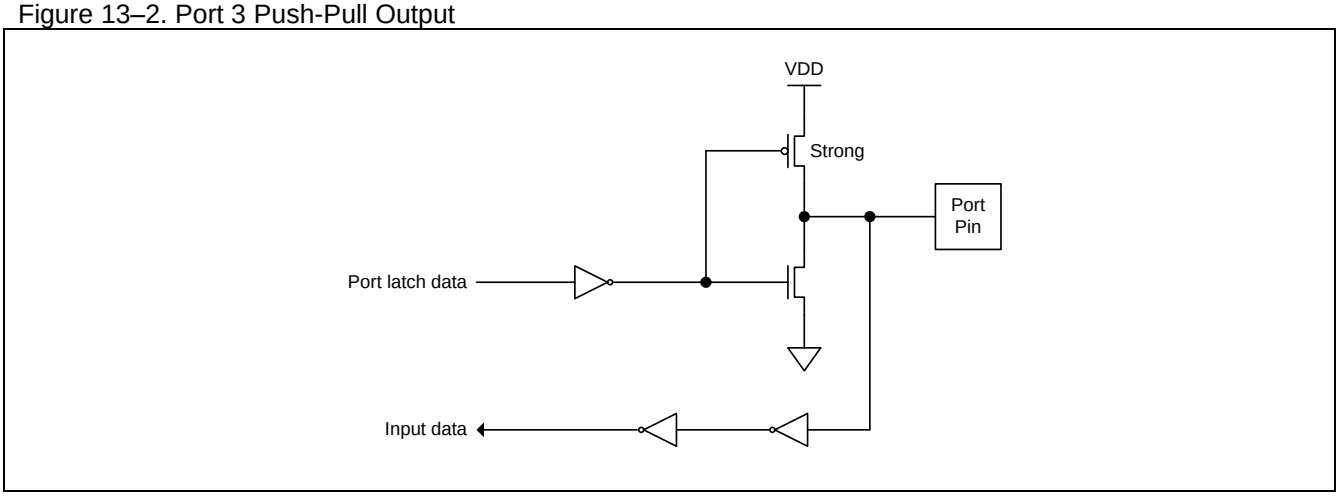


13.1.2. Port 3 Push-Pull Output Structure

The push-pull output configuration on Port 3 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 on Port 3 is shown in Figure 13–2.

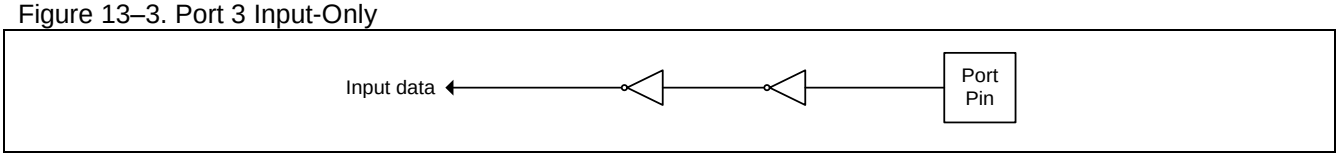
Figure 13–2. Port 3 Push-Pull Output



13.1.3. Port 3 Input-Only (High Impedance Input) Structure

The input-only configuration on Port 3 is an input without any pull-up resistors on the pin, as shown in Figure 13–3.

Figure 13–3. Port 3 Input-Only

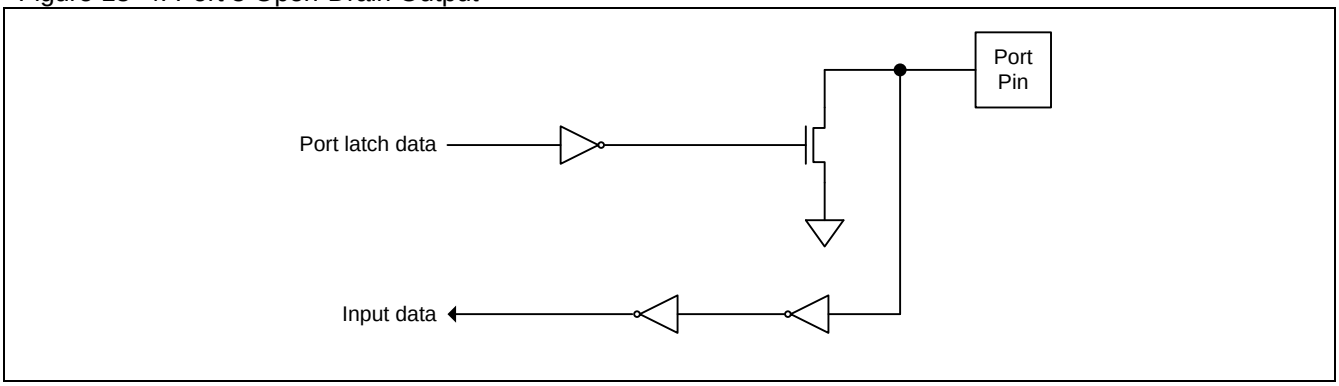


13.1.4. Port 3 Open-Drain Output Structure

The open-drain output configuration on Port 3 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 Port 3 open-drain port configuration is shown in Figure 13–4.

Figure 13–4. Port 3 Open-Drain Output

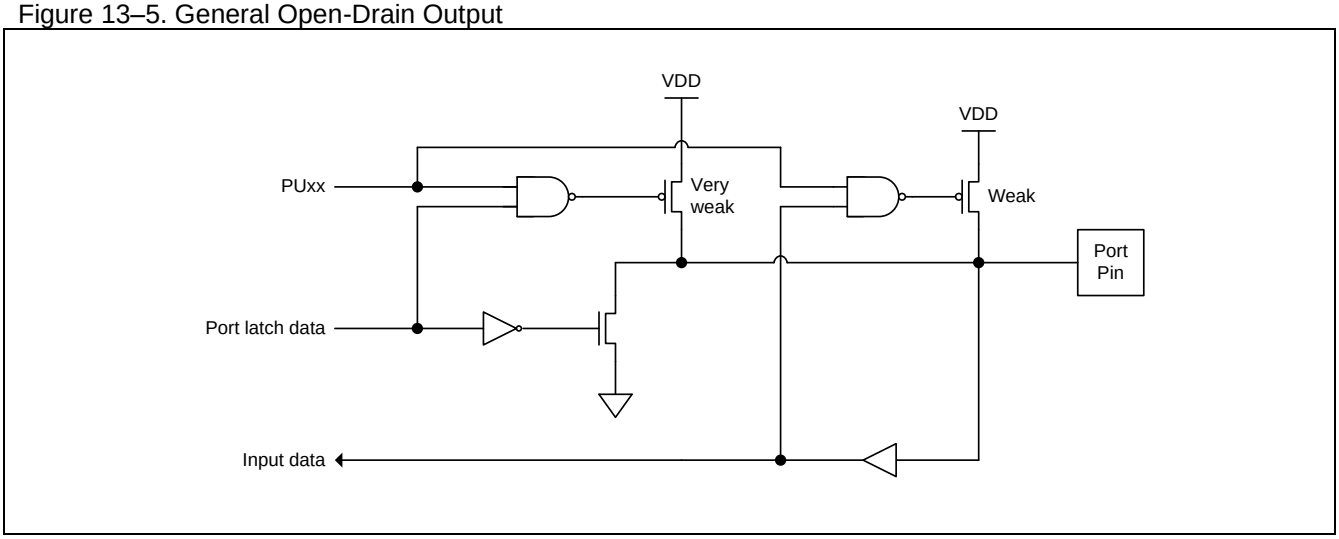


13.1.5. General Open-Drain Output Structure

The open-drain output configuration on general port pins only drives the pull-down transistor of the port pin when the Port Data register contains a logic “0”. To use this configuration in application, a port pin can select an external pull-up, or an on-chip pull-up by software enabled in PUCON0.

The general open-drain port configuration is shown in Figure 13–5.

Figure 13–5. General Open-Drain Output

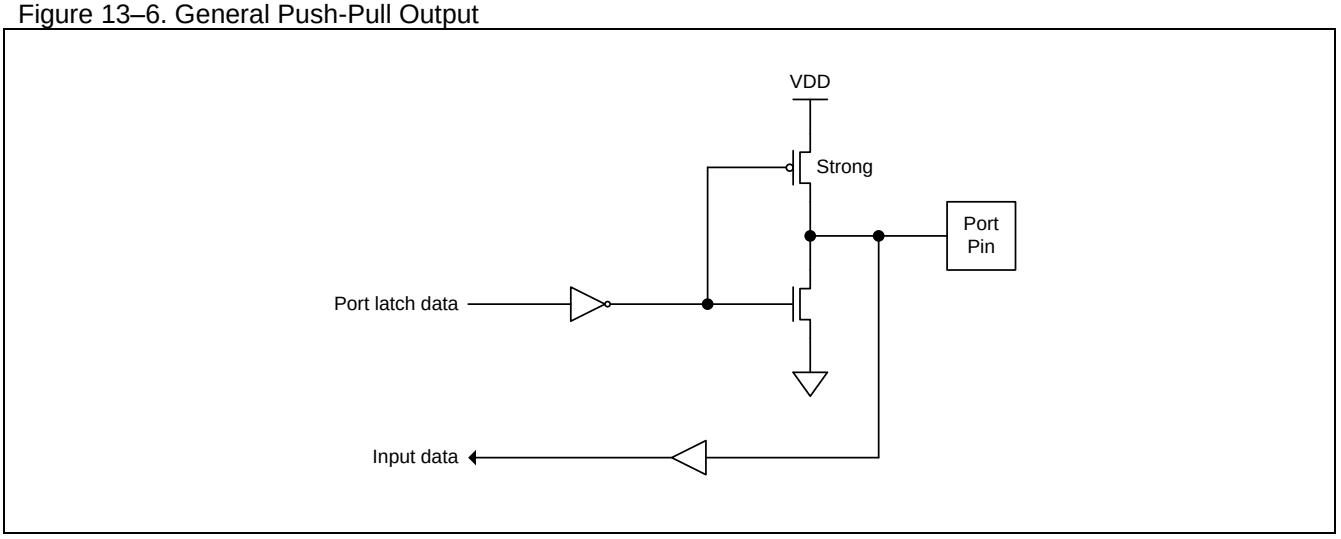


13.1.6. General Push-Pull Output Structure

The push-pull output configuration on general port pins has the same pull-down structure as the open-drain 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 open-drain mode.

The push-pull port configuration is shown in Figure 13–6.

Figure 13–6. General Push-Pull Output



13.1.7. General Port Input Configured

A Port pin is configured as a digital input by setting its output mode to “Open-Drain” and writing a logic “1” to the associated bit in the Port Data register. For example, P1.7 is configured as a digital input by setting P1M0.7 to a logic 0 and P1.7 to a logic 1.

13.2. I/O Port Register

All I/O port pins on the **MG86FE/L508** may be individually and independently configured by software to select its operating mode. Only Port 3 has four operating modes, as shown in Table 13–2. Two mode registers select the output type for each port 3 pin.

Table 13–2. Port 3 Configuration Settings

P3M0.y	P3M1.y	Port Mode
0	0	Quasi-Bidirectional
0	1	Push-Pull Output
1	0	Input Only (High Impedance Input)
1	1	Open-Drain Output

Where y=0~7 (port pin). The registers P3M0 and P3M1 are listed in each port description.

Other general port pins support two operating modes, as shown in Table 13–3. One mode register selects the output type for each port pin.

Table 13–3. Port Configuration Settings

PxM0.y	Port Mode
0	Open-Drain Output
1	Push-Pull Output

Where x=1, 2, 4 (port number), and y=0~7 (port pin). The registers PxM0 and PxM1 are listed in each port description.

13.2.1. Port 1 Register

P1: Port 1 Register

SFR Page = Normal
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 Page = Normal
SFR Address = 0x91 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**

0: Port pin output mode is configured to open-drain.
1: Port pin output mode is configured to push-pull.

P1AIO: Port 1 Analog Input Only

SFR Page = Normal
SFR Address = 0x92 RESET = 0000-0000

7	6	5	4	3	2	1	0
P17AIO	**P16AIO**	**P15AIO**	**P14AIO**	**P13AIO**	**P12AIO**	**P11AIO**	**P10AIO**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: Port pin has digital and analog input capability.
1: Port pin only has analog input only for ADC input application. The corresponding Port PIN Register bit will always read as “0” when this bit is set.

13.2.2. Port 2 Register

P2: Port 2 Register

SFR Page = Normal
SFR Address = 0xA0 RESET = 1111-1111

7	6	5	4	3	2	1	0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: P2.7~P2.0 could be only set/cleared by CPU.

P2M0: Port 2 Mode Register 0

SFR Page = Normal
SFR Address = 0x95 RESET = 0000-0000

7	6	5	4	3	2	1	0
P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: Port pin output mode is configured to open-drain.
1: Port pin output mode is configured to push-pull.

13.2.3. Port 3 Register

P3: Port 3 Register

SFR Page = Normal
SFR Address = 0xB0 RESET = 1111-1111

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

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

P3M0: Port 3 Mode Register 0

SFR Page = Normal
SFR Address = 0xB1 RESET = 0000-0000

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

P3M1: Port 3 Mode Register 1

SFR Page = Normal
SFR Address = 0xB2 RESET = 0000-0000

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

13.2.4. Port 4 Register

P4: Port 4 Register

SFR Page = Normal
SFR Address = 0xE8 RESET = xxxx-xx11

7	6	5	4	3	2	1	0
--	--	--	--	--	--	P4.1	P4.0
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7~2: Reserved.

Bit 1~0: P4.1~P4.0 could be only set/cleared by CPU. P4.1 and P4.0 have the alternated function for crystal oscillating circuit, XTAL1 and XTAL2.

P4M0: Port 4 Mode Register 0

SFR Page = Normal

SFR Address = 0xB3 RESET = xxxx-xx00

7	6	5	4	3	2	1	0
--	--	--	--	--	--	P4M0.1	P4M0.0
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

0: Port pin output mode is configured to open-drain.
1: Port pin output mode is configured to push-pull.

13.2.5. Pull-Up Control Register

PUCON0: Port Pull-up Control Register 0

SFR Page = Normal

SFR Address = 0xB4 RESET = x000-00xx

7	6	5	4	3	2	1	0
--	PU40	P2PU1	P2PU0	PU11	PU10	--	--
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 7: Reserved. Software must write “0” on this bit when PUCON0 is written.

Bit 6: Port 4 pull-up enable control on low nibble.
0: Disable the P4.0 & P4.1 pull-up resistor in open-drain output mode.
1: Enable the P4.0 & P4.1 pull-up resistor in open-drain output mode.
If set this bit in P4 push-pull mode, the bit control capability will be inhibited its function on associated push-pull enabled pin. Because it is considered in GPIO design, there is no concern in control logic.

Bit 5: Port 2 pull-up enable control on high nibble.
0: Disable the P2.7 ~ P2.4 pull-up resistor in open-drain output mode.
1: Enable the P2.7 ~ P2.4 pull-up resistor in open-drain output mode.

Bit 4: Port 2 pull-up enable control on low nibble.
0: Disable the P2.3 ~ P2.0 pull-up resistor in open-drain output mode.
1: Enable the P2.3 ~ P2.0 pull-up resistor in open-drain output mode.

Bit 3: Port 1 pull-up enable control on high nibble.
0: Disable the P1.7 ~ P1.4 pull-up resistor in open-drain output mode.
1: Enable the P1.7 ~ P1.4 pull-up resistor in open-drain output mode.
If set this bit in P1 push-pull mode, the bit control capability will be inhibited its function on associated push-pull enabled pin. Because it is considered in GPIO design, there is no concern in control logic.

Bit 2: Port 1 pull-up enable control on low nibble.
0: Disable the P1.3 ~ P1.0 pull-up resistor in open-drain output mode.
1: Enable the P1.3 ~ P1.0 pull-up resistor in open-drain output mode.
If set this bit in P1 push-pull mode, the bit control capability will be inhibited its function on associated push-pull enabled pin. Because it is considered in GPIO design, there is no concern in control logic.

Bit 1~0: Reserved. Software must write “0” on these bits when PUCON0 is written.

13.3. GPIO Sample Code

(1). Required Function: Set P1.0 to input mode with on-chip pull-up resistor enabled

Assembly Code Example:

```
ANL     P1M0,#~P1M00            ; Configure P1.0 to open drain mode
SETB    P10                     ; Set P1.0 data latch to “1” to enable input mode
ORL     PUCON0,#PU10            ; Enable the P1.3~P1.0 on-chip pull-up resistor
```

C Code Example:

```
P1M0 &= P1M00;                  // Configure P1.0 to open drain mode
P10 = 1;                        // Set P1.0 data latch to “1” to enable input mode
PUCON0 |= PU10;                 // Enable the P1.3~P1.0 on-chip pull-up resistor
```

(2). Required Function: Switch RST pint to P3.6

Assembly Code Example:

```
MOV     IFADRL,#DCON0           ; Index Page-P address to DCON0
CALL    _page_p_sfr_read        ; Read DCON0 data

ANL     IFD,#~(RSTIO)           ; Select I/O pad function for P36
CALL    _page_p_sfr_write       ; Write data to DCON0
```

C Code Example:

```
IFADRL = DCON0;                 // Index Page-P address to DCON0
page_p_sfr_read();              // Read DCON0 data.

IFD &= ~RSTIO;                  // Select I/O pad function for P36
page_p_sfr_write();             // Write data to DCON0
```

14. Interrupt

The **MG86FE/L508** has **8** interrupt sources with a four-level interrupt structure. There are several SFRs associated with the four-level interrupt. They are the IE, IP0L, IP0H, EIE1, EIP1L and EIP1H. The IP0H (Interrupt Priority 0 High) and EIP1H (Extended Interrupt Priority 1 High) registers make the four-level interrupt structure possible. The four priority level interrupt structure allows great flexibility in handling these interrupt sources.

14.1. Interrupt Structure

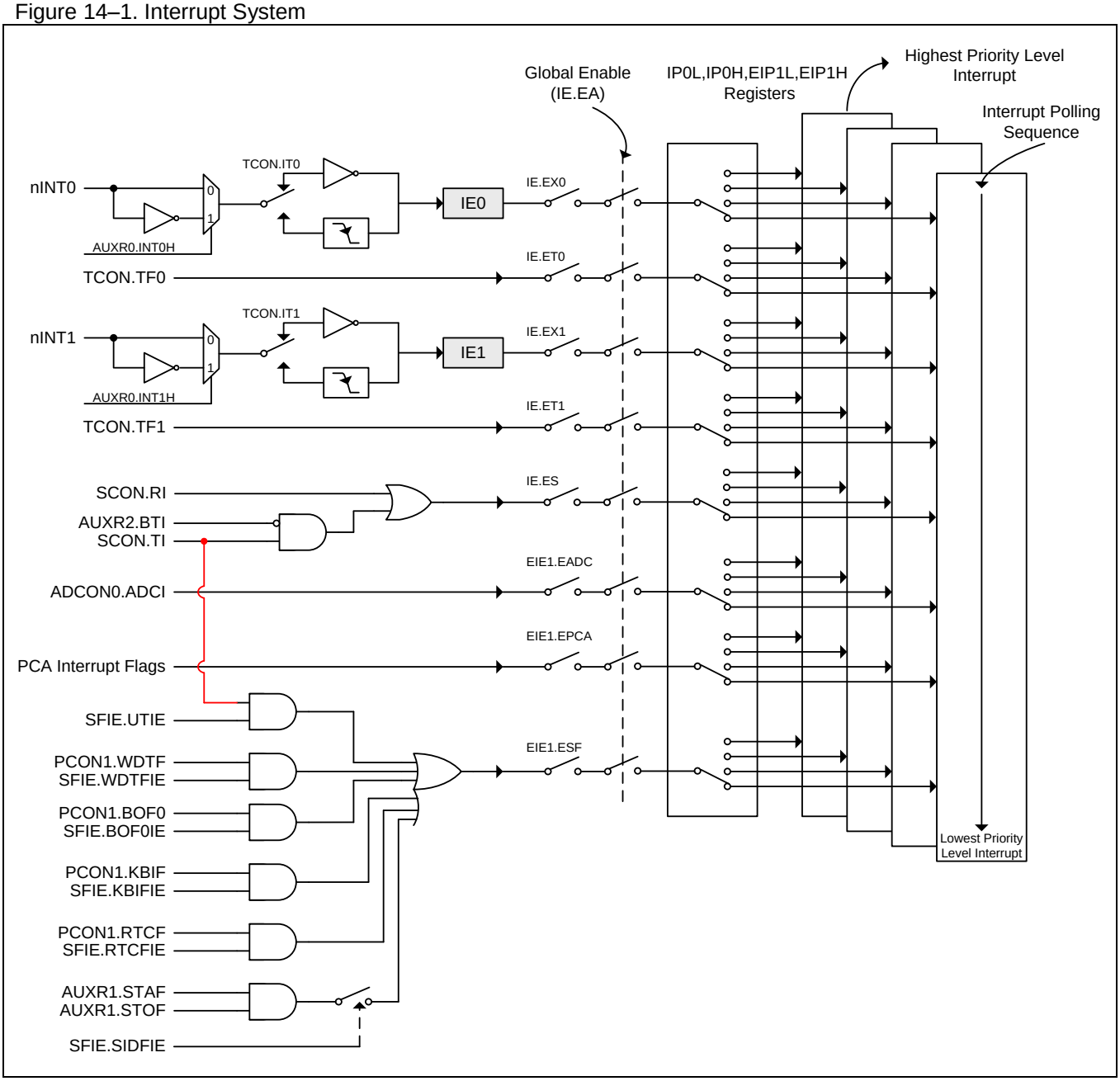
Table 14–1 lists all the interrupt sources. The 'Request Bits' are the interrupt flags that will generate an interrupt if it is enabled by setting the 'Enable Bit'. Of course, the global enable bit EA (in IE0 register) should have been set previously. The 'Request Bits' can be set or cleared by software, 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 14–1 shows the interrupt system. Each of these interrupts will be briefly described in the following sections.

Table 14–1. Interrupt Sources

No	Source Name	Enable Bit	Request Bits	Priority Bits	Polling Priority	Vector Address
#1	External Interrupt 0, nINT0	EX0	IE0	[PX0H, PX0L]	(Highest)	0003H
#2	Timer 0	ET0	TF0	[PT0H, PT0L]	…	000Bh
#3	External Interrupt 1, nINT1	EX1	IE1	[PX1H, PX1L]	...	0013H
#4	Timer 1	ET1	TF1	[PT1H, PT1L]	…	001BH
#5	Serial Port 0	ES	RI, TI	[PSH, PSL]	…	0023H
#6	ADC	EADC	ADCI	[PADCH, PADCL]	…	002BH
#7	PCA	EPCA	CF, CCFn (n=0~1)	[PPCAH, PPCAL]	…	0033H
#8	System Flag	ESF	BOF0,WDTF KBIF,RTCF STAF,STOF (TI)	[PSFH, PSFL]	(Lowest)	003BH

Figure 14–1. Interrupt System



14.2. Interrupt Source

Table 14–2. Interrupt flags

No	Source Name	Request Bits	Bit Location
#1	External Interrupt, nINT0	IE0	TCON.1
#2	Timer 0	TF0	TCON.5
#3	External Interrupt, nINT1	IE1	TCON.3
#4	Timer 1	TF1	TCON.7
#5	Serial Port 0	RI0 TI0	SCON0.0 SCON0.1
#6	ADC	ADCI	ADCON0.4
#7	PCA	CF, CCFn (n=0~1)	CCON.7 CCON.1~0
#8	System Flag	WDTF BOF0 KBIF RTCF STAF STOF (TI)	PCON1.0 PCON1.1 PCON1.3 PCON1.4 AUXR1.3 AUXR1.2 SCON.1

The external interrupt nINT0 and nINT1 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.

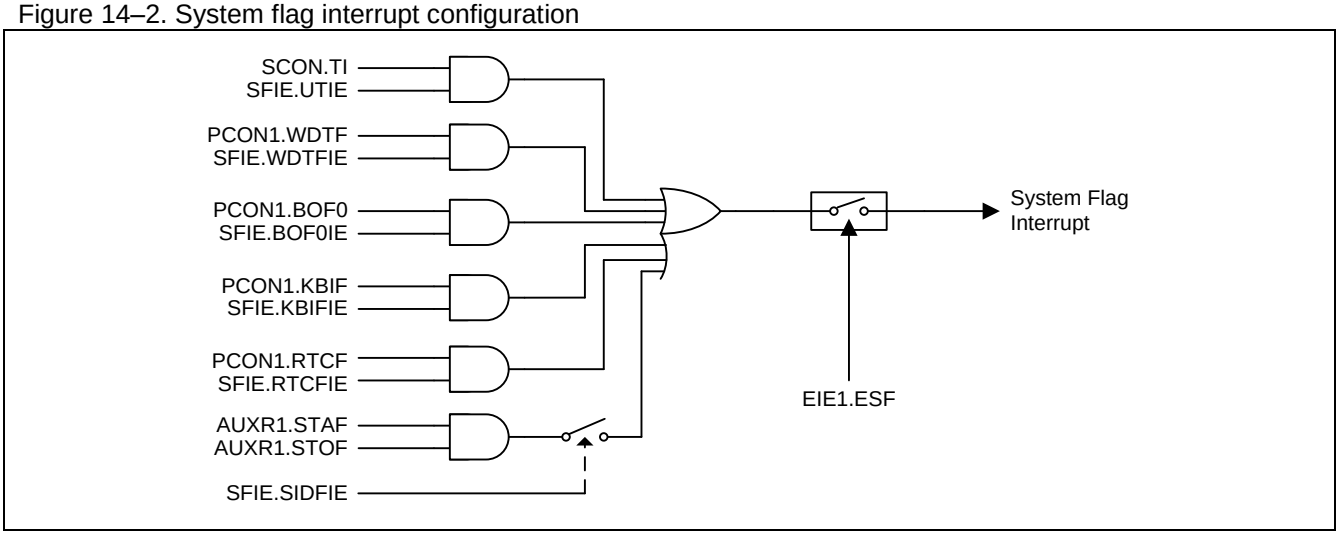
The serial port interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll RI and TI to determine which one to request service and it will be cleared by software.

The ADC interrupt is generated by ADCI in ADCON0. It will not be cleared by hardware when the service routine is vectored to.

The PCA interrupt is generated by the logical OR of CF, CCF1 and CCF0 in CCON. Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll these flags to determine which one to request service and it will be cleared by software.

The System Flag interrupt is generated by STAF, STOF, RTCF, KBIF, BOF0 and WDTF. STAF and STOF are set by serial interface detection and stored in AUXR1. The rest flags are stored in PCON1. RTCF is set by RTC counter overflow. KBIF is set by KBI event. BOF0 is set by on chip Brownout-Detector (BOD0) met the low voltage event. WDTF is set by Watch-Dog-Timer overflow. The Serial Port TI flag is optional to locate the interrupt vector shared with system flag interrupt which is enabled by UTIE set. These flags will not be cleared by hardware when the service routine is vectored to.

Figure 14–2. System flag interrupt configuration



All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. In other words, interrupts can be generated or pending interrupts can be canceled in software.

14.3. Interrupt Enable

Table 14–3. Interrupt enable control

No	Source Name	Enable Bit	Bit Location
#1	External Interrupt, nINT0	EX0	IE.0
#2	Timer 0	ET0	IE.1
#3	External Interrupt, nINT1	EX1	IE.2
#4	Timer 1	ET1	IE.3
#5	Serial Port 0	ES0	IE.4
#6	ADC	EADC	EIE1.1
#7	PCA	EPCA	EIE1.2
#8	System Flag	ESF & (UTIE, SDIFIE, RTCFIE, KBIFIE, BOF0IE, WDTFIE)	EIE1.3 & SFIE.7,6,4,3,1,0

There are **8** interrupt sources available in **MG86FE/L508**. Each of these interrupt sources can be individually enabled or disabled by setting or clearing an interrupt enable bit in the registers IE and EIE1. 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.

14.4. 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 14–1) determine the priority level of each interrupt. IP0L, IP0H, EIP1L and EIP1H are combined to 4-level priority interrupt. Table 14–4 shows the bit values and priority levels associated with each combination.

Table 14–4. Interrupt priority level

{IPH.x , IPL.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 IP**n**H and the other in IP**n**L 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 14–2 shows the internal polling sequence in the same priority level and the interrupt vector address.

14.5. 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, IP0L, IP0H, EIE1, EIP1L and EIP1H 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 or IP, then at least one or more instruction will be executed before any interrupt is vectored to.

14.6. Special Interrupt Vector for TI

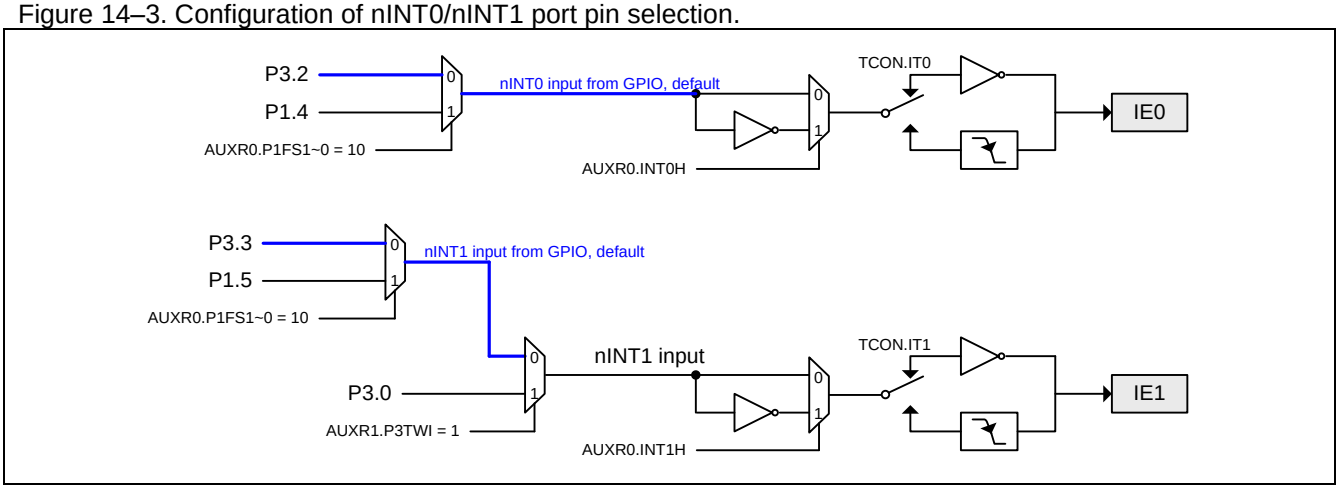
The serial port interrupt from TI flag can be masked by BTI (AUXR2.6). If BTI is set, set TI flag will not generate a serial port interrupt. The serial port interrupt only reflects the RI flag.

If UTIE (SFIE.7) is set, TI flag will be combined into System Flag Interrupt. In this mode, TI interrupt shares the interrupt vector with KBIF, BOF0 and WDTF in System Flag Interrupt.

14.7. nINT0/nINT1 Input Source Selection

The **MG86FE/L508** provides flexible nINT0 and nINT1 source selection to share the port pin input of on-chip serial interface. That will support the additional remote wakeup function for communication peripheral in power-down mode. The nINT0/nINT1 input can be routed to the interface pin to catch port change and set them as an interrupt input event to wakeup MCU. INT0H (AUXR0.0) and INT1H (AUXR0.1) configure the port change detection level on low/falling or high/rising event. In MCU power-down mode, both of the falling edge or rising edge configurations of the external interrupts are forced to level-sensitive operation.

Figure 14–3. Configuration of nINT0/nINT1 port pin selection.



14.8. Interrupt Register

TCON: Timer/Counter Control Register

SFR Page = Normal
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. If INT1H (AUXR0.1) is set, this bit specifies high level triggered on nINT1.
1: Set by software to specify falling edge triggered external interrupt 1. If INT1H (AUXR0.1) is set, this bit specifies rising edge triggered on nINT1.

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. If INT0H (AUXR0.0) is set, this bit specifies high level triggered on nINT0.
1: Set by software to specify falling edge triggered external interrupt 0. If INT0H (AUXR0.0) is set, this bit specifies rising edge triggered on nINT0.

IE: Interrupt Enable Register

SFR Page = Normal
SFR Address = 0xA8 RESET = 0xx0-0000

7	6	5	4	3	2	1	0
EA	--	--	ES	ET1	EX1	ET0	EX0
R/W	**W**	**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: Reserved. Software must write “0” on this bit when IE is written.

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

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.

EIE1: Extended Interrupt Enable 1 Register

SFR Page = Normal
SFR Address = 0xAD RESET = xxxx-0xxx

7	6	5	4	3	2	1	0
--	--	--	--	ESF	EPCA	EADC	--
W	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**	**W**

Bit 7~4: Reserved. Software must write “0” on these bits when EIE1 is written.

Bit 3: ESF, Enable System Flag interrupt.
0: Disable the interrupt when the group of {KBIF, BOF0, WDTF} in PCON1 or TI in SCON is set
1: Enable the interrupt of the flags of {KBIF, BOF0, WDTF} in PCON1 or TI in SCON when the associated system flag interrupt is enabled in SFIE.

Bit 2: EPCA, Enable PCA interrupt.
0: Disable PCA interrupt.
1: Enable PCA interrupt.

Bit 1: EADC, Enable ADC Interrupt.
0: Disable the interrupt when ADCON0.ADCI is set in ADC module.
1: Enable the interrupt when ACCON0.ADCI is set in ADC module.

Bit 0: Reserved. Software must write “0” on this bit when EIE1 is written.

SFIE: System Flag Interrupt Enable Register

SFR Page = Normal
SFR Address = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	RTCFIE	KBIFIE	--	BOF0IE	WDTFIE
R/W	**W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: UART TI Enabled in system flag interrupt.
0: Disable the interrupt vector sharing for TI in system flag interrupt.
1: Set TI flag will share the interrupt vector with system flag interrupt.

Bit 6: SDIFIE, Serial Interface Detection Flag Interrupt Enabled.
0: Disable SDIF(STAF or STOF) interrupt.
1: Enable SDIF(STAF or STOF) interrupt.

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

Bit 4: RTCFIE, Enable RTCF (PCON1.4) Interrupt.
0: Disable RTCF interrupt.
1: Enable RTCF interrupt.

Bit 3: KBIFIE, Enable KBIF (PCON1.3) Interrupt.
0: Disable KBIF interrupt.
1: Enable KBIF interrupt.

Bit 2: Reserved. Software must write “0” on this bit when SFIE is written.

Bit 1: BOF0IE, Enable BOF0 (PCON1.1) Interrupt.
0: Disable BOF0 interrupt.
1: Enable BOF0 interrupt.

Bit 0: WDTFIE, Enable WDTF (PCON1.0) Interrupt.
0: Disable WDTF interrupt.
1: Enable WDTF interrupt.

IP0L: Interrupt Priority 0 Low Register

SFR Page = Normal
SFR Address = 0xB8 RESET = xxx0-0000

7	6	5	4	3	2	1	0
--	--	--	PSL	PT1L	PX1L	PT0L	PX0L
W	**W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~5: Reserved. Software must write “0” on these bits when IP0L is written.
Bit 4: PSL, Serial port interrupt priority-L register.
Bit 3: PT1L, Timer 1 interrupt priority-L register.
Bit 2: PX1L, external interrupt 1 priority-L register.
Bit 1: PT0L, Timer 0 interrupt priority-L register.
Bit 0: PX0L, external interrupt 0 priority-L register.

IP0H: Interrupt Priority 0 High Register

SFR Page = Normal
SFR Address = 0xB7 RESET = xxx0-0000

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

Bit 7~5: Reserved. Software must write “0” on these bits when IP0H is written.
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.

EIP1L: Extended Interrupt Priority 1 Low Register

SFR Page = Normal
SFR Address = 0xAE RESET = xxxx-000x

7	6	5	4	3	2	1	0
--	--	--	--	PSFL	PPCAL	PADCL	--
W	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**	**W**

Bit 7~4: Reserved. Software must write “0” on these bits when EIP1L is written.
Bit 3: PSFL, system flag interrupt priority-L register.
Bit 2: PPCAL, PCA interrupt priority-L register.
Bit 1: PADCL, ADC interrupt priority-L register.
Bit 0: Reserved. Software must write “0” on this bit when EIP1L is written.

EIP1H: Extended Interrupt Priority 1 High Register

SFR Page = Normal
SFR Address = 0xAF RESET = xxxx-000x

7	6	5	4	3	2	1	0
--	--	--	--	PSFH	PPCAH	PADCH	--
W	**W**	**W**	**W**	**R/W**	**W**	**W**	**W**

Bit 7~4: Reserved. Software must write “0” on these bits when EIP1H is written.
Bit 3: PSFH, system flag interrupt priority-H register.
Bit 2: PPCAH, PCA interrupt priority-H register.
Bit 1: PADCH, ADC interrupt priority-H register.
Bit 0: Reserved. Software must write “0” on this bit when EIP1H is written.

AUXR0: Auxiliary Register 0

SFR Page = Normal

SFR Address = 0xA1 RESET = 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	**INT1H**	**INT0H**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1: INT1H, INT1 High/Rising trigger enable.
0: Remain INT1 triggered on low level or falling edge on nINT1 port pin.
1: Set INT1 triggered on high level or rising edge on nINT1 port pin.

Bit 0: INT0H, INT0 High/Rising trigger enable.
0: Remain INT0 triggered on low level or falling edge on nINT0 port pin.
1: Set INT0 triggered on high level or rising edge on nINT0 port pin.

14.9. Interrupt Sample Code

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

Assembly Code Example:

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

main:

      SETB      P32                          ;

      ORL       IP0L,#PX0L                   ; Select INT0 interrupt priority
      ORL       IP0H,#PX0H                   ;

      ORL       AUXR0,#INT0H                 ; Set INT0 High level active

      JB        P32,$                        ; Confirm P3.2 input low

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

      ORL       PCON0,#PD                    ; Set MCU into Power Down mode
```

C Code Example:

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

void main(void)
{
      P32 = 1;

      IP0L |= PX0L;                          // Select INT0 interrupt priority
      IP0H |= PX0H;

      AUXR0 |= INT0H;                        // Set INT0 High level active

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

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

      PCON0 |= PD;                           // Set MCU into Power Down mode
}
```

15. Timers/Counters

MG86FE/L508 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. AUXR2.T0X12 and AUXR2.T1X12 are the function for Timer 0/1 to set the timer rate on every clock cycle. The AUXR0.T0XL is combined with T0X12 to select additional prescaler value, SYSCLK/48 and SYSCLK/192, for Timer 0 clock input.

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.

15.1. Timer0 and Timer1

15.1.1. Mode 0 Structure

The timer register is configured as a PWM generator. 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 PWM function of Timer 0/1 is shown in Figure 15–1 and Figure 15–2.

Figure 15–1. Timer 0 Mode 0 Structure

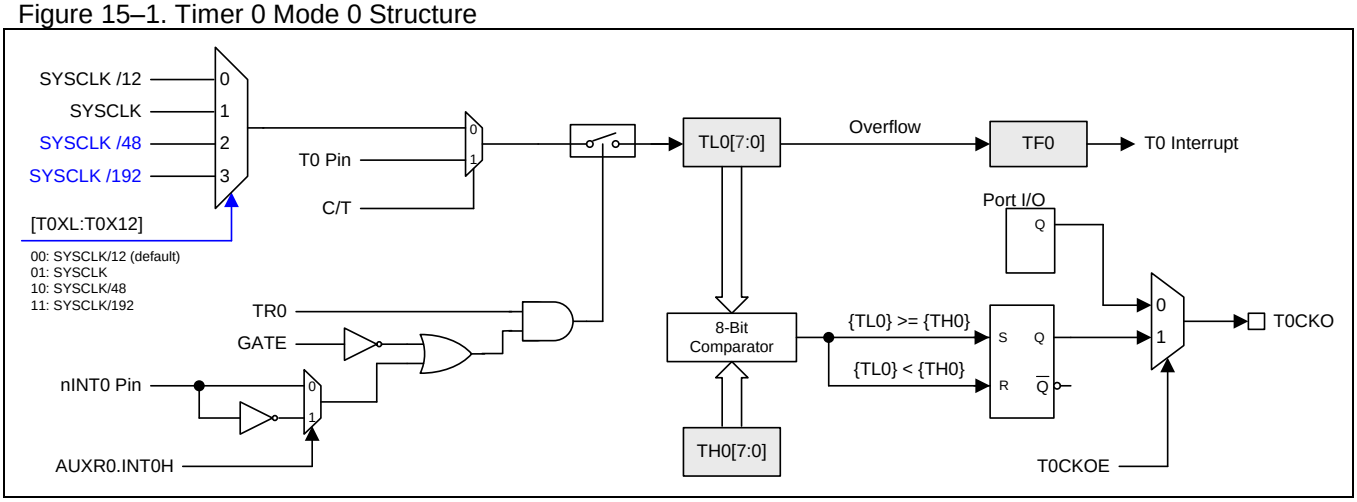
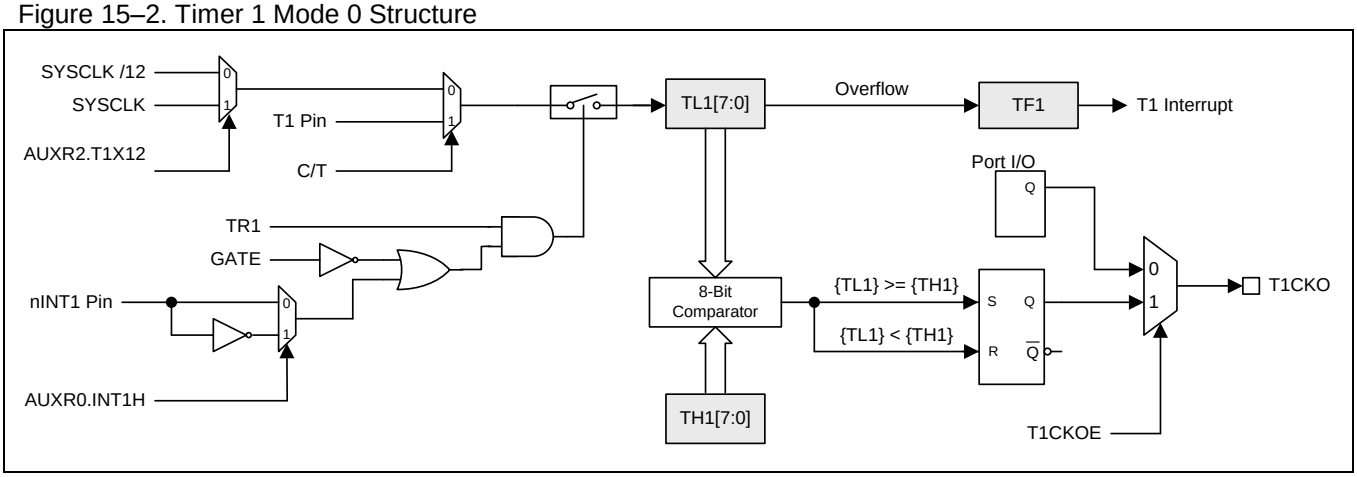


Figure 15–2. Timer 1 Mode 0 Structure



15.1.2. 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 15–3 and Figure 15–4 show the mode 1 structure of Timer 0 and Timer 1.

Figure 15–3. Timer 0 Mode 1 Structure

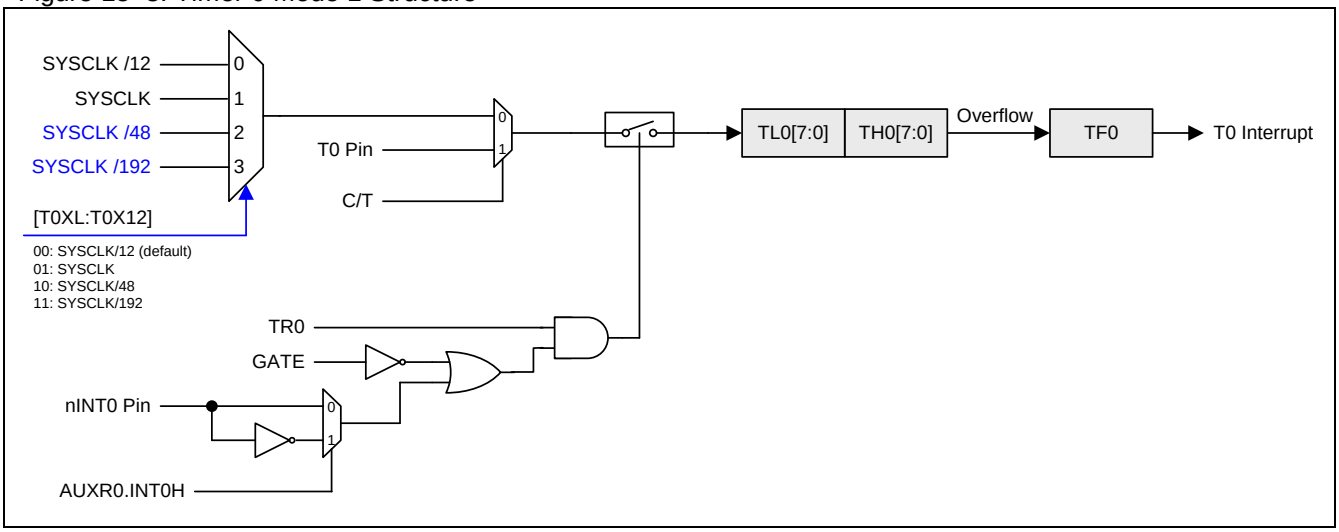
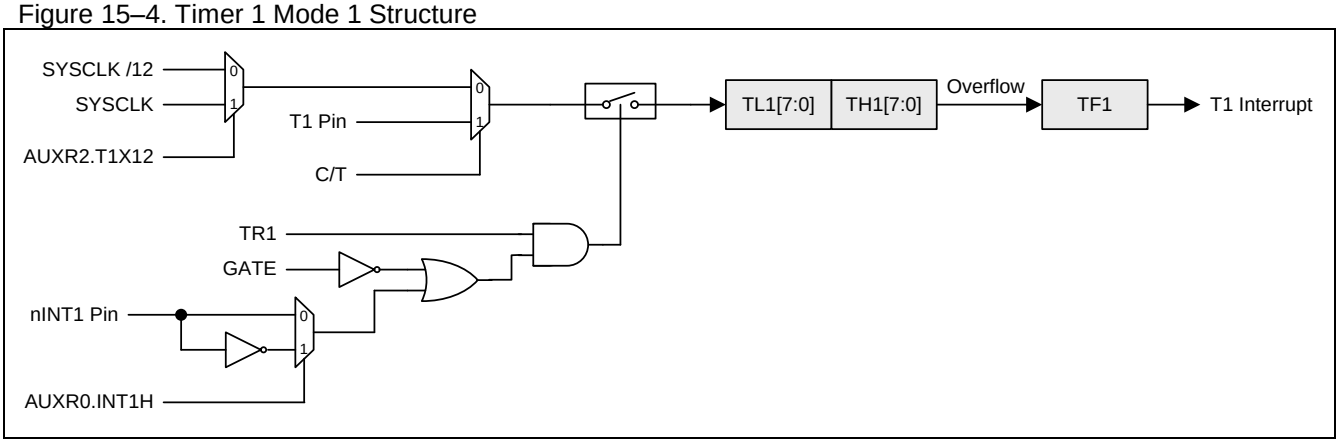


Figure 15–4. Timer 1 Mode 1 Structure



15.1.3. 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 15–5. Timer 0 Mode 2 Structure

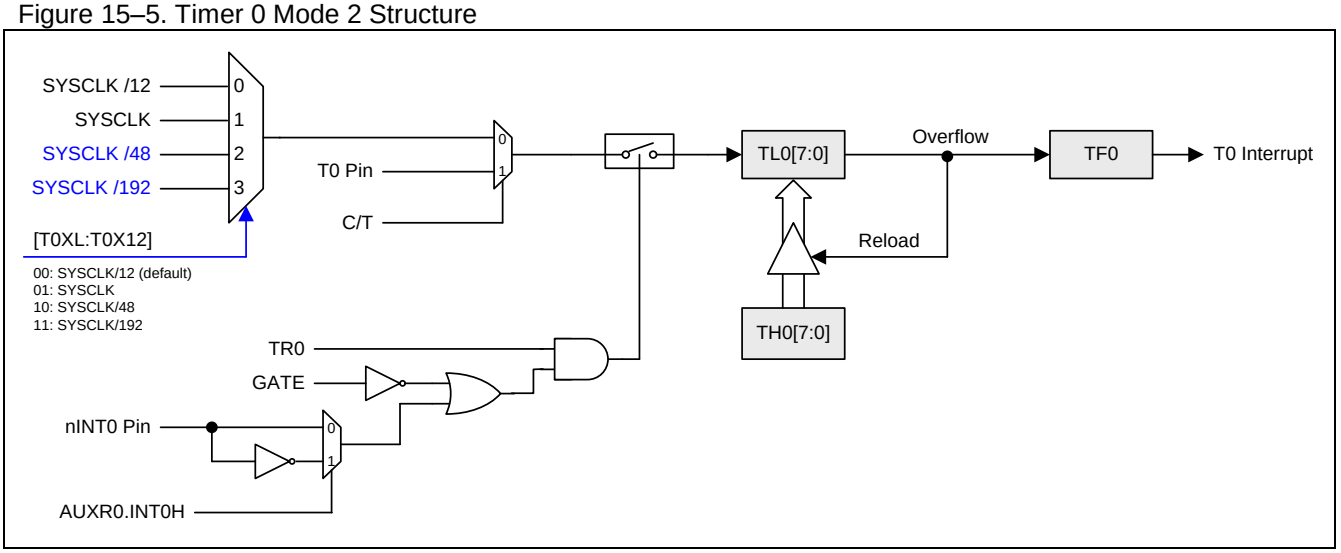
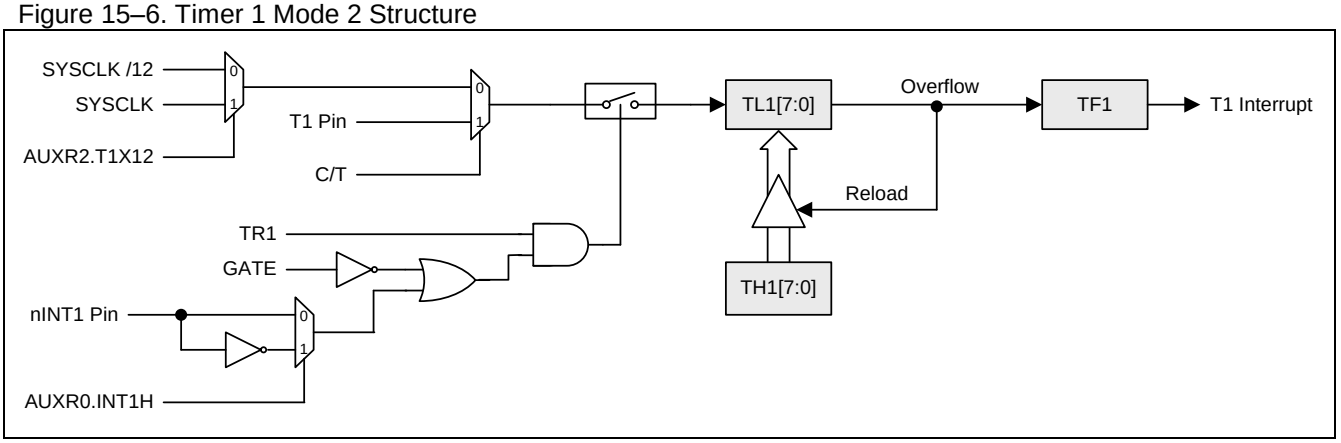


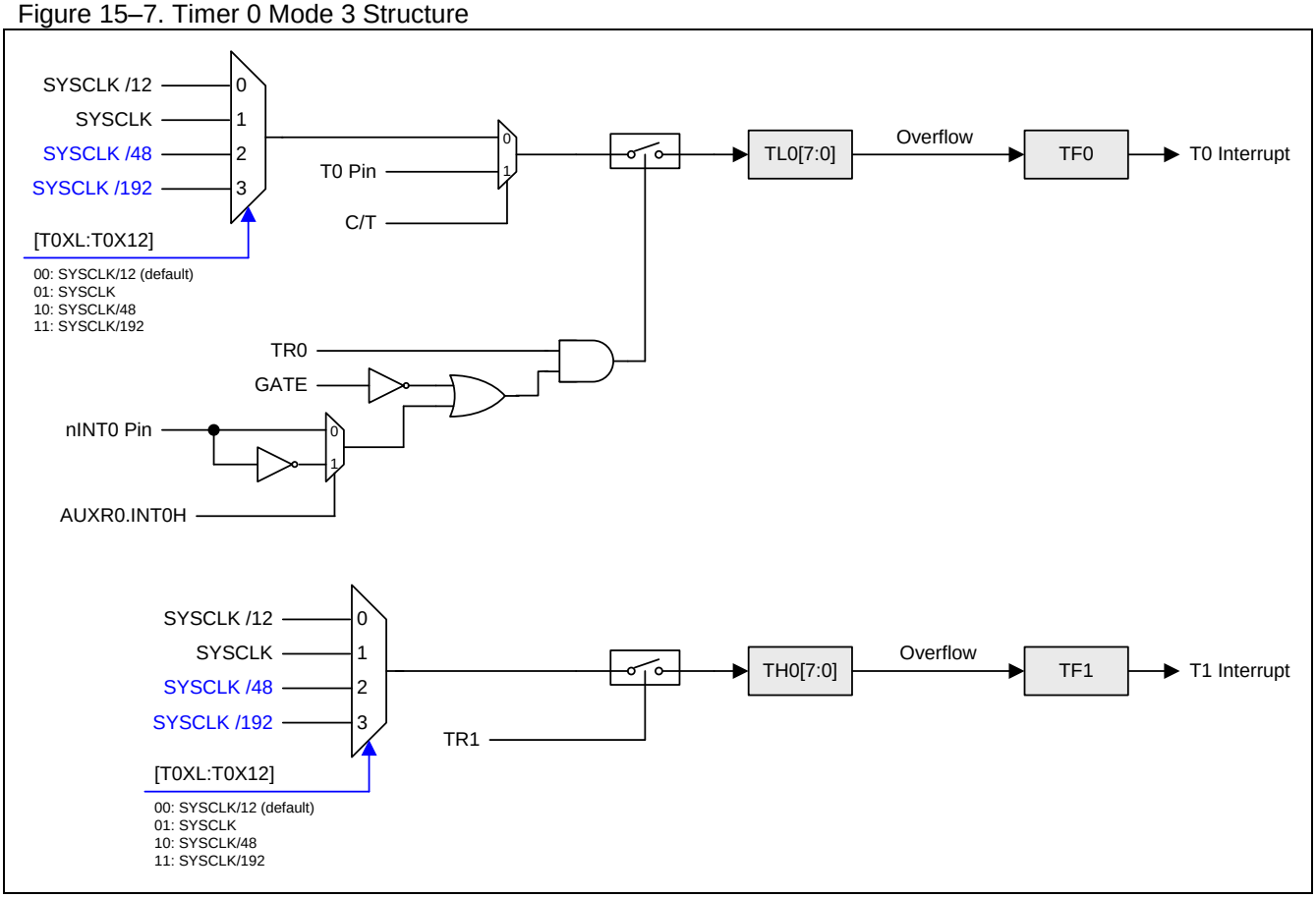
Figure 15–6. Timer 1 Mode 2 Structure



15.1.4. 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 (cannot be external event counter) and take over the use of TR1, TF1 from Timer1. TH0 now controls the Timer1 interrupt.

Figure 15–7. Timer 0 Mode 3 Structure



15.1.5. Timer 0/1 Programmable Clock-Out

Timer 0 and Timer 1 have a Clock-Out Mode (while C/Tx=0 & TxCKOE=1). In this mode, Timer 0 or Timer 1 operates as 8-bit auto-reload timer for a programmable clock generator with 50% duty-cycle. The generated clocks come out on P3.4 (T0CKO) and P3.5 (T1CKO) individually. The input clock (SYSCLK/12, SYSCLK, SYSCLK/48 or SYSCLK/192) increments the 8-bit timer, TL0, in Timer 0 module. The input clock (SYSCLK/12 or SYSCLK) increments the 8-bit timer, TL1, in Timer 1 module. The timer repeatedly counts to overflow from a loaded value. Once overflows occur, the contents of (TH0 and TH1) are loaded into (TL0, TL1) for the consecutive counting. The following formula gives the clock-out frequency:

Figure 15–8. Timer 0 clock out equation

$$\text{T0 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (256 - \text{THx})}$$

; n=24, if {T0XL,T0X12}=00
; n=2, if {T0XL,T0X12}=01
; n=96, if {T0XL,T0X12}=10
; n=384, if {T0XL,T0X12}=11
; C/T = 0

Figure 15–9. Timer 0 clock out equation

$$\text{T1 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (256 - \text{TH1})}$$

; n=24, if T1X12=0
; n=2, if T1X12=1
; C/T = 0

Note:
(1) Timer 0/1 overflow flag, TF0/1, will be set when Timer 0/1 overflows but not generate interrupt.
(2) For SYSCLK=12MHz & TxX12=0, Timer 0/1 has a programmable output frequency range from 1.95KHz to 500KHz.
(3) For SYSCLK=12MHz & TxX12=1, Timer 0/1 has a programmable output frequency range from 23.43KHz to 6MHz.
(4) For SYSCLK=12MHz, T0X12=0 & T0XL=1, Timer 0 has a programmable output frequency range from 488Hz to 125KHz.
(5) For SYSCLK=12MHz, TxX12=1 & T0XL=1, Timer 0 has a programmable output frequency range from 122Hz to 31.25KHz.

Figure 15–10. Timer 0 in Clock Output Mode

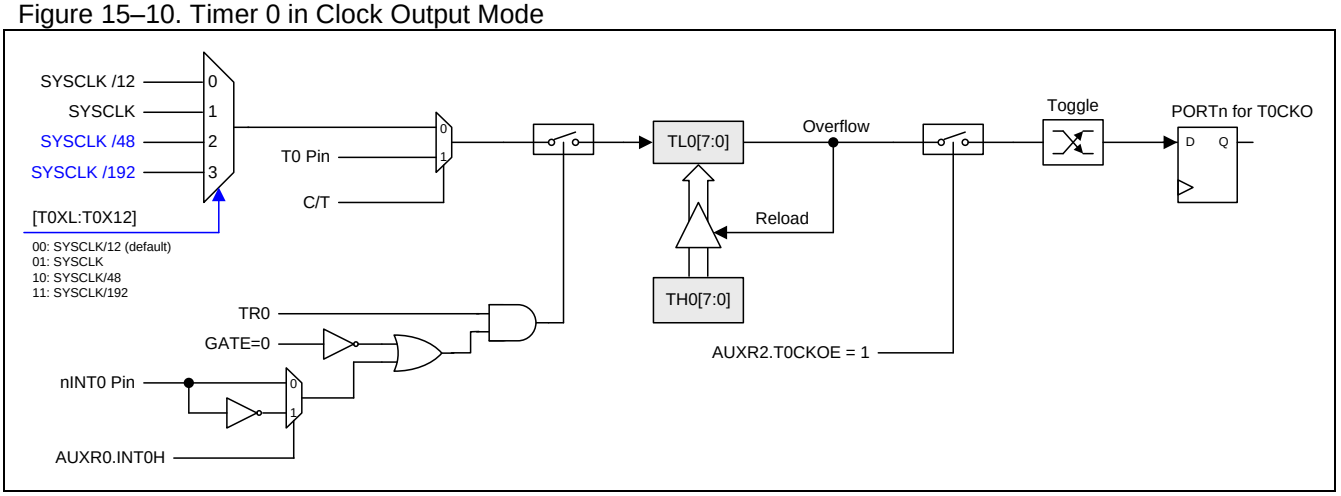
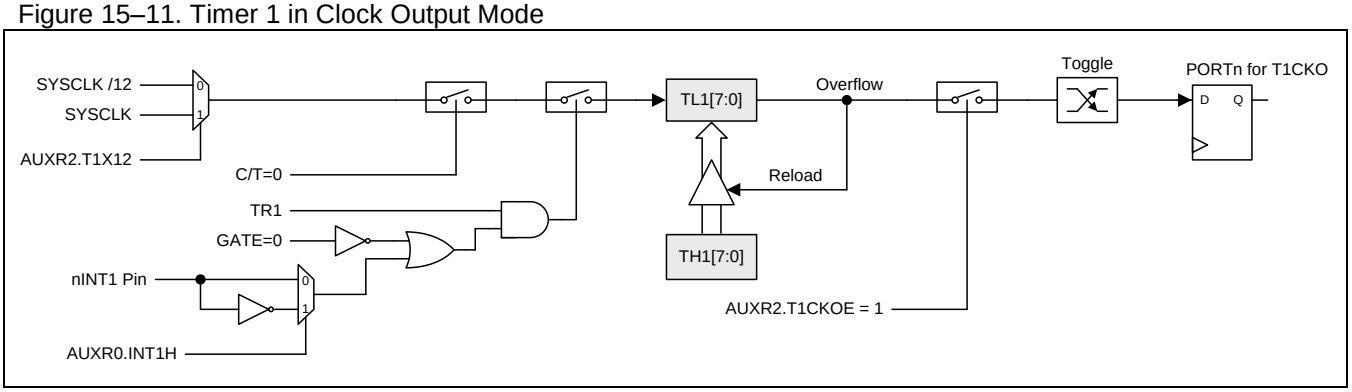


Figure 15–11. Timer 1 in Clock Output Mode



How to Program Timer 0/1 in Clock-out Mode

- Select AUXR2.T0X12 and AUXR0.T0XL bits decide the Timer 0 clock source. Or select T1X12 in AUXR2 register to decide the Timer 1 clock source.
- Set T0CKOE/T1CKOE bit in AUXR2 register.
- Clear C/T bit in TMOD register.
- Determine the 8-bit reload value from the formula and enter it in the TH0/TH1 register.
- Enter the same reload value as the initial value in the TL0/TL1 register.
- Set TR0/TR1 bit in TCON register to start the Timer 0/1.

In the Clock-Out mode, Timer 0/1 rollovers will not generate an interrupt. This is similar to when Timer 1 is used as a baud-rate generator. It is possible to use Timer 1 as a baud rate generator and a clock generator simultaneously. Note, however, that the baud-rate and the clock-out frequency depend on the same overflow rate of Timer 1.

15.1.6. Timer0/1 Register

TCON: Timer/Counter Control Register

SFR Page = Normal
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 Page = Normal
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	8-bit PWM generator 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 Page = Normal
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 Page = Normal

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 Page = Normal

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 Page = Normal

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**

AUXR2: Auxiliary Register 2

SFR Page = Normal

SFR Address = 0xA3 RESET = 0000-0000

7	6	5	4	3	2	1	0
--	BTI	URM0X3	SM3	T1X12	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: 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 2: 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.

T0XL, T0X12	Timer 0 Clock Selection
0 0	SYSCLK/12
0 1	SYSCLK
1 0	SYSCLK/48
1 1	SYSCLK/192

Bit 1: T1CKOE, Timer 1 Clock Output Enable.
0: Disable Timer 1 clock output.
1: Enable Timer 1 clock output on P3.5.

Bit 0: T0CKOE, Timer 0 Clock Output Enable.
0: Disable Timer 0 clock output.
1: Enable Timer 0 clock output on P3.4.

15.1.7. Timer0/1 Sample Code

(1). Required Function: IDLE mode with T0 wake-up frequency 640Hz, SYSCLK = ILRCO

Assembly Code Example:

```
    ORG     0000Bh
time0_isr:
    to do…
    RETI

main:                                       ; (unsigned short value)
    //Switch Sysclk to ILRCO
    MOV     IFADRL,#(CKCON2)                ; Index Page-P address to CKCON2
    CALL    _page_p_sfr_read                ; Read CKCON2 data

    ANL     IFD,#~(OSCS1 | OSCS0)           ; Switch OSCin source to ILRCO
    ORL     IFD,#(OSCS1)
    CALL    _page_p_sfr_write               ; Write data to CKCON2

    ANL     IFD,#~(XTALE | IHRCOE)          ; Disable XTAL and IHRCO
    CALL    _page_p_sfr_write               ; Write data to CKCON2

    MOV     IFADRL,#(DCON0)                 ; Index Page-P address to DCON0
    CALL    _page_p_sfr_read                ; Read DCON0 data

    ANL     IFD,#~(HSE)                     ; Disable HSE when SYSCLK ≤ 6MHz for power saving
    CALL    _page_p_sfr_write               ; Write data to DCON0

    ANL     CKCON0,#(AFS)                   ; Select SCKS[2:0] = 0 = OSCin/1

    ORL     AUXR2,#T0X12                    ; Select SYSCLK/1 for Timer 0 clock input
    ANL     AUXR0,#~T0XL                    ;

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

    ORL     IP0L,#PT0L                      ; Select Timer 0 interrupt priority
    ORL     IP0H,#PT0H                      ;

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

    SETB    TR0                             ; Start Timer 0 running

    ORL     PCON0,#IDL                      ; Set MCU into IDLE mode
```

C Code Example:

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

void main(void)
{
    IFADRL = CKCON2;                        // Index Page-P address to CKCON2
    page_p_sfr_read();                      // Read CKCON2 data.

    IFD = ~(OSCS1 | OSCS0);                 // Switch OSCin source to ILRCO
    IFD |= OSCS1;
    page_p_sfr_write();                     // Write data to CKCON2

    IFD &= ~(XTALE | IHRCOE);               // Disable XTAL and IHRCO
```

```
    page_p_sfr_write();                     // Write data to CKCON2

    IFADRL = DCON0;                         // Index Page-P address to DCON0
    page_p_sfr_read();                      // Read DCON0 data

    IFD &= ~HSE;                            // Disable HSE when SYSCLK ≤ 6MHz for power saving
    page_p_sfr_write();                     // Write data to DCON0

    CKCON0 &= AFS;                          // Select SCKS[2:0] = 0 = OSCin/1

    AUXR2 |= T0X12;                         // Select SYSCLK/1 for Timer 0 clock input
    AUXR0 &= ~T0XL;

    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

    IP0L |= PT0L;                           // Select Timer 0 interrupt priority
    IP0H |= PT0H;

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

    TR0 = 1;                                // Start Timer 0 running

    PCON0=IDL;                              // Set MCU into IDLE mode
}
```

(2). Required Function: Set Timer 0 clock output by SYSCLK/48 input

Assembly Code Example:

```
    CLR     TR0                             ;

    ANL     P3M0,#0EFh                      ; Set P3.4(T0CKO) to push-pull output
    ORL     P3M1,#010h                      ;
    ORL     AUXR2,#T0CKOE                   ; Enable T0CKO

    ANL     AUXR2,#~T0X12                   ; Select SYSCLK/48 for Timer 0 clock input
    ORL     AUXR0,#T0XL                     ;

    MOV     TH0,#0FFh                       ;
    MOV     TL0,#0FFh                       ;

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

    SETB    TR0                             ; Start Timer 0 running
```

C Code Example:

```
    TR0 = 0;

    P3M0 &= 0xEF;                           // Set P3.4(T0CKO) to push-pull output
    P3M1 |= 0x10;
    AUXR2 |= T0CKOE;                        // Enable T0CKO

    AUXR2 &= ~T0X12;                        // Select SYSCLK/48 for Timer 0 clock input
    AUXR0 |= T0XL;

    TH0 = TL0 = 0xFF;

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

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

(3). Required Function: Set Timer 1 clock output by SYSCLK input

Assembly Code Example:

```
ORL     P3M1,#020h                  ; Set P3.5(T1CKO) to push-pull output
ANL     P3M0,#0DFh                  ;

ORL     AUXR2,#(T1X12|T1CKOE)       ; Select SYSCLK for Timer 1 clock input
                                    ; Enable T1CKO

MOV     TH1,#0FFh                   ;
MOV     TL1,#0FFh                   ;

ANL     TMOD,#00Fh                  ; Set Timer 1 to Mode 2
ORL     TMOD,#T1M1                  ;

SETB    TR1                         ; Start Timer 1 running
```

C Code Example:

```
P3M1 |= 0x20;                       // Set P3.5(T1CKO) to push-pull output
P3M0 &= 0xDF;

AUXR2 |= (T1X12|T1CKOE);            // Select SYSCLK for Timer 1 clock input
                                    // Enable T1CKO

TH1 = TL1 = 0xFF;

TMOD &= 0x0F;                       // Set Timer 1 to Mode 2
TMOD |= T1M1;

TR1 = 1;                            // Start Timer 1 running
```

(4). Required Function: Set Tim0 Mode 0 to output 25% PWM, PWM Freq.= 46.875K, SYSCLK = 12MHz

Assembly Code Example:

```
CLR     TR0                         ;

ORL     P3M1,#010h                  ; Set P3.4(T0CKO) to push-pull output
ANL     P3M0,#0EFh                  ;

ANL     AUXR0,#~T0XL                ;
ORL     AUXR2,#(T0X12|T0CKOE);      ; Select SYSCLK for Timer 0 clock input
                                    ; Enable T0CKO

MOV     TH0,#0C0h                   ; Set PWM duty = 25%
MOV     TL0,#000h                   ;

ANL     TMOD,#0F0h                  ; Set Timer 0 to Mode 0 for PWM function

SETB    TR0                         ; Start Timer 0 running
```

C Code Example:

```
TR0 = 0;

P3M0 &= 0xEF;                       // Set P3.4(T0CKO) to push-pull output
P3M1 |= 0x10;

AUXR0 &= ~T0XL;
AUXR2 |= (T0X12|T0CKOE);            // Select SYSCLK for Timer 0 clock input
                                    // Enable T0CKO

TH0 = 0xC0;                         // Set PWM duty = 25%
TL0 = 0x00;

TMOD &= 0xF0;                       // Set Timer 0 to Mode 0 for PWM function
```

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

(5). Required Function: Set Tim1 Mode 0 to output 75% PWM, PWM Freq.= 46.875K, SYSCLK = 12MHz

Assembly Code Example:

```
CLR     TR1                                 ;

ORL     P3M1,#020h                          ; Set P3.5(T1CKO) to push-pull output
ANL     P3M0,#0DFh                          ;

ORL     AUXR2,#(T1X12|T1CKOE);              ; Select SYSCLK for Timer 1 clock input
                                            ; Enable T1CKO

MOV     TH1,#040h                           ; Set PWM duty = 75%
MOV     TL1,#000h                           ;

ANL     TMOD,#00Fh                          ; Set Timer 1 to Mode 0 for PWM function

SETB    TR1                                 ; Start Timer 1 running
```

C Code Example:

```
TR1 = 0;

P3M0 &= 0xDF;                               // Set P3.5(T1CKO) to push-pull output
P3M1 |= 0x20;

AUXR2 |= (T1X12|T1CKOE);                    // Select SYSCLK for Timer 1 clock input
                                            // Enable T1CKO

TH1 = 0x40;                                 // Set PWM duty = 75%
TL1 = 0x00;

TMOD &= 0x0F;                               // Set Timer 1 to Mode 0 for PWM function

TR1 = 1;                                    // Start Timer 1 running
```

16. Serial Port (UART)

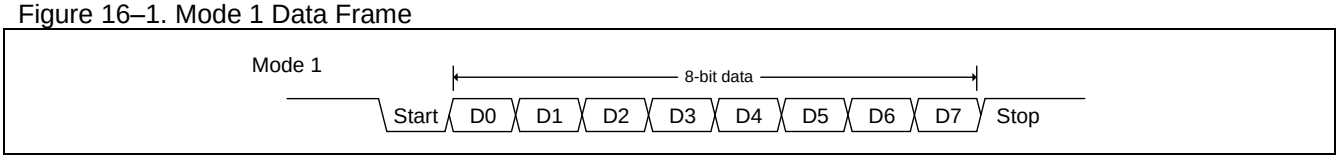
The serial port of **MG86FE/L508** support full-duplex transmission, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the register. However, if the first byte still 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 **1/4** the system clock frequency by URM0X3 setting in AUXR2 register. In **MG86FE/L508**, the clock polarity of serial port Mode 0 can be selected by software. It is decided by P3.1 state before serial data shift in or shift out. Figure 16–4 and Figure 16–5 show the clock polarity waveform in Mode 0.

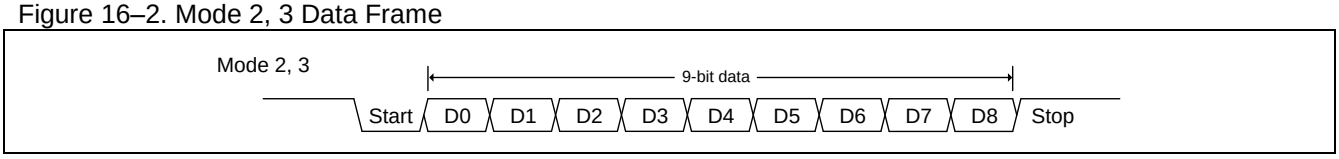
Mode 1: 10 bits are transmitted through TXD or received through RXD. The frame data includes a start bit (0), 8 data bits (LSB first), and a stop bit (1), as shown in Figure 16–1. On receive, the stop bit would be loaded into RB8 in SCON register. The baud rate is variable.

Figure 16–1. 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 16–2. 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 16–2. 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.

16.1. 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 **1/4** the system clock frequency by URM0X3 setting in AUXR2 register. Figure 16–3 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 16–4 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 16–5 shows the reception waveform in Mode 0. The clock polarity can be selected by software setting on P3.1 data latch before serial transfer shifted. If P3.1 is set to logic high, the clock polarity is same as standard 8051. If P3.1 data latch is cleared to logic low, the clock polarity is inverted to standard 8051 UART Mode 0.

Figure 16–3. Serial Port Mode 0

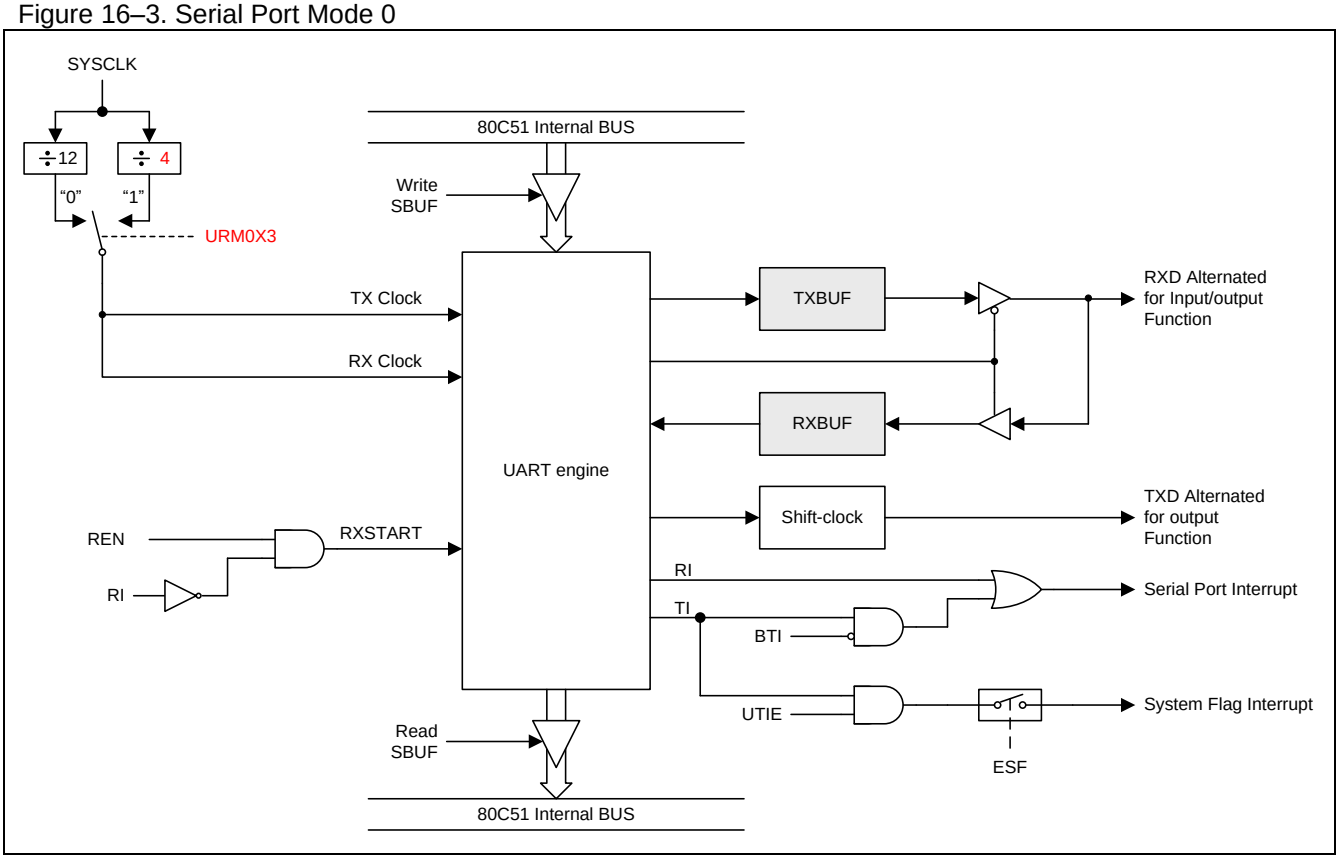


Figure 16–4. Mode 0 Transmission Waveform

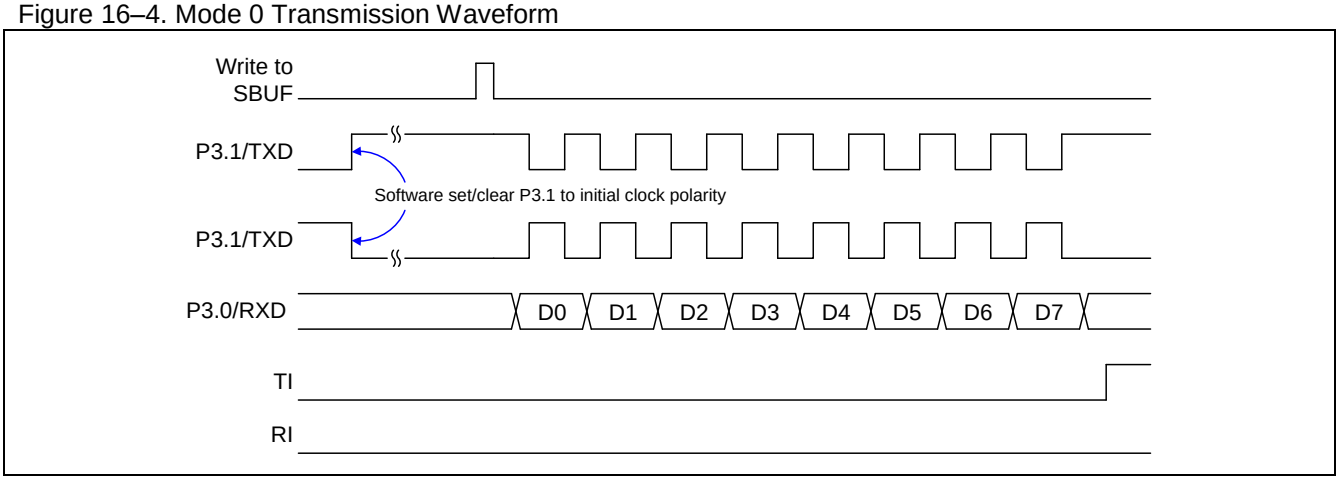
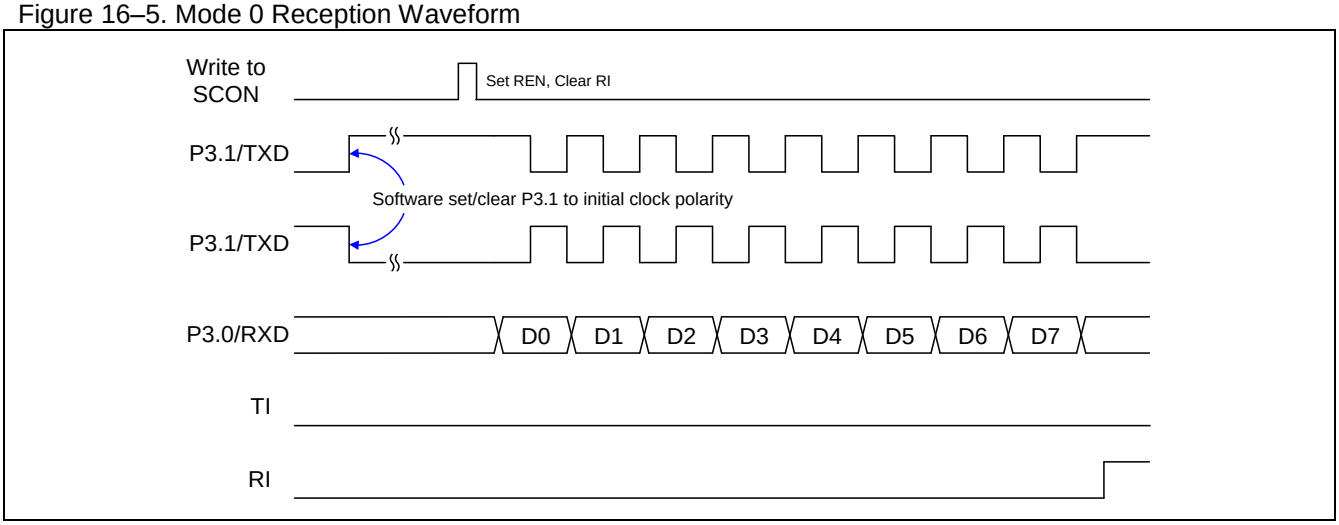


Figure 16–5. Mode 0 Reception Waveform



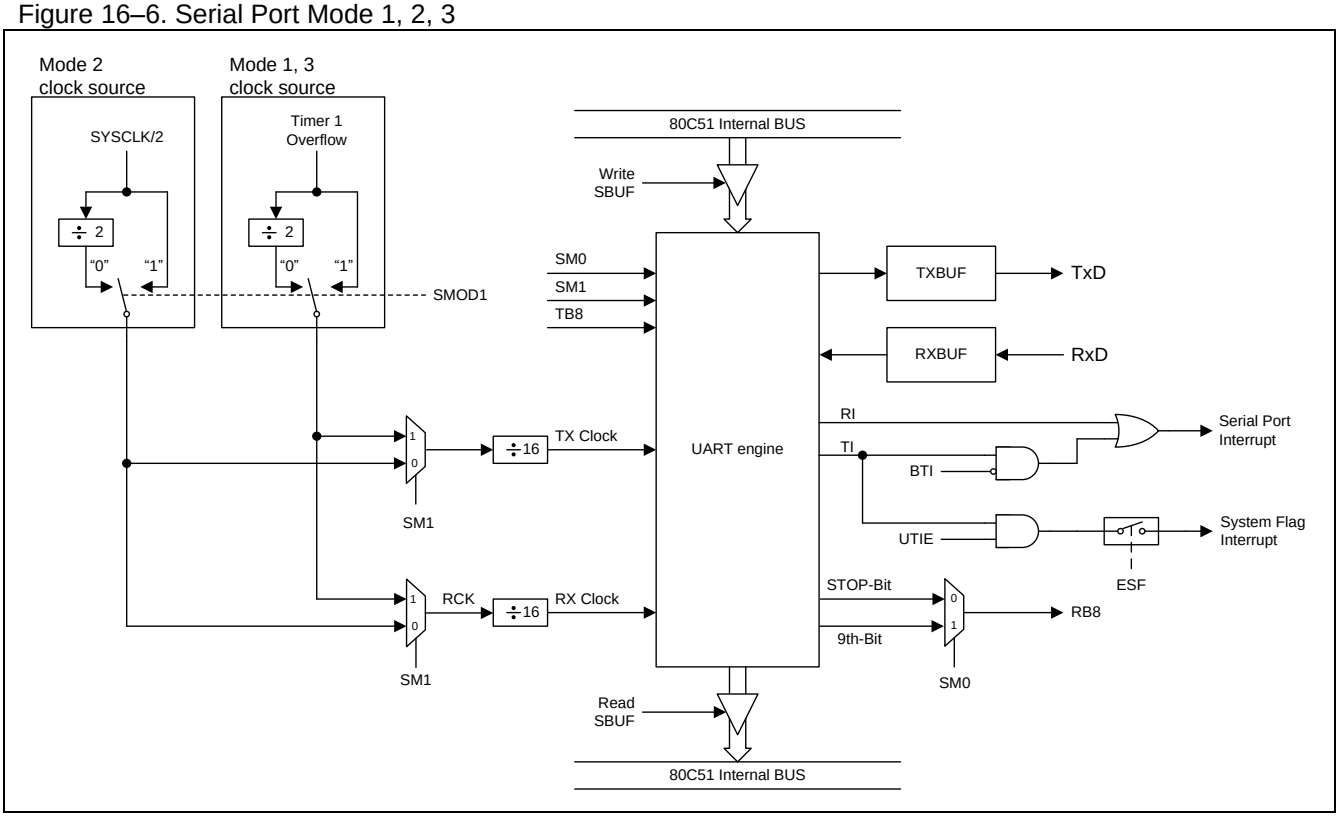
16.2. Serial Port Mode 1

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 16–1 shows the data frame in Mode 1 and Figure 16–6 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 16–1 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 16–6. Serial Port Mode 1, 2, 3



16.3. 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 one of 1/16, 1/32 or 1/64 the system clock frequency in Mode 2. Mode 3 may have a variable baud rate generated from Timer 1 or Timer 2.

Figure 16–2 shows the data frame in Mode 2 and Mode 3. Figure 16–6 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 shift 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 16–2 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.

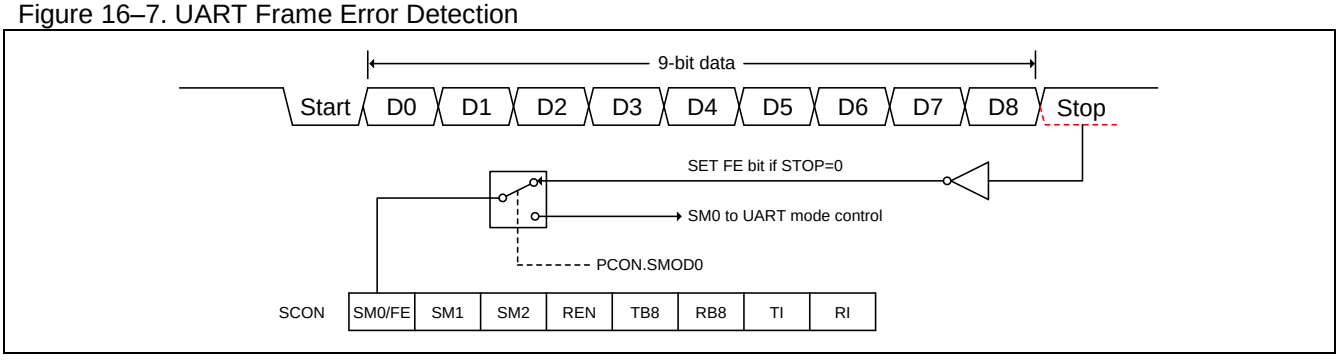
Reception is initiated when the UART engine detected 1-to-0 transition at RXD sampled by RCK. The data on the RXD pin would be sampled by Bit Detector in UART engine. After the end of 9th data bit reception, RI would be asserted by hardware to indicate the end of data reception and load the 9th data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

16.4. 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 16–7.

Figure 16–7. UART Frame Error Detection



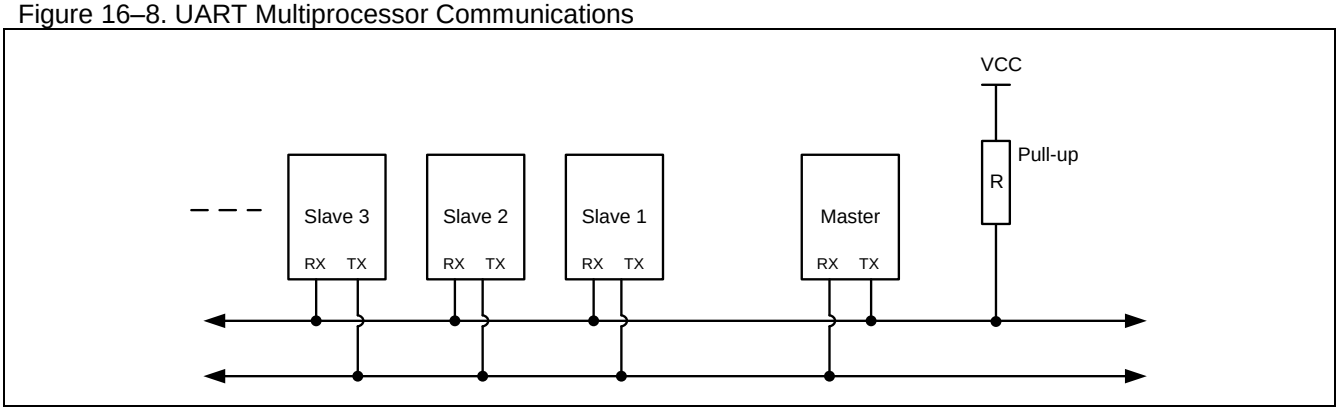
16.5. Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications as shown in Figure 16–8. 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 16–8. UART Multiprocessor Communications



16.6. 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 16–9. 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:

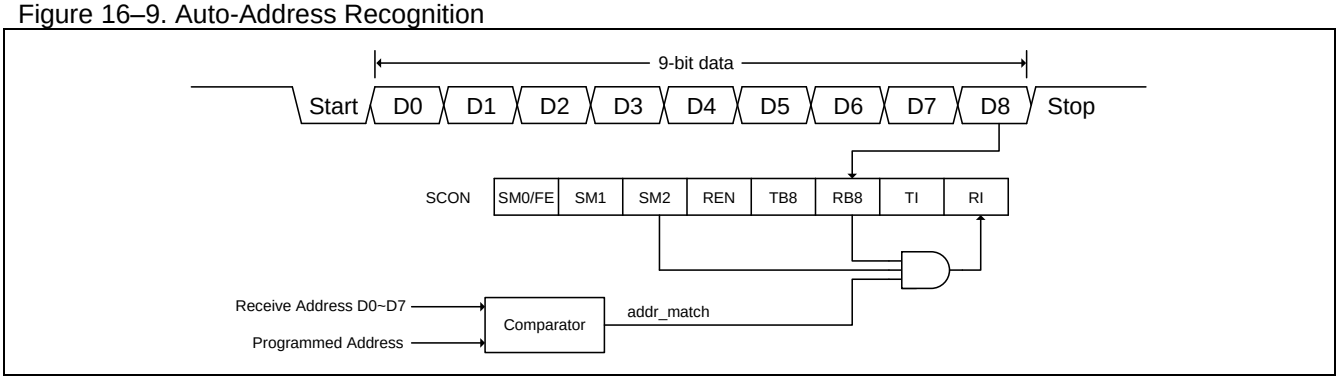
Slave 0	Slave 1	Slave 2
SADDR = 1100 0000	SADDR = 1110 0000	SADDR = 1110 0000
SADEN = 1111 1001	SADEN = 1111 1010	SADEN = 1111 1100
Given = 1100 0XX0	Given = 1110 0X0X	Given = 1110 00XX

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 16–9. 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.

16.7. Baud Rate Setting

Bits T1X12 and URM0X3 in AUXR2 register provide a new option for the baud rate setting, as listed below.

16.7.1. Baud Rate in Mode 0

Figure 16–10. Mode 0 baud rate equation

$$\text{Mode 0 Baud Rate} = \frac{F_{SYSCLK}}{n}$$; n=12, if URM0X3=0 ; n=4, if URM0X3=1

Note:

If URM0X3=0, the baud rate formula is as same as standard 8051.

Table 16–1. Serial Port Mode 0 baud rate example

SYSCLK	URM0X3	Mode 0 Baud Rate
12MHz	0	1M bps
12MHz	1	3M bps
24MHz	0	2M bps
24MHz	1	6M bps

16.7.2. Baud Rate in Mode 2

Figure 16–11. Mode 2 baud rate equation

$$\text{Mode 2 Baud Rate} = \frac{2^{SMOD1}}{64} \times (F_{SYSCLK})$$

Table 16–2. Serial Port Mode 2 baud rate example

SYSCLK	SMOD1	Mode 2 Baud Rate
22.1184MHz	0	345.6K bps
22.1184MHz	1	172.8K bps
24MHz	0	750K bps
24MHz	1	375K bps

16.7.3. Baud Rate in Mode 1 & 3

Using Timer 1 as the Baud Rate Generator

Figure 16–12. Mode 1/3 baud rate equation

$$\text{Mode 1, 3 Baud Rate} = \frac{2^{SMOD1}}{32} \times \frac{F_{SYSCLK}}{n \times (256 - TH1)}$$; n=12, if T1X12=0 ; n=1, if T1X12=1

Table 17-1 ~ Table 17-6 list various commonly used baud rates and how they can be obtained from Timer 1 in its 8-Bit Auto-Reload Mode.

Table 16–3. Timer 1 Generated Commonly Used Baud Rates @ F_{SYSCLK}=11.0592MHz

Baud Rate	**TH1**, the Reload Value					
	T1X12=0			**T1X12=1**		
	SMOD1=0	SMOD1=1	Error	SMOD1=0	SMOD1=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 16–4. Timer 1 Generated Commonly Used Baud Rates @ F_{SYSCLK}=22.1184MHz

Baud Rate	**TH1**, the Reload Value					
	T1X12=0			**T1X12=1**		
	SMOD1=0	SMOD1=1	Error	SMOD1=0	SMOD1=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 16–5. 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 16–6. 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%

16.8. Serial Port Mode 4 (SPI Master)

The Serial Port of **MG86FE/L508** is embedded an additional Mode 4 to support SPI master engine. The Mode 4 is selected by SM3, SM0 and SM1. Table 16–7 shows the serial port mode definition in **MG86FE/L508**.

Table 16–7. Serial Port Mode Selection

SM3	SM0	SM1	Mode	Description	Baud Rate
0	0	0	0	shift register	SYSCLK/12 or SYSCLK/4
0	0	1	1	8-bit UART	variable
0	1	0	2	9-bit UART	SYSCLK/64, /32
0	1	1	3	9-bit UART	variable
1	0	0	4	**SPI Master**	SYSCLK/12 or SYSCLK/4
1	0	1	5	Reserved	Reserved
1	1	0	6	Reserved	Reserved
1	1	1	7	Reserved	Reserved

URM0X3 also controls the SPI transfer speed. If URM0X3 = 0, the SPI clock frequency is SYSCLK/12. If URM0X3 = 1, the SPI clock frequency is SYSCLK/4.

The SPI master in **MG86FE/L508** uses the TXD as SPICLK, RXD as MOSI, and S0MI as MISO. nSS is selected by MCU software on other port pin. Figure 16–13 shows the SPI connection. It also can support the configuration for multiple slaves communication in Figure 16–14.

Figure 16–13. Serial Port Mode 4, Single Master and Single Slave configuration

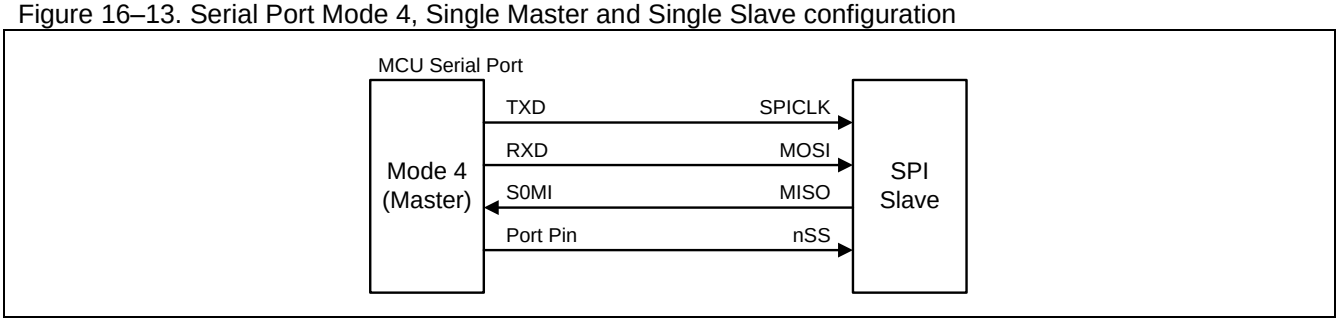
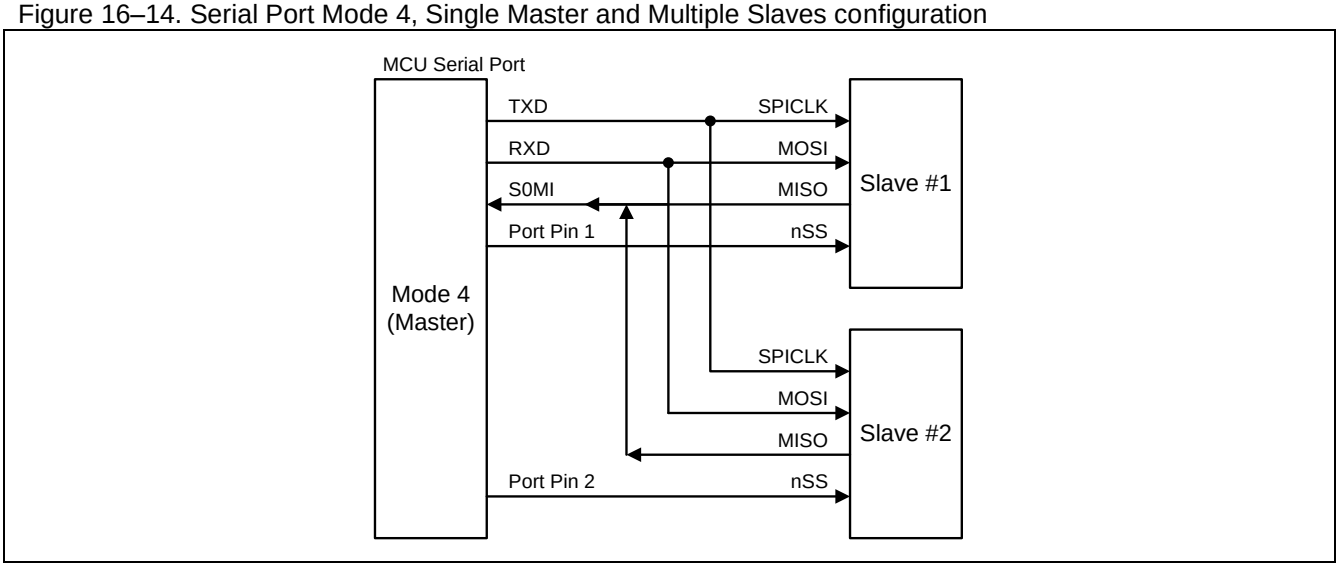


Figure 16–14. Serial Port Mode 4, Single Master and Multiple Slaves configuration



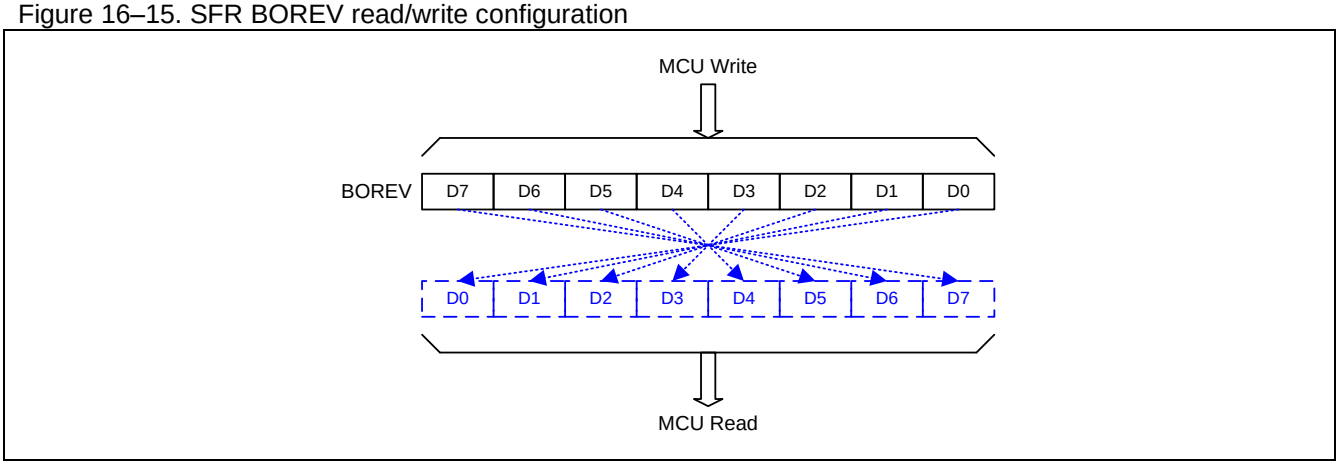
The SPI master satisfies the transfer with the full function SPI module of Megawin MG82/84 series MCU with CPOL, CPHA and DORD selection. For CPOL and CPHA condition, **MG86FE/L508** uses an easy way by initialize SPI clock (TXD/P3.1) polarity to fit them. Table 16–8 shows the serial port Mode 4 mapping with the four SPI operating mode.

Table 16–8. SPI mode mapping with Serial Port Mode 4 configuration

SPI Mode	CPOL	CPHA	Configuration in **MG86FE/L508**
0	0	0	Clear P3.1 to “0”
1	0	1	Clear P3.1 to “0”
2	1	0	Set P3.1 to “0”
3	1	1	Set P3.1 to “0”

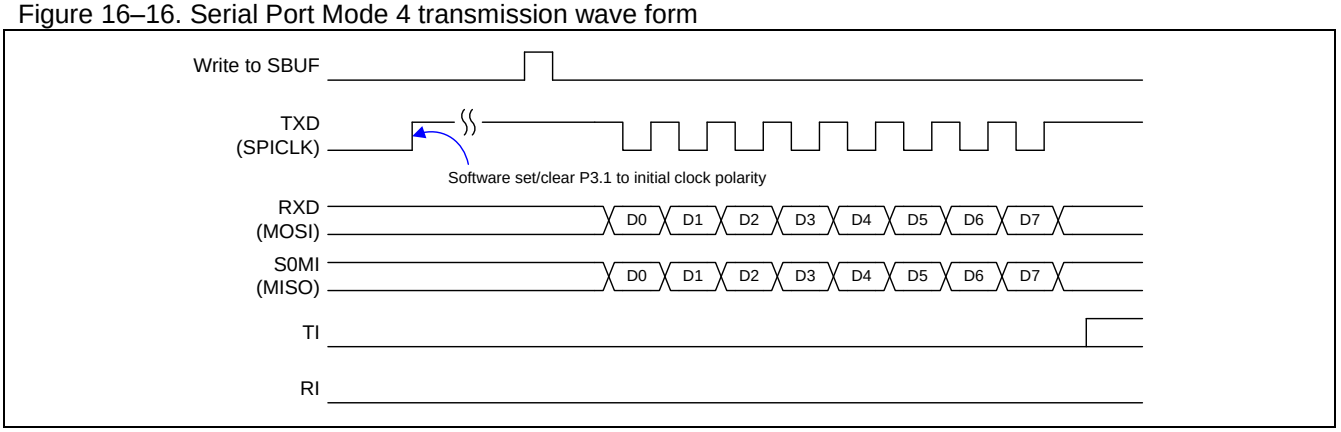
For bit order control (DORD) on SPI serial transfer, **MG86FE/L508** provides a SFR, BOREV, to reverse the bit order by software program. After MCU writing a MSB first data format to BOREV, MCU will get the LSB first data by reading BOREV back. The SPI master engine in serial port Mode 4 is the LSB first transferred which is same as serial port Mode 0. To support SPI MSB first shift, MCU must use the BOREV write/read operation to reverse the data bit order for SPI IN/OUT transmission. Figure 16–15 shows the BOREV configuration.

Figure 16–15. SFR BOREV read/write configuration



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 pin as MOSI serial data. The SPI shift clock is built on the TXD pin for SPICLK output. After eight raising edge of shift clocks passing, TI would be asserted by hardware to indicate the end of transmission. And the contents on the S0MI pin would be sampled and shifted into shift register. Then, “read SBUF” can get the SPI shift-in data. Figure 16–16 shows the transmission waveform in Mode 0. RI will not be asserted in Mode 4.

Figure 16–16. Serial Port Mode 4 transmission wave form



16.9. 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. Three registers, PCON, AUXR and AUXR2, are related to the baud rate setting:

SCON: Serial port Control Register

SFR Page = Normal

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: Serial port mode bit 0, (SMOD0 must = 0 to access bit SM0)

Bit 6: Serial port mode bit 1.

SM3	SM0	SM1	Mode	Description	Baud Rate
0	0	0	0	shift register	SYSCLK/12 or SYSCLK/4
0	0	1	1	8-bit UART	variable
0	1	0	2	9-bit UART	SYSCLK/64, /32
0	1	1	3	9-bit UART	variable
1	0	0	4	SPI Master	SYSCLK/12 or SYSCLK/4
1	0	1	5	Reserved	Reserved
1	1	0	6	Reserved	Reserved
1	1	1	7	Reserved	Reserved

Bit 5: 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 Page = Normal

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 Page = Normal

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 Page = Normal

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.

PCON0: Power Control Register 0

SFR Page = Normal & Page P

SFR Address = 0x87 POR = 00X1-0000, RESET = 00X0-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: SMOD1, 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.

AUXR2: Auxiliary Register 2

SFR Page = Normal

SFR Address = 0xA3 RESET = 0000-0000

7	6	5	4	3	2	1	0
--	BTI	URM0X3	SM3	T1X12	T0X12	T1CKOE	T0CKOE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 6: BTI, Block TI in Serial Port Interrupt.
0: Retain the TI to be a source of Serial Port Interrupt.
1: Block TI to be a source of Serial Port Interrupt.

Bit 5: URM0X3, Serial Port mode 0 and mode 4 baud rate selector.
0: Clear to select SYSCLK/12 as the baud rate for UART Mode 0 and Mode 4.
1: Set to select SYSCLK/4 as the baud rate for UART Mode 0 and Mode 4.

Bit 4: SM3, Serial Port Mode control bit 3.
0: Disable Serial Prot Mode 4.
1: Enable SM3 to control Serial Port Mode 4, SPI Master.

Bit 3: 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.

SFIE: System Flag Interrupt Enable Register

SFR Page = Normal
SFR Address = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	RTCFIE	KBIFIE	--	BOF0IE	WDTFIE
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: UTIE, UART TI Enabled in system flag interrupt.
0: Disable the interrupt vector sharing for TI in system flag interrupt.
1: Set TI flag will share the interrupt vector with system flag interrupt.

BOREV: Bit Order Reversed Register

SFR Page = Normal
SFR Address = 0x96 RESET = 0000-0000

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

This register serves data read as a Bit-Order **Reversed** function with data written into.

16.10. Serial Port Sample Code

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

Assembly Code Example:

```
      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

      MOV      IP0L,#PSL                    ; Select S0 interrupt priority
      MOV      IP0H,#PSH                    ;

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

      ORL      PCON0,#IDL;                  ; Set MCU into IDLE mode
```

C Code Example:

```
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

      IP0L = PSL;                           // Select S0 interrupt priority
      IP0H = PSH;                           //

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

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

(2). Required Function: S0 Mode 4 for SPI Master Mode0/1, SYSCLK = 24MHz

Assembly Code Example:

```
;     SETB      nSS                          ; user defined on GPIO
      CLR       P31                          ; for SPI CPOL=0, SPICLK is Logic Low in initial state
                                             ; CPHA = 0 or 1

; initial
      MOV       P3M1,#0x03                   ; Set P3.1 (SPICLK) to push-pull output mode
      MOV       P3M0,#0x00                   ; Set P3.0 (MOSI) to push-pull output mode
                                             ; P37 (MISO) is quasi mode in default

      ANL       AUXR2,#~URM0X3               ; SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
;     ORL       AUXR2,#URM0X3                ; or SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

      ORL       AUXR2,#SM3                   ; Set S0 to Mode 4 for SPI master operation
      MOV       SCON,#0x00

;   Transtor Data
;     CLR       nSS                          ; user defined on GPIO
      MOV       SBUF,#0x55                   ; Write 1st SPI data
      JNB       TI,$                         ; SPI master finished transfer
      MOV       data_reg,SBUF                ; Read 1st SPI input data
      CLR       TI

      MOV       SBUF,#0xAA;                  ; Write 2nd SPI data
      JNB       TI,$                         ; SPI master finished transfer
      MOV       data_reg,SBUF                ; Read 2nd SPI input data
      CLR       TI
;     SETB      nSS                          ; user defined on GPIO
```

C Code Example:

```
      unsigned char reg;
//    nSS=High;                              // user defined on GPIO
      P31 = 0;                               // for SPI CPOL=0, SPICLK is Logic Low in initial state
                                             // CPHA = 0 or 1

      P3M1 = 0x03;                           // Set P3.1 (SPICLK) to push-pull output mode
      P3M0 = 0x00;                           // Set P3.0 (MOSI) to push-pull output mode
                                             // P37 (MISO) is quasi mode in default

      AUXR2 &= ~URM0X3;                      // SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
//    AUXR2 |= URM0X3;                       // or SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

      AUXR2 |= SM3;                          // Set S0 to Mode 4 for SPI master operation
      SCON = 0x00;

//      Transtor Data
//      nSS = Low;                           // user defined on GPIO
      SBUF = 0x55;                           // Write 1st SPI data
      while(!TI);                            // SPI master finished transfer
      reg = SBUF;                            // Read 1st SPI input data
      TI = 0;

      SBUF = 0xAA;                           // Write 2nd SPI data
      while(!TI);                            // SPI master finished transfer
      reg = SBUF;                            // Read 2nd SPI input data
      TI = 0;

      //nSS = High;                          // user defined on GPIO
```

(3). Required Function: S0 Mode 4 for SPI Master Mode2/3, SYSCLK = 24MHz, ***MSB first***

Assembly Code Example:

```
;    SETB    nSS                    ; user defined on GPIO
     SETB    P31                    ; for SPI CPOL=1, SPICLK is Logic High in initial state
                                    ; CPHA = 0 or 1

; initial
     MOV     P3M1,#0x03             ; Set P3.1 (SPICLK) to push-pull output mode
     MOV     P3M0,#0x00             ; Set P3.0 (MOSI) to push-pull output mode
                                    ; P37 (MISO) is quasi mode in default

     ANL     AUXR2,#~URM0X3         ; SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
;    ORL     AUXR2,#URM0X3          ; or SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

     ORL     AUXR2,#SM3             ; Set S0 to Mode 4 for SPI master operation
     MOV     SCON,#0x00

;  Transtor Data
;    CLR     nSS                    ; user defined on GPIO
     MOV     BOREV,#0x5A            ; Reverse data out bit order to MSB first
     MOV     SBUF,BOREV             ; Write 1st SPI data

     JNB     TI,$                   ; SPI master finished transfer
     MOV     BOREV,SBUF             ; Read 1st SPI input data
     MOV     data_reg,BOREV         ; Reverse data in bit order to LSB first
     CLR     TI

     MOV     BOREV,#0xA5;           ; Reverse data out bit order to MSB first
     MOV     SBUF,BOREV;            ; Write 2nd SPI data

     JNB     TI,$                   ; SPI master finished transfer
     MOV     BOREV,SBUF             ; Read 2nd SPI input data
     MOV     data_reg,BOREV         ; Reverse data in bit order to LSB first
     CLR     TI
;    SETB    nSS                    ; user defined on GPIO
```

C Code Example:

```
     unsigned char reg;
//   nSS=High;                      // user defined on GPIO
     P31 = 1;                       // for SPI CPOL = 1, SPICLK is Logic High in initial state
                                    // CPHA = 0 or 1

     P3M1 = 0x03;                   // Set P3.1 (SPICLK) to push-pull output mode
     P3M0 = 0x00;                   // Set P3.0 (MOSI) to push-pull output mode
                                    // P37 (MISO) is quasi mode in default

     AUXR2 &= ~URM0X3;              // SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
//   AUXR2 |= URM0X3;               // or SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

     AUXR2 |= SM3;                  // Set S0 to Mode 4 for SPI master operation
     SCON = 0x00;

//     Transtor Data
//     nSS = Low;                   // user defined on GPIO
     BOREV = 0x5A;                  // Reverse data out bit order to MSB first
     _nop_ ();                      // Prevent C-Compiler optimization
     SBUF = BOREV;                  // Write 1st SPI data

     while(!TI);                    // SPI master finished transfer
     BOREV = SBUF;                  // Read 1st SPI input data
     _nop_ ();                      // Prevent C-Compiler optimization
     reg = BOREV;                   // Reverse data in bit order to LSB first
     TI = 0;

     BOREV = 0xA5;                  // Reverse data out bit order to MSB first
```

```
_nop_ ();                          // Prevent C-Compiler optimization
SBUF = BOREV;                      // Write 2nd SPI data
while(!TI);                        // SPI master finished transfer
BOREV = SBUF;                      // Read 2nd SPI input data
_nop_ ();                          // Prevent C-Compiler optimization
reg = BOREV;                       // Reverse data in bit order to LSB first
TI = 0;
//nSS = High;                      // user defined on GPIO
```

17. Programmable Counter Array (PCA)

The **MG86FE/L508** 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.

17.1. PCA Overview

The PCA consists of a dedicated timer/counter which serves as the time base for an array of four compare/capture modules. Figure 17–1 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.

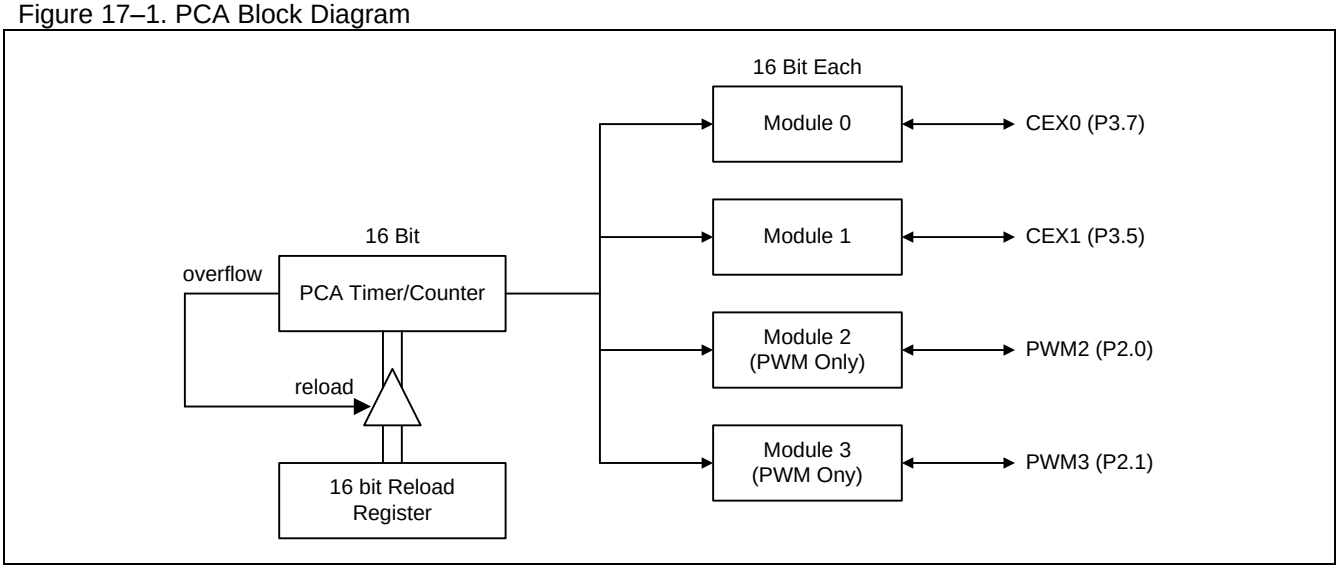
Module 0 and module 1 can be programmed in any one of the following modes:

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

Module 2 and module 3 only support PWM output mode.

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 17–1. PCA Block Diagram



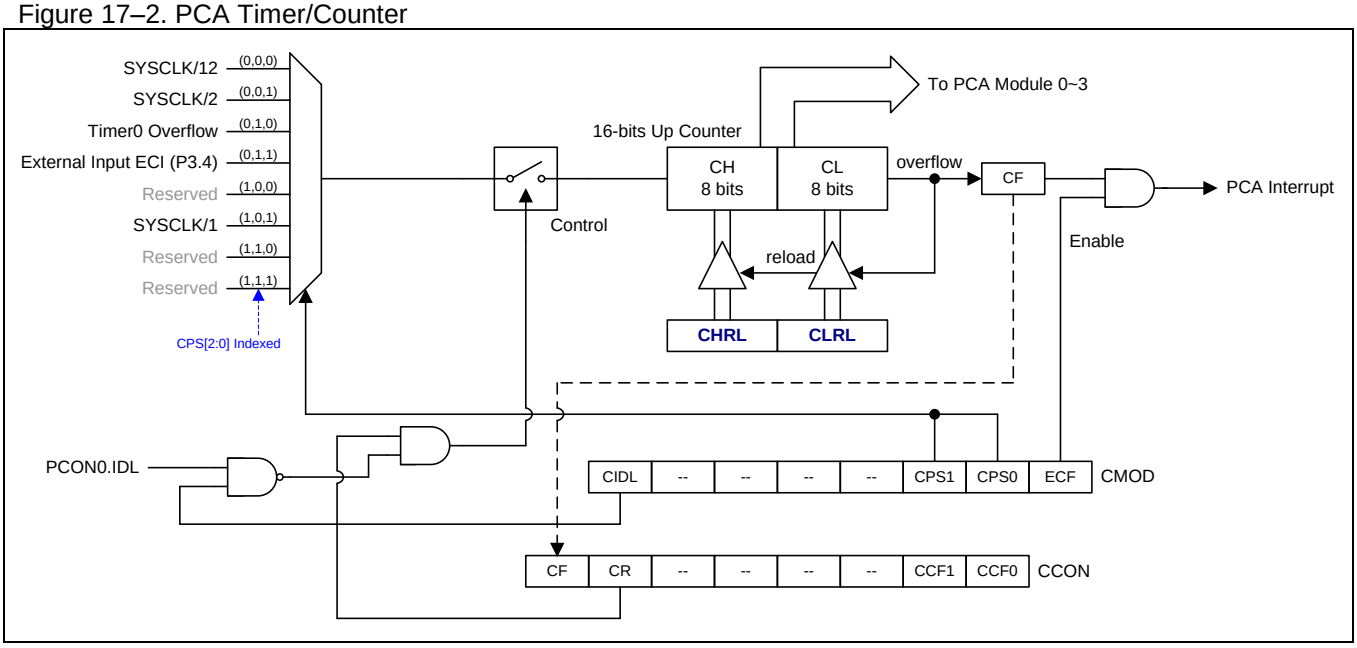
17.2. PCA Timer/Counter

The timer/counter for the PCA is an auto-reload 16-bit timer consisting of registers CH, CL (the high and low bytes of the count values), CHRL, CLRL (the high and low bytes reload registers), as shown in Figure 17–2. CHRL and CLRL are reload to CH and CL at each time overflow on CH+CL counter which can change the PCA cycle time for variable PWM resolution, such as 7-bit PWM.

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),
- directly from the system clock frequency.

Special Function Register CMOD contains the Count Pulse Select bits (CPS2, 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 17–2. PCA Timer/Counter



CMOD: PCA Counter Mode Register

SFR Page = Normal
SFR Address = 0xD9 RESET = 0xxx-0000

7	6	5	4	3	2	1	0
CIDL	--	--	--	**CPS2**	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 6~4: Reserved. Software must write “0” on these bits when CMOD is written.

Bit 3~1: CPS2-CPS0, PCA counter clock source select bits. **CPS2 is serviced on write-only function.**

CPS2	CPS1	CPS0	PCA Clock Source
0	0	0	Internal clock, SYSCLK/12
0	0	1	Internal clock, SYSCLK/2
0	1	0	Timer 0 overflow
0	1	1	External clock at the ECI pin
1	0	0	Reserved
1	0	1	Internal clock, SYSCLK/1
1	1	0	Reserved
1	1	1	Reserved

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 17–3.

CCON: PCA Counter Control Register

SFR Page = Normal
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.

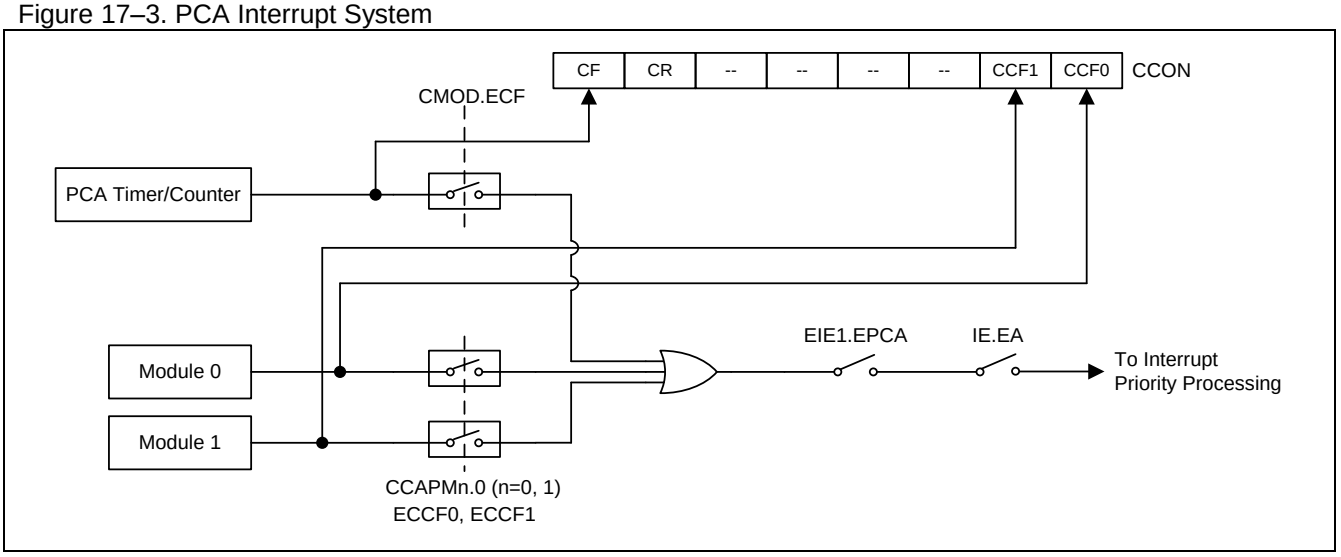
Bit 6: CR, PCA Counter Run control bit.
0: Must be cleared by software to turn the PCA counter off.
1: Set by software to turn the PCA counter on.

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 17–3. PCA Interrupt System



17.3. Compare/Capture Modules

Each of the compare/capture module 0 and 1 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 or 1

SFR Page = Normal

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=0~3

SFR Page = Normal
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=0~3

SFR Page = Normal
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 Page = Normal
SFR Address = 0xF2~0xF3 RESET = xxxx-x000

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

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

Bit 2: Invert PWM output on CEXn.
0: Non-inverted PWM output.
1: Inverted PWM output.

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.

PCAPWMn: PWM Mode Auxiliary Register, n=2, 3

SFR Page = Normal

SFR Address = 0xF4~0xF5 RESET = 0xxx-x000

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

Bit 7: PWMn, PWM control.
0: Disable the PWM mode in PCA module 2 and module 3.
1: Enable the PWM function and cause PWM2 or PWM3 pin to be used as a pulse width modulated output.

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

Bit 2: Invert PWM output on PWMn.
0: Non-inverted PWM output.
1: Inverted PWM output.

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.

17.4. Operation Modes of the PCA

Table 17–1 shows the CCAPMn register settings for the various PCA functions.

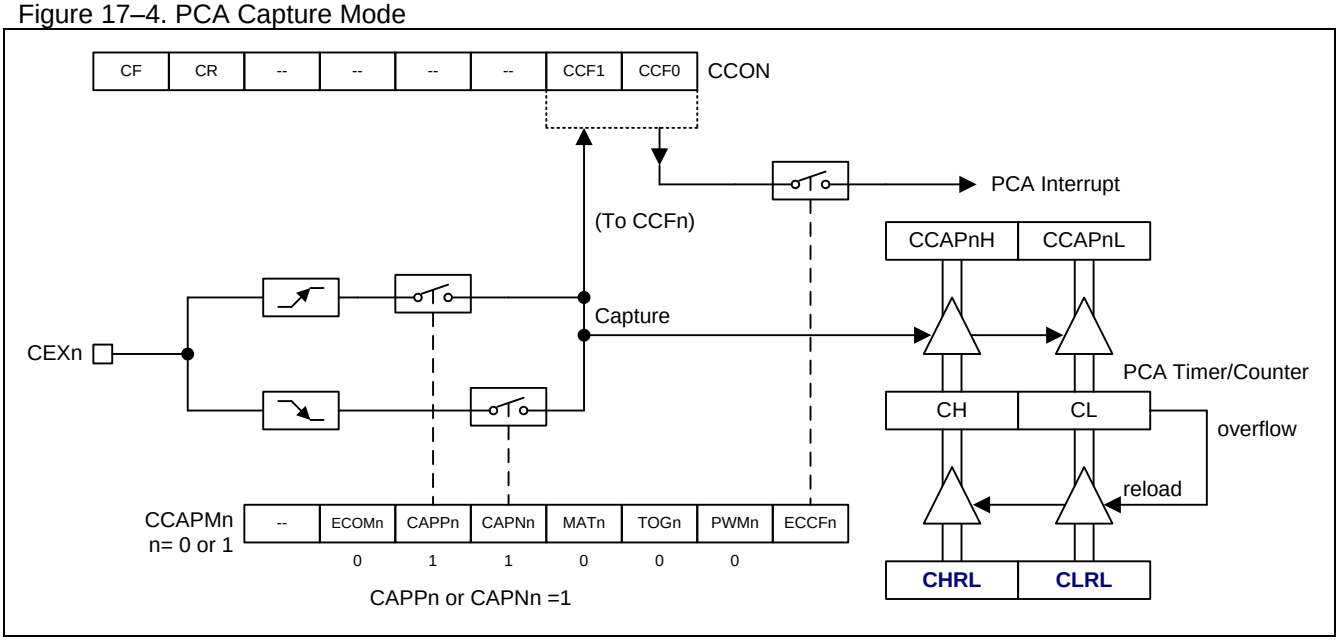
Table 17–1. PCA Module Modes

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)

17.4.1. 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. Only module 0 and module 1 support capture mode.

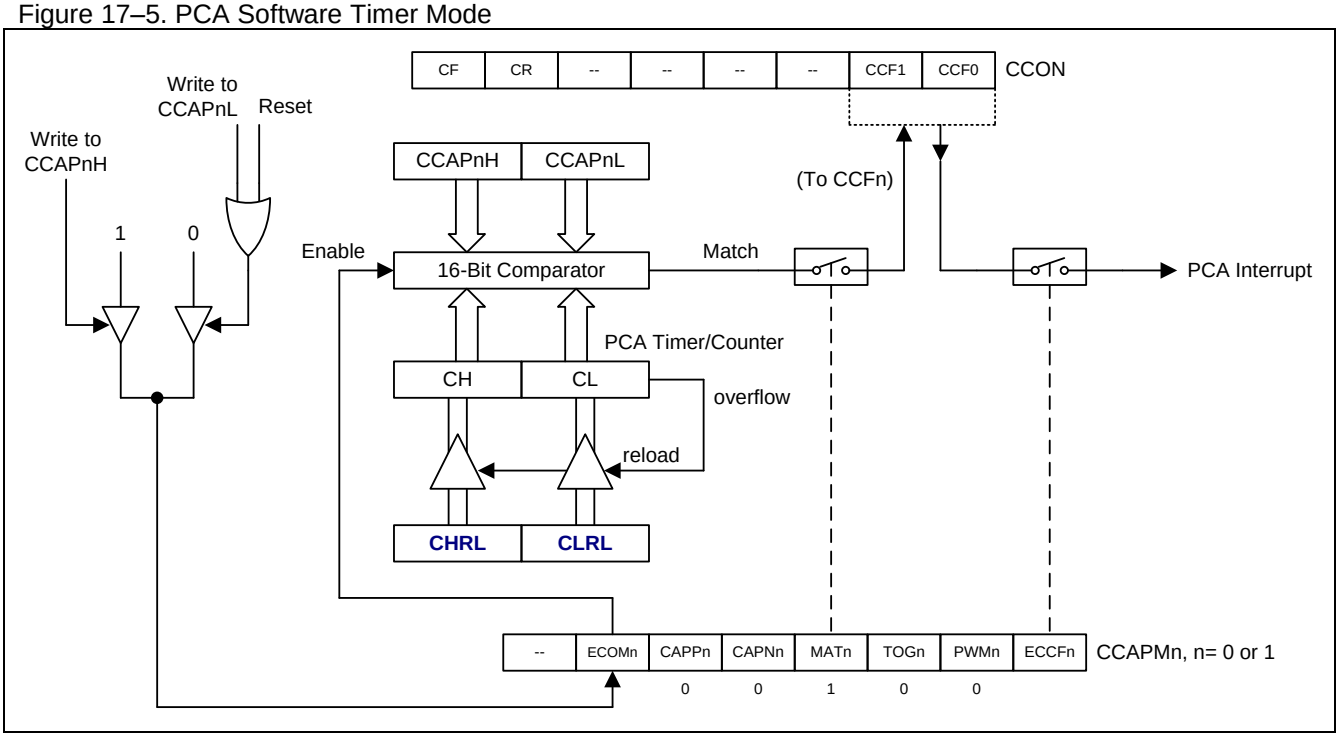
Figure 17–4. PCA Capture Mode



17.4.2. 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. Only module 0 and module 1 support software timer mode.

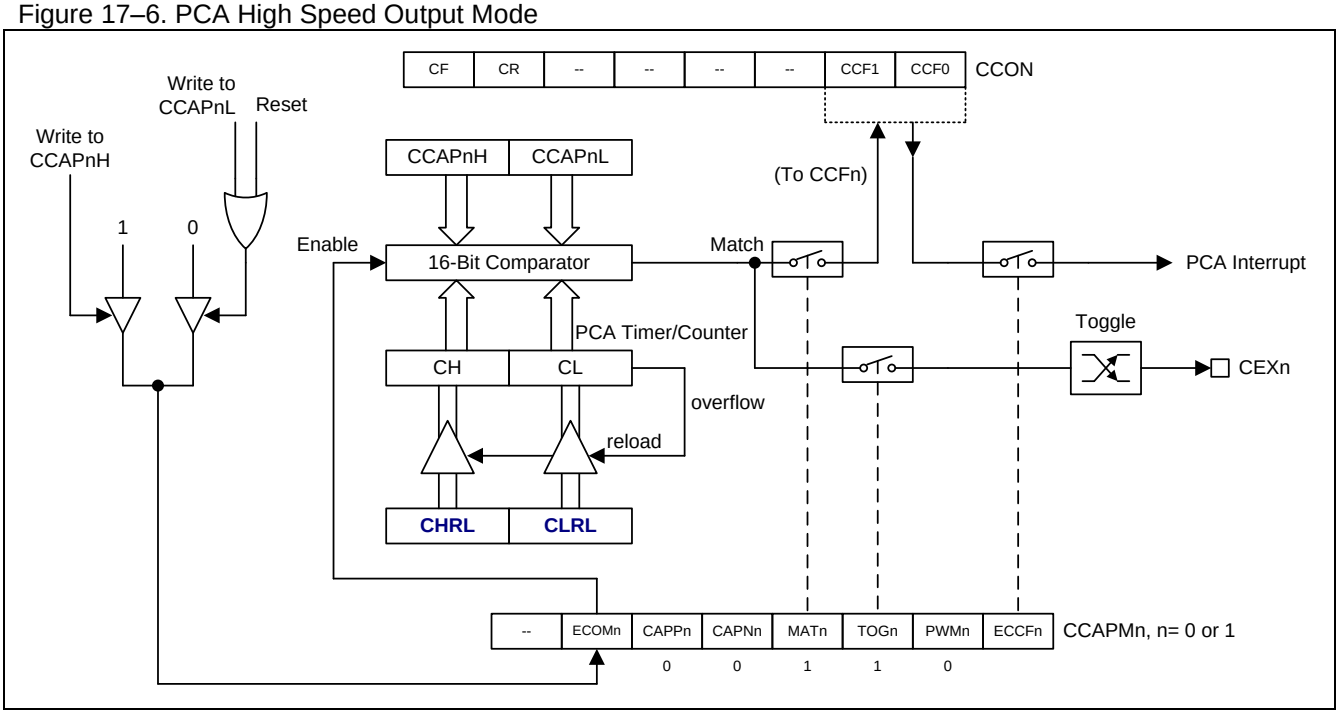
Figure 17–5. PCA Software Timer Mode



17.4.3. 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 17–6. PCA High Speed Output Mode



17.4.4. 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. **CLRL** stores the CL reload value on each time CL overflow which can modify the PWM frequency and resolution.

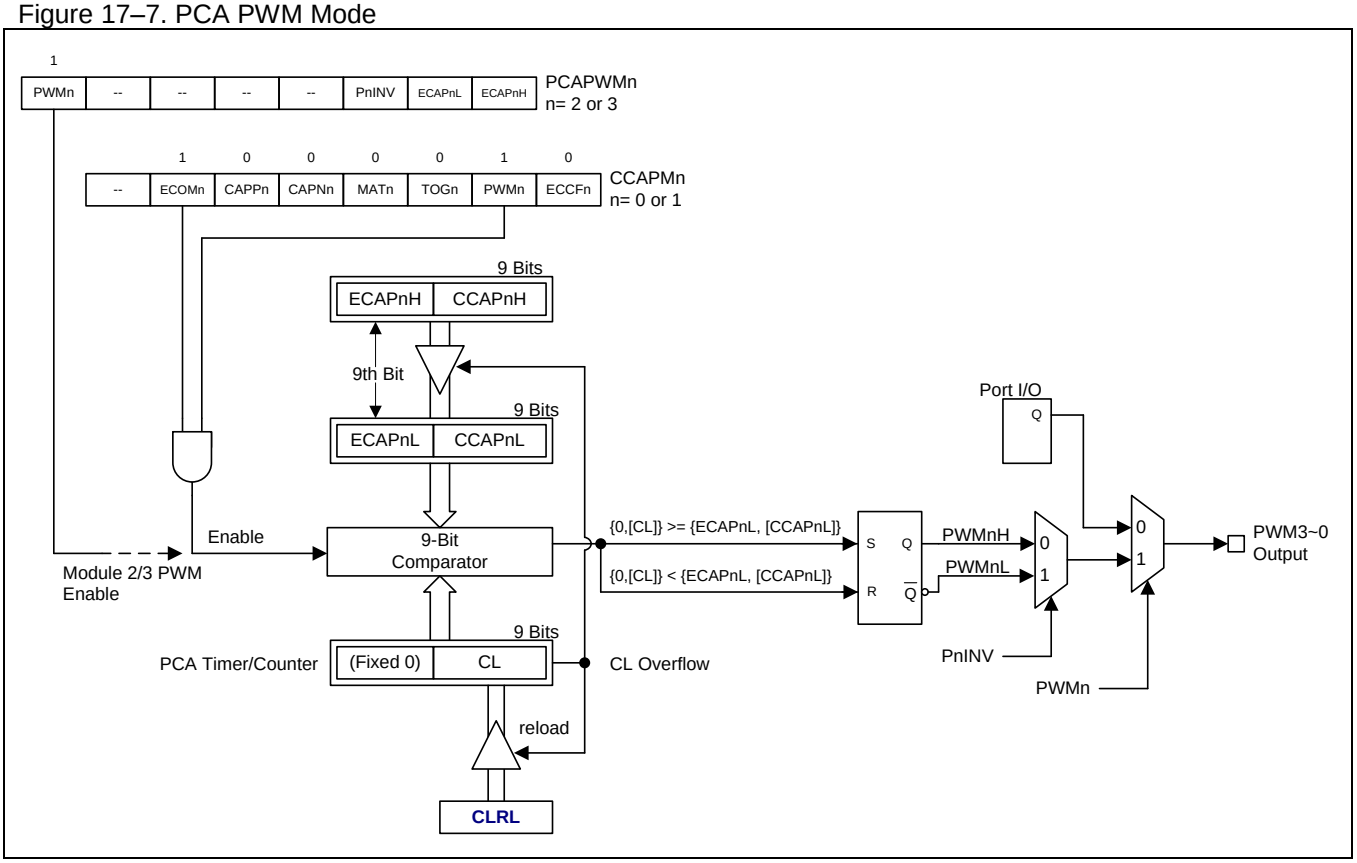
Using the 9-bit comparison, the duty cycle of the output can be improved to really start from 0%, and up to 100%. The formula for the duty cycle is:

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 17–7. PCA PWM Mode



PAOE: PWM Additional Output Enable Register

SFR Page = Normal

SFR Address = 0xF1 RESET = 0001-1001

7	6	5	4	3	2	1	0
P16OP0	P33OP3	P32OP3	P24OP3	P37OP0	P21OP2	P17OP2	P20OP2
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P16OP0, P16 Output PWM0.
0: Disable P16 to output PWM0. Default is disabled.
1: Enable P16 to output PWM0.

Bit 6: P33OP3, P33 Output PWM3.
0: Disable P33 to output PWM3. Default is disabled.
1: Enable P33 to output PWM3.

Bit 5: P32OP3, P32 Output PWM3.
0: Disable P32 to output PWM3. Default is disabled.
1: Enable P32 to output PWM3.

Bit 4: P24OP3, P24 Output PWM3.
0: Disable P24 to output PWM3.
1: Enable P24 to output PWM3. **Default is enabled.**

Bit 3: P37OP0, P37 or P26 Output PWM0. CEX0 in P37 or P26 is selected by P2PCA (AUXR1.5).
0: Disable P37 or P26 to output PWM0.
1: Enable P37 or P26 to output PWM0. **Default is enabled.**

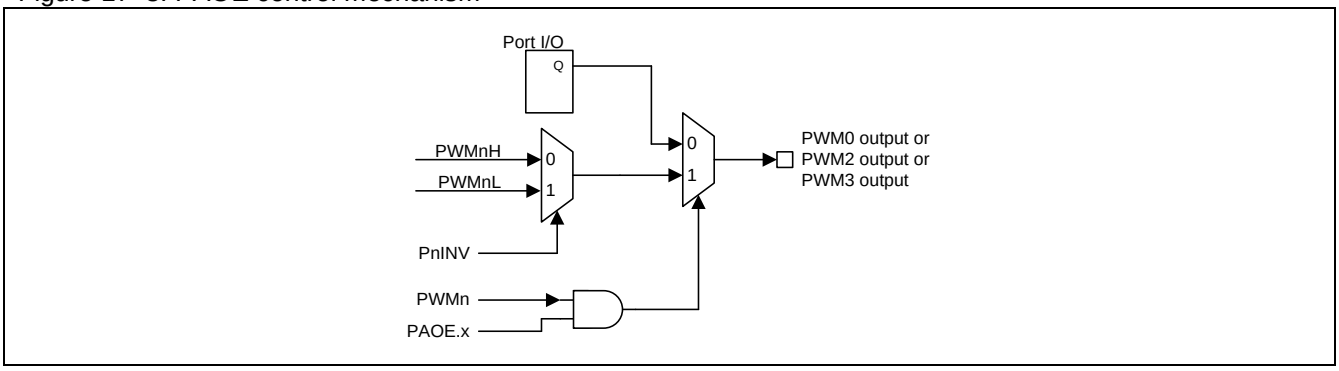
Bit 2: P21OP2, P21 Output PWM2.
0: Disable P21 to output PWM2. Default is disabled.
1: Enable P21 to output PWM2**.**

Bit 1: P17OP2, P17 Output PWM2.
0: Disable P17 to output PWM2. Default is disabled.
1: Enable P17 to output PWM2.

Bit 0: P20OP2, P20 Output PWM2.
0: Disable P20 to output PWM2.
1: Enable P20 to output PWM2. **Default is enabled.**

PAOE provides the multiple outputs for one PWM channel. PWM2 can be configured to 3 outputs on P2.0, P1.7 and P2.1 concurrently. PWM3 can output on P2.4, P3.2 and P3.3 concurrently. PWM0 can output on P3.7 and P1.6 concurrently. PWM1 has no additional output channel in **MG86FE/L508**. Figure 17–8 shows the PWM output control mechanism.

Figure 17–8. PAOE control mechanism



17.5. PCA Sample Code

(1). Required Function: Set PWM2/PWM3 output with 25% & 75% duty cycle

Assembly Code Example:

```
PWM2_PWM3:
        MOV     CCON,#00H                       ; stop CR
        MOV     CMOD,#02H                       ; PCA clock source = system clock / 2
;;;;;;;;; MOV   CMOD,#08H                       ; PCA clock source = system clock / 1

        MOV     CH,#00H                         ; initial state
        MOV     CL,#00H
        MOV     CHRL,#00H                       ; initial reload
        MOV     CLRL,#00H
        ;
        MOV     PCAPWM2,#PWM2                   ; enable PCA module 2 (PWM mode)
        MOV     CCAP2H,#0C0H                    ; 25%
        MOV     CCAP2L,#0C0H

        MOV     PCAPWM3,#PWM3                   ; enable PCA module 3 (PWM mode)
        MOV     CCAP3H,#40H                     ; 75%
        MOV     CCAP3L,#40H
        ;
        MOV     P2M0,#00010001B                 ; enable P2.0 & P2.4 pull-up
        SETB    CR                              ; start PCA
```

C Code Example:

```
void main(void)
{
        // set PCA
        CCON = 0x00;                            // disable PCA & clear CCF0, CCF1, CF flag
        CMOD = 0x02;                            // PCA clock source = system clock / 2
//      CMOD = 0x08;                            // PCA clock source = system clock / 1

        CL = 0x00; CH = 0x00;
        CHRL = 0x00; CLRL = 0x00;               // PCA counter range
        //---------------------------------------------
        PCAPWM2 = PWM2;                         // module 2 (Non-inverted)
        CCAP2H = 0xC0; CCAP2L = 0xC0;           // 25%

        PCAPWM3 = PWM3;                         // module 3
        CCAP3H = 0x40; CCAP3L = 0x40;           // 75 %
        //---------------------------------------------
        P2M0 = 0x11;
        CR = 1;                                 // start PCA's PWM output

        while (1);
}
```

(2). Required Function: Set PCA with 6-bit PWM (but the frequency will have 4X speed)

Assembly Code Example:

```
PWM_6BIT:
        ; CEX0 = P3.7 (Quasi-bidirection)
        MOV         CCON,#00H
        MOV         CMOD,#02H

        MOV         CH,#0FFH                    ; 6 bit mode (65536-64)
        MOV         CL,#0C0H
        MOV         CHRL,#0FFH                  ; 6 bit mode (65536-64)
        MOV         CLRL,#0C0H
        ;
        MOV         CCAPM0,#ECOM0+PWM0
        MOV         PCAPWM0,#00H
        MOV         CCAP0H,#0D0H                ; (duty cycle) 48/64 = 75 %
        MOV         CCAP0L,#0D0H                ; 256 - 48 = 204 (0D0H)
      ;

        SETB        CR                          ; start CR
        RET
```

C Code Example:

```
void main(void)
{
        // CEX0 = P3.7 (Quasi-bidirection)
        CCON = 0x00;                            // disable PCA & clear CCF0, CCF1, CF flag
        CMOD = 0x02;                            // PCA clock source = system clock / 2
        CCAPM0 = ECOM0+PWM0;                  // module 0 = PWM mode
        PCAPWM0 = 0x00;
        //-----------------------------------------------
        CHRL = CH = (65536-64) >> 8;            // 6bit
        CLRL = CL = (65536-64);

        CCAP0L = 240;                           // (duty cycle) 16/64 = 25 %
        CCAP0H = 240;                           // 256 - 16 = 240
        //-----------------------------------------------
        CR    = 1;

        while (1);
}
```

18. Keypad Interrupt (KBI)

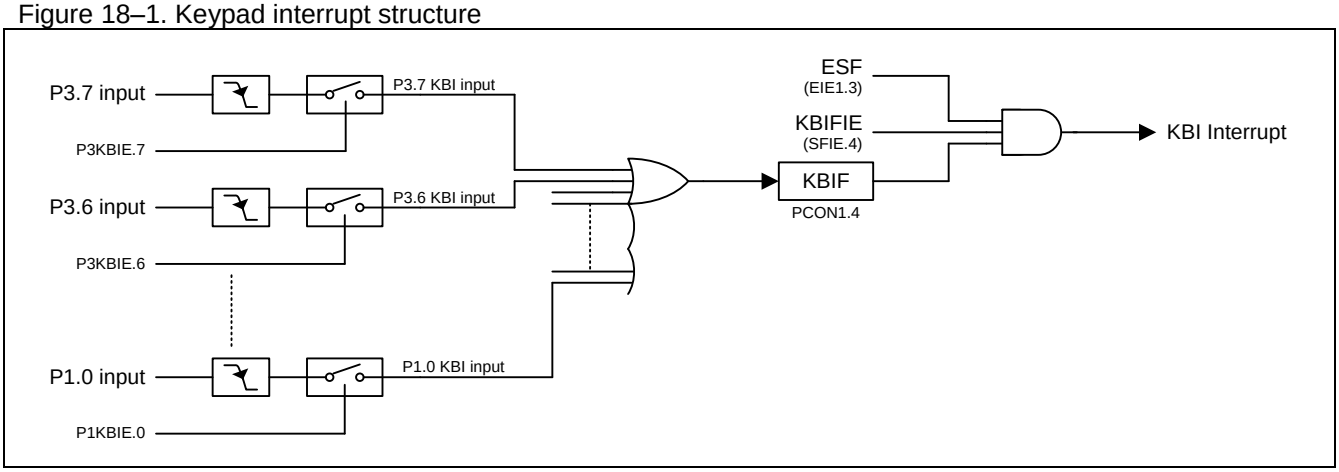
The Keypad Interrupt function is intended primarily to allow a single interrupt to be generated when enabled port pin is a falling edge occurred. This function can be used keypad recognition. Figure 18–1 shows the structure of the keypad interrupt function.

There are two SFRs used for this function. Each bit in P1KBIE and P3KBIE controls the associated port pin to enable or disable the KBI function at falling edge occurred. Any recognized KBI event will cause the hardware to set the interrupt flag KBIF and generate an interrupt if it has been enabled. Not necessary to enable the KBI interrupt, enabled KBI port pin can wakeup CPU from idle mode (falling edge) and power-down mode (low level).

An enabled KBI port pin must be configured to input mode by software programming. For Port 3 on KBI function, the selected port pin can be configured to input-only mode, quasi-mode with output high, or open-drain mode with output high. For Port 1 and Port 4 on KBI application, the selected port pin must be configured to open-drain mode with output high. Any output low on KBI port pin will trigger the KBI event by MCU itself. The on-chip pull-up resistor on Port 1 and Port 4 can be enabled by software to provide a weak input high state. Port 3 only has an on-chip pull-up resistor in quasi-mode.

18.1. Keypad Interrupt Structure

Figure 18–1. Keypad interrupt structure



18.2. Keypad Interrupt Register

P1KBIE: Port 1 KBI Enable Control Register

SFR Page = Normal
SFR Address = 0xD7 RESET = 0000-0000

7	6	5	4	3	2	1	0
P17KBIE	P16KBIE	P15KBIE	P14KBIE	P13KBIE	P12KBIE	P11KBIE	P10KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: Keypad Input function enable bit for each Port 1 pins.
0: Disable associated port pin for keypad input function.
1: Enable associated port pin for keypad input function.

P3KBIE: Port 3 KBI Enable Control Register

SFR Page = Normal
SFR Address = 0xD6 RESET = 0000-0000

7	6	5	4	3	2	1	0
P37KBIE	P36KBIE	P35KBIE	P34KBIE	**P41KBIE**	**P40KBIE**	P31KBIE	P30KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: Keypad Input function enable bit for P3.7 ~ P3.4, P4.1, P4.0, P3.1 and P3.0 pins.
0: Disable associated port pin for keypad input function.
1: Enable associated port pin for keypad input function.

PCON1: Power Control Register 1

SFR Page = Normal & Page P
SFR Address = 0x97 POR = 00x0-0X00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	**KBIF**	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 3: KBIF, Keypad Interrupt Flag.
0: This bit must be cleared by software writing **"1"** on it. Software writing ":0" is no operation.
1: This bit is only set by falling edge on enabled KBI port pin. Writing **"1"** on this bit will clear KBIF. In power down mode, this bit is set by low level on enabled KBI port pin.

18.3. Keypad Interrupt Sample Code

(1). Required Function: Implement a KBI function on P1.3~P1.0

Assembly Code Example:

```
      ORG       0003Bh
SystemFlag_ISR:                                 ;
      PUSH      ACC                             ;

      MOV       A,PCON1                         ;
      JNB       ACC.3,Not_KBIF_ISR              ;
      To do……
      ANL       PCON1,#KBIF                     ; Clear KBI flag (write “1”)

Not_KBIF_ISR:
      POP       ACC                             ;
      RETI                                      ;

main:
      ORL       PUCON0,#PU10                    ; Enable P1.3~P1.0 on-chip pull-up resistor
      ANL       P1M0,#0F0h                      ; Set P1.3~P1.0 to Open-Drain Output
      ORL       P1,#00Fh                        ; Set P1.3~P1.0 to Input mode

      ORL       EIP1L,#PSFL                     ; Select SystemFlag interrupt priority
      ORL       EIP1H,#PSFH                     ;

      MOV       P1KBIE,#00Fh                    ; Enable P1.3~P1.0 KBI function
;     MOV       P3KBIE,#0xF3                    ; Enable P3 KBI function
;     MOV       P3KBIE,#0x0C                    ; Enable P4.1~0 KBI function

      ANL       PCON1,#KBIF                     ; Clear KBI flag (write “1”)

      ORL       SFIE,#KBIFIE                    ; Enable KBI interrupt
      ORL       EIE1,#ESF                       ; Enable SystemFlag interrupt
      SETB      EA                              ; Enable global interrupt

      ORL       PCON0,#PD                       ; Set MCU into power-down mode
```

C Code Example:

```
void SystemFlag_ISR(void) interrupt 7
{     if(PCON1 & KBIF)
      {
            PCON1 &= KBIF;                      // Clear KBI Flag
      }
}

void main(void)
{
      PUCON0 |= PU10;                           // Enable P1.3~P1.0 on-chip pull-up resistor
      P1M0 &= 0xF0;                             // Set P1.3~P1.0 to Open-Drain Output
      P1 |= 0x0F;                               // Set P1.3~P1.0 to Input mode

      EIP1L |= PSFL;                            // Select SystemFlag interrupt priority
      EIP1H |= PSFH;

      P1KBIE = 0x0F;                            // Enable P1.3~P1.0 KBI function
//    P3KBIE = 0xF3;                            // Enable P3 KBI function
//    P3KBIE = 0x0C;                            // Enable P4.1~0 KBI function

      PCON1&= KBIF;                             // Clear KBI flag (write “1”)

      SFIE |= KBIFIE;                           // Enable KBI interrupt
      EIE1 |= ESF;                              // Enable SystemFlag interrupt
      EA = 1;                                   // Enable global interrupt

      PCON0 |= PD;                              // Set MCU into power-down mode
}
```

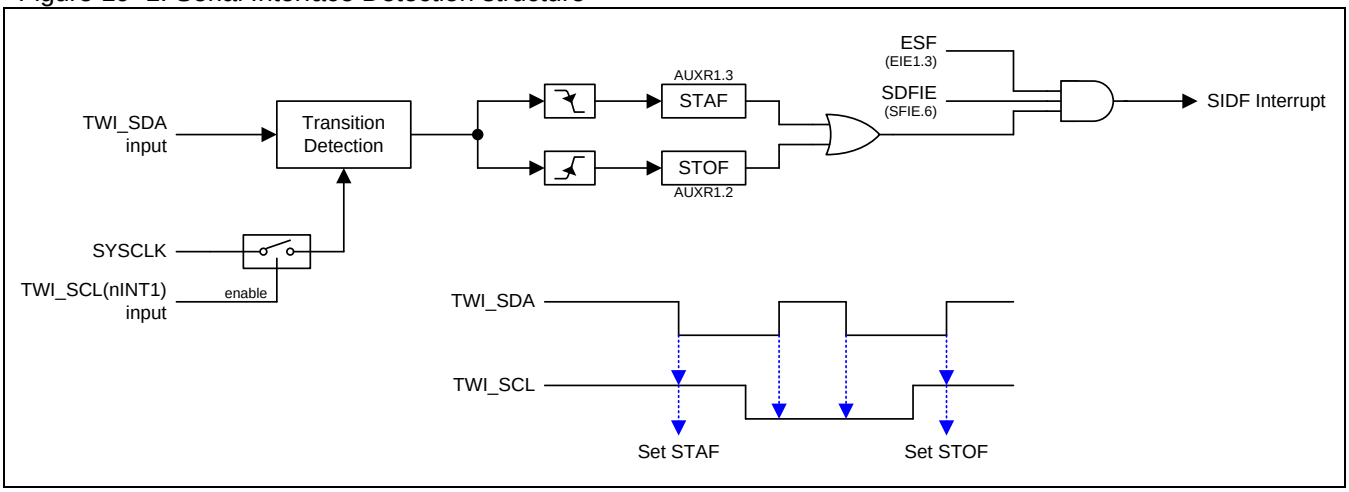
19. Serial Interface Detection

The serial interface detection module is always monitoring the “Start” and “Stop” condition on two-wire-interface. TW_SCL is the serial clock signal and TWI_SDA is the serial data signal. If any matched condition is detected, hardware set the flag on STAF and STOF. Software can poll these two flags or set SDFIE (SFIE.6) to share the interrupt vector on System Flag. And TWI_SCL is located on nINT1 which helps MCU to strobe the serial data by nINT1 interrupt. Software can use these resources to implement a variable TWI slave device.

19.1. Serial Interface Detection Structure

Figure 19–1 shows the configuration of STAF and STOF detection, interrupt architecture and event detecting waveform.

Figure 19–1. Serial Interface Detection structure



19.2. Serial Interface Detection Register

AUXR1: Auxiliary Control Register 1

SFR Page = Normal
SFR Address = 0xA2 RESET = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	**STAF**	**STOF**	BPOC1	BPOC0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: STAF, Start Flag detection of TWI.
0: Clear by firmware by writing “0” on it.
1: Set by hardware to indicate the START condition occurred on TWI bus.

Bit 2: STOF, Stop Flag detection of TWI.
0: Clear by firmware by writing “0” on it.
1: Set by hardware to indicate the START condition occurred on TWI bus.

SFIE: System Flag Interrupt Enable Register

SFR Page = Normal
SFR Address = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	RTCFIE	KBIFIE	--	BOF0IE	WDTFIE
R/W	**W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 6: SDIFIE, Serial Interface Detection Flag Interrupt Enabled.
0: Disable SDIF(STAF or STOF) interrupt.
1: Enable SDIF(STAF or STOF) interrupt.

19.3. SIDF Sample Code

There are two sample codes in the following diagram to implement TWI slave device. The first one is fully interrupt mode. It uses the STAF and STOF interrupt to detect the Start/Stop event and nINT interrupt to strobe serial data input. Whenn SYSCLK = 24MHz, the maximum speed of TWI slave is 200K bps. But the real speed must consider the other interrupt service duration in system application.

The second sample code is burst mode. Software only uses STAF and STOF for TWI event detection. Then, software polls the port pin state for TWISCL and TWI_SDA control. When SYSCLK = 24MHz, the general speed is 200K bps in this mode

(1). Required Function: TWI Slave on SYSCLK=24MHz in fully interrupt mode:

Assembly Code Example:

```
$INCLUDE (REG_MG86FE508.INC)

SLAVE_DEV_ADDR    EQU  20H              ; declare slave device address
DATA_LENGTH       EQU  32               ; declare buffer size

;-------------------------------------------------------------------------
; declare the TWSI state
;-------------------------------------------------------------------------
I2C_SlaveStandby  EQU  0x00
I2C_SLA_with_W    EQU  0x01
I2C_SLA_with_R    EQU  0x02
I2C_Disable       EQU  0x03
I2C_SL_W_ACK      EQU  0x04
I2C_SL_R_ACK      EQU  0x05
I2C_SL_R_NAK      EQU  0x06

;-------------------------------------------------------------------------
; declare the TWSI pin
;-------------------------------------------------------------------------
SDA               EQU  P3.4
SCL               EQU  P3.3

;-------------------------------------------------------------------------
; data area
;-------------------------------------------------------------------------
CONTROLDATA SEGMENT DATA
        RSEG CONTROLDATA
ReceiveString:  DS      DATA_LENGTH     ; data buffer
STACK:          DS      40              ; stack area size
position:       DS      1
tempByte:       DS      1

ADDR:           DS      1
IICByte:        DS      1
Stage:          DS      1

BITDATA SEGMENT BIT
        RSEG BITDATA
firstByte:      DBIT    1               ; the flag for receive SLA+R/W
completeAByte:  DBIT    1               ; set complete flag when transfer/receive one byte
Slave_RW:       DBIT    1               ; clear Slave_RW to beceive / set to transfer


;-------------------------------------------------------------------------
; code area
;-------------------------------------------------------------------------
    CSEG    AT   0000H                  ;start address = 0x0000
    JMP     ASSEMBLY_MAIN

    CSEG    AT   0013H                  ; EX0 interrupt ISR address
    JMP     SCL_DETECT_ISR
```

```
        CSEG    AT   003BH                  ; detect STAF or STOF ISR address
        JMP     SystemFlag_ISR

TWSI_CS SEGMENT CODE
        RSEG    TWSI_CS
        USING   0

ASSEMBLY_MAIN:
        MOV     SP,#STACK                  ; initial SP for stack size
        ANL     CKCON0,#11111000B          ; system clock / 1
        CALL    INITIAL_TWSI               ; initial TWSI

MAIN_LOOP:
        ; to do ...

        MOV     ACC,Stage
        XRL     A,#I2C_Disable
        JZ      MAIN_LOOP

        JNB     completeAByte,MAIN_LOOP    ; have a event ?

        ; ---------------------------------------------
        MOV     ACC,Stage
        CJNE    A,#I2C_SLA_with_W,SUBROUTINE_I2C_SLA_with_R

SUBROUTINE_I2C_SLA_with_W:
        MOV     R1,#ReceiveString          ; initial for receive
        CLR     completeAByte              ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SLA_with_R:
        CJNE    A,#I2C_SLA_with_R,SUBROUTINE_I2C_SL_W_ACK

        MOV     R1,#ReceiveString          ; initial for transfer
        MOV     A,@R1
        MOV     IICByte,A
        RLC     A                          ; it must transfer MSB to SDA
        MOV     SDA,C
        CLR     completeAByte              ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_W_ACK:
        CJNE    A,#I2C_SL_W_ACK,SUBROUTINE_I2C_SL_R_ACK
        MOV     @R1,IICByte                ; save data to "ReceiveString"

        INC     R1                         ; limit buffer index
        CJNE    R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV     R1,#ReceiveString

        CLR     completeAByte              ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_ACK:
        CJNE    A,#I2C_SL_R_ACK,SUBROUTINE_I2C_SL_R_NAK

        INC     R1                         ; limit buffer index
        CJNE    R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV     R1,#ReceiveString

        MOV     IICByte,@R1                ; prepare data form data buffer
        MOV     ACC,@R1
        RLC     A
        MOV     SDA,C                      ; SDA = MSB

        CLR     completeAByte              ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_NAK:
```

```
        CJNE    A,#I2C_SL_R_NAK,MAIN_LOOP

        SETB    SDA                             ; NAK-event
        CLR     completeAByte
        JMP     MAIN_LOOP


;-------------------------------------------------------------------------
; initial TWSI interrupt (priority) & trigger mode
;-------------------------------------------------------------------------
INITIAL_TWSI:
        ; System Flag have the highest priority
        ORL     EIP1H,#08H
        ORL     EIP1L,#08H

        ; the EX1 priority
        ORL     IP0H,#00000100B
        ANL     IP0L,#11111011B

        ; enable ETWSI
        ORL     EIE1,#ESF
        ORL     SFIE,#SDIFIE
        SETB    EA

        ; P33 & P34 is open drain mode for TWSI
        MOV     P3M0,#18H
        MOV     P3M1,#18H

        ; edge detect
        SETB    IT1
        ORL     AUXR0,#INT1H

        ; declare slave device address
        MOV     ADDR,#SLAVE_DEV_ADDR
        MOV     Stage,#I2C_Disable
        CLR     completeAByte

        RET

;-------------------------------------------------------------------------
; inital TWSI's SDA (STAF & STOF) edge detection
;-------------------------------------------------------------------------
SystemFlag_ISR:
        PUSH    ACC
        PUSH    PSW

        MOV     ACC, AUXR1                      ; check STAF or STOF ?
        JB      ACC.3,STAF_ROUTINE
        JB      ACC.2,STOF_ROUTINE

EXIT_FLAG_ISR:
        POP     PSW
        POP     ACC
        RETI

STAF_ROUTINE:                                   ; start of TWSI

; initial EX0 for raising edge detection and enable EX0 interrupt
        ORL     AUXR0, #INT1H
        NOP
        CLR     IE1
        SETB    SDA
        SETB    EX1

        MOV     position,#0FFH                  ; initial position for TWSI
        ANL     AUXR1,#~STAF                    ; clear STAF flag

        CLR     Slave_RW                        ; clear for receive a byte or address
        MOV     Stage,#I2C_SlaveStandby
```

```
        SETB    firstByte                       ; address byte flag
        JMP     EXIT_FLAG_ISR

STOF_ROUTINE:                                   ; stop of TSWI
        CLR     EX1                             ; disable EX0 interupt service routine
        ANL     AUXR1,#~STOF                    ; clear STOF flag
        MOV     Stage,#I2C_Disable
        JMP     EXIT_FLAG_ISR

;-------------------------------------------------------------------------
;   access SDA by EX1 interrupt
;-------------------------------------------------------------------------
SCL_DETECT_ISR:
        PUSH    ACC
        PUSH    PSW

        INC     position
        JB      Slave_RW,SLAVE_READ
    ;---------------------------------------------------------------------
SLAVE_WRITE:
        MOV     A,position
        CLR     C
        SUBB    A,#8
        JNC     SLAVE_WRITE_WAIT_FOR_ACK  ; is ACK signal ?

SLAVE_WRITE_8BITS:                              ; MSB~LSB (8 bits)
        MOV     ACC,tempByte                    ; 1. rotate tempByte
        MOV     C,SDA
        RLC     A                               ; 2. rotate SDA to tempByte.0
        MOV     tempByte,ACC
    ;
        MOV     A,position
        CJNE    A,#7,EXIT_SCL_DETECT_ISR
    ;
        ANL     AUXR0,#~INT1H                   ; the falling edge of ACK signal
        NOP
        CLR     IE1
        JMP     EXIT_SCL_DETECT_ISR

SLAVE_WRITE_WAIT_FOR_ACK:                       ; for 9 bit (ACK/NAK)
        JNZ     COMPLETE_WRITE_ONE_BYTE
        JNB     firstByte,SLAVE_WRITE_RESPONSE_ACK

        MOV     ACC,tempByte
        CLR     C
        RRC     A
        CJNE    A,ADDR,NOT_SLAVE_ADDR
SLAVE_WRITE_RESPONSE_ACK:
        CLR     SDA
        JMP     EXIT_SCL_DETECT_ISR
NOT_SLAVE_ADDR:
        CLR     EX1
        MOV     Stage,#I2C_Disable
        JMP     EXIT_SCL_DETECT_ISR

COMPLETE_WRITE_ONE_BYTE:                        ; 9th falling edge
        ORL     AUXR0,#INT1H                    ; the raising edge of SCL signal
        NOP
        CLR     IE1
        SETB    SDA
        SETB    completeAByte                   ; set '1' when it receives a byte
        MOV     position,#0FFH                  ; reset position

        JNB     firstByte,REPEAT_RECEIVE_MODE
        CLR     firstByte

        ; SLA+W or SLA+R ?
        CLR     Slave_RW
        MOV     ACC,tempByte
```

```
        JNB     ACC.0,SET_IN_SLAW_MODE
        SETB    Slave_RW

        ANL     AUXR0,#~INT1H               ; the falling edge of ACK signal
        NOP
        CLR     IE1

        MOV     Stage,#I2C_SLA_with_R
        JMP     EXIT_SCL_DETECT_ISR

SET_IN_SLAW_MODE:
        MOV     Stage,#I2C_SLA_with_W
        JMP     EXIT_SCL_DETECT_ISR

REPEAT_RECEIVE_MODE:
        MOV     IICByte,tempByte
        MOV     Stage,#I2C_SL_W_ACK
      ;-------------------------------------------------------------------------
EXIT_SCL_DETECT_ISR:
        POP     PSW
        POP     ACC
        RETI

      ;-------------------------------------------------------------------------
SLAVE_READ:
        MOV     A,position
        CLR     C
        SUBB    A,#7
        JNC     SLAVE_READ_WAIT_FOR_ACK     ; is ACK signal ?
SLAVE_READ_8BITS:
        MOV     A,IICByte                   ; rotate tempByte.7 to SDA
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C
        JMP     EXIT_SCL_DETECT_ISR

SLAVE_READ_WAIT_FOR_ACK:
        SETB    SDA
        JNZ     COMPLETE_READ_ONE_BYTE
        JMP     EXIT_SCL_DETECT_ISR

COMPLETE_READ_ONE_BYTE:
        SETB    completeAByte               ; set '1' when it receives a byte
        MOV     position,#0FFH              ; reset position
        JNB     SDA,SET_I2C_SLAVE_READ_ACK
        MOV     Stage,#I2C_SL_R_NAK
        JMP     EXIT_SCL_DETECT_ISR

SET_I2C_SLAVE_READ_ACK:
        MOV     Stage,#I2C_SL_R_ACK
        JMP     EXIT_SCL_DETECT_ISR

;-------------------------------------------------------------------------------
        END
```

C Code Example:

```
#include <REG_MG86FE508.H>
#include <intrins.h>

#define SLAVE_DEV_ADDR  0x20                // declare slave device   address
#define DATA_LENGTH     32                  // declare buffer size

//-------------------------------------------------------------------------
// declare I2C stage
```

```
//-------------------------------------------------------------------------
#define I2C_SlaveStandby    0x00
#define I2C_SLA_with_W      0x01
#define I2C_SLA_with_R      0x02
#define I2C_Disable         0x03
#define I2C_SL_W_ACK        0x04                    // SLA_W with data ACK
#define I2C_SL_R_ACK        0x05                    // SLA_R with data ACK
#define I2C_SL_R_NAK        0x06                    // SLA_R with data NAK

//-------------------------------------------------------------------------
// declare global variable
//-------------------------------------------------------------------------
typedef     struct {
        unsigned char ADDR;
        unsigned char IICByte;
        unsigned char Stage:8;
        unsigned char completeAByte:1;
        unsigned char Slave_RW:1;
} _TWSI;

_TWSI twsi;
unsigned char tempByte;
unsigned char position;
bit firstByte;

//-------------------------------------------------------------------------
// declare the TWSI       pin
//-------------------------------------------------------------------------
sbit SDA = P3^4;
sbit SCL = P3^3;

//-------------------------------------------------------------------------
// initial TWSI       interrupt (priority) & trigger mode
//-------------------------------------------------------------------------
void INITIAL_TWSI ()
{
        // System Flag have the highest priority
        EIP1H |= 0x08;
        EIP1L |= 0x08;

        // the EX1 have normal priority
        IP0H |= 0x08;
        IP0L &= ~0x08;

        EIE1 |= ESF;                                // enable ETWSI
        SFIE |= SDIFIE;
        EA = 1;

        // P33 & P34 is open drain mode for TWSI
        P3M0 = 0x18;
        P3M1 = 0x18;

        IT1 = 1;
        AUXR0 |= INT1H;

        // declare slave device address
        twsi.ADDR = SLAVE_DEV_ADDR;
        twsi.completeAByte = 0;
        twsi.Stage = I2C_Disable;
}

//-------------------------------------------------------------------------
// main()
//-------------------------------------------------------------------------
void main(void)
{
        unsigned char BufferIndex;
        unsigned char ReceiveString  [DATA_LENGTH];
```

```
	CKCON0 &= ~0x07;                        // system clock / 1
	INITIAL_TWSI ();                        // initial interrupt and priority

	while (1) {
		if (twsi.Stage != I2C_Disable) {
			if (twsi.completeAByte == 1) {
				switch (twsi.Stage)	{
				case I2C_SLA_with_W:
					BufferIndex = 0;	// initial BufferIndex
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
					break;

				case I2C_SLA_with_R:
					// prepare MSB on SDA pin
					twsi.IICByte = ReceiveString [0];
					SDA = twsi.IICByte & 0x80;

					BufferIndex = 0;	// initial BufferIndex
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
					break;

				case I2C_SL_W_ACK:
					ReceiveString [BufferIndex] = twsi.IICByte;
					twsi.completeAByte = 0;

					BufferIndex ++;		// limit BufferIndex 0~31
					BufferIndex &= 0x1F;
					twsi.Stage = I2C_SlaveStandby;

					break;

				case I2C_SL_R_ACK:
					BufferIndex ++;		// limit BufferIndex 0~31
					BufferIndex &= 0x1F;

					twsi.IICByte = ReceiveString [BufferIndex];
					SDA = twsi.IICByte & 0x80;
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
					break;

				case I2C_SL_R_NAK:
					SDA = 1;
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
					break;

				}
			}

		}

		// to do ...
	}

}

//-------------------------------------------------------------------------
// inital TWSI's SDA (STAF & STOF) edge detection
//-------------------------------------------------------------------------
void SystemFlag_ISR   (void) interrupt 7
{
	unsigned char tempReg;

	tempReg = AUXR1;
	AUXR1 &= ~(STAF+STOF);                  // clear STAF & STOF flag
```

```
        if (tempReg & STOF)     {
                EX1 = 0;
                twsi.Stage = I2C_Disable;

        } else if (tempReg & STAF){
                AUXR0 |= INT1H;                         // SCL raise edge detection
                _nop_ ();
                IE1 = 0;
                SDA = 1;

                EX1 = 1;                                // enable EX1
                position = 0xFF;
                twsi.Slave_RW = 0;                      // clear for receive a byte or address

                twsi.Stage = I2C_SlaveStandby;
                firstByte = 1;
        }
}

//-------------------------------------------------------------------------
// access SDA by EX1 interrupt

//-------------------------------------------------------------------------
void TWSI_EX1_ISR (void) interrupt 2
{
        position ++;

        if ((twsi.Slave_RW) == 0) {
                if (position < 8) {                     // 0~7th bit
                        tempByte = tempByte    << 1;    // 6th
                        tempByte |= SDA;

                        if (position == 7) {            // detect falling edge
                                AUXR0 &= ~INT1H;
                                _nop_ ();
                                IE1 = 0;
                                return;
                        } else {
                                IE1 = 0;
                                return;
                        }

                } else if (position == 8){              // 9th bit - ACK bit

                        if (firstByte) {
                                if ((tempByte >> 1) == twsi.ADDR) {
                                        SDA = 0;
                                } else {
                                        EX1 = 0;
                                        twsi.Stage = I2C_Disable;
                                }

                        } else {
                                SDA = 0;
                        }

                } else {
                        position = 0xFF;                // reset position
                        AUXR0 |= INT1H;                 // reset SCL interrupt for raising edge detection
                        _nop_ ();
                        IE1 = 0;
                        SDA = 1;

                        if (firstByte) {
                                firstByte = 0;

                                if ((tempByte &     0x01) == 0x01) {
                                        twsi.Slave_RW = 1;
                                        twsi.Stage = I2C_SLA_with_R;
```

```
                                // for SCL falling edge    detection
                                AUXR0 &= ~INT1H;
                                _nop_ ();
                                IE1 = 0;

                        } else {
                                twsi.Slave_RW = 0;
                                twsi.Stage = I2C_SLA_with_W;
                        }
                } else {
                        twsi.Stage = I2C_SL_W_ACK;
                }

                twsi.IICByte = tempByte;
                twsi.completeAByte = 1;         // set '1' when it tranfer a byte

        }
    } else {
        if (position < 7) {
                twsi.IICByte = twsi.IICByte << 1;       // send 6~0th bit to SDA
                SDA = twsi.IICByte & 0x80;
        } else if (position == 8) {
                twsi.completeAByte = 1;         // set '1' when it tranfer a byte
                position = 0xFF;                        // reset position

                if (SDA) {
                        twsi.Stage = I2C_SL_R_NAK;
                } else {
                        twsi.Stage = I2C_SL_R_ACK;
                }
                return;
        } else {                                        // ACK/NAK bit
                SDA = 1;
        }

    }
}
```

(2). Required Function: TWI Slave on SYSCLK=24MHz in burst mode:

Assembly Code Example:

```
$INCLUDE (REG_MG86FE508.INC)

SLAVE_DEV_ADDR  EQU     20H     ; declare slave device address
DATA_LENGTH     EQU     32      ; declare buffer size

;-------------------------------------------------------------------------
; declare the TWSI state
;-------------------------------------------------------------------------
I2C_SlaveStandby    EQU     0x00
I2C_SLA_with_W      EQU     0x01
I2C_SLA_with_R      EQU     0x02
I2C_Disable         EQU     0x03
I2C_SL_W_ACK        EQU     0x04
I2C_SL_R_ACK        EQU     0x05
I2C_SL_R_NAK        EQU     0x06

;-------------------------------------------------------------------------
; declare the TWSI pin
;-------------------------------------------------------------------------
SDA                 EQU     P3.4
SCL                 EQU     P3.3

;-------------------------------------------------------------------------
; data area
;-------------------------------------------------------------------------
```

```
CONTROLDATA  SEGMENT DATA
             RSEG CONTROLDATA
ReceiveString:      DS      DATA_LENGTH  ; data buffer
STACK:              DS      40           ; stack area size
position:           DS      1
tempByte:           DS      1

ADDR:               DS      1
IICByte:            DS      1
Stage:              DS      1

BITDATA SEGMENT BIT
             RSEG BITDATA
firstByte:          DBIT    1            ; the flag for receive SLA+R/W
completeAByte:      DBIT    1            ; set complete flag when transfer/receive one byte
Slave_RW:           DBIT    1            ; clear Slave_RW to beceive / set to transfer
DisableTWSI:        DBIT    1            ; clear DisableTWSI to active TWSI transreceiver
StartTWSI:          DBIT    1


;------------------------------------------------------------------------
; code area
;------------------------------------------------------------------------
        CSEG    AT 0000H                 ;start address = 0x0000
        JMP     ASSEMBLY_MAIN

        CSEG    AT 0013H                 ; EX0 interrupt ISR address
        JMP     SCL_DETECT_ISR

        CSEG    AT 003BH                 ; detect STAF or STOF ISR address
        JMP     SystemFlag_ISR

TWSI_CS SEGMENT CODE
        RSEG    TWSI_CS
        USING   0

ASSEMBLY_MAIN:
        MOV     SP,#STACK                ; initial SP for stack size
        ANL     CKCON0,#11111000B        ; system clock / 1
        CALL    INITIAL_TWSI             ; initial TWSI

MAIN_LOOP:
        ; to do     ...

        MOV     ACC,Stage
        XRL     A,#I2C_Disable
        JZ      MAIN_LOOP

        JNB     completeAByte,MAIN_LOOP  ; have a event ?

        ; ---------------------------------------------
        MOV     ACC,Stage
        CJNE    A,#I2C_SLA_with_W,SUBROUTINE_I2C_SLA_with_R

SUBROUTINE_I2C_SLA_with_W:
        MOV     R1,#ReceiveString        ; initial for receive
        CLR     completeAByte            ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SLA_with_R:
        CJNE    A,#I2C_SLA_with_R,SUBROUTINE_I2C_SL_W_ACK

        MOV     R1,#ReceiveString        ; initial for transfer
        MOV     A,@R1
        MOV     IICByte,A
        RLC     A                        ; it must transfer MSB to SDA
        MOV     SDA,C
        CLR     completeAByte            ; clear event flag
        JMP     MAIN_LOOP
```

```
SUBROUTINE_I2C_SL_W_ACK:
        CJNE    A,#I2C_SL_W_ACK,SUBROUTINE_I2C_SL_R_ACK
        MOV     @R1,IICByte                     ; save data  to "ReceiveString"

        INC     R1                              ; limit buffer index
        CJNE    R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV     R1,#ReceiveString

        CLR     completeAByte                   ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_ACK:
        CJNE    A,#I2C_SL_R_ACK,SUBROUTINE_I2C_SL_R_NAK

        INC     R1                              ; limit buffer index
        CJNE    R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV     R1,#ReceiveString

        MOV     IICByte,@R1                     ; prepare data form data buffer
        MOV     ACC,@R1
        RLC     A
        MOV     SDA,C                           ; SDA = MSB

        CLR     completeAByte                   ; clear event flag
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_NAK:
        CJNE    A,#I2C_SL_R_NAK,MAIN_LOOP

        SETB    SDA                             ; NAK-event
        CLR     completeAByte
        JMP     MAIN_LOOP


;-------------------------------------------------------------------------
; initial TWSI interrupt (priority)       & trigger mode
;-------------------------------------------------------------------------
INITIAL_TWSI:
        ; System Flag have the highest priority
        ORL     EIP1H,#08H
        ORL     EIP1L,#08H

        ; the EX1 priority
        ORL     IP0H,#00000100B
        ANL     IP0L,#11111011B

        ; enable ETWSI
        ORL     EIE1,#ESF
        ORL     SFIE,#SDIFIE
        SETBEA

        ; P33 & P34 is open drain mode for TWSI
        MOV     P3M0,#18H
        MOV     P3M1,#18H

        ; edge detect
        SETB    IT1
        ORL     AUXR0,#INT1H

        ; declare slave     device address
        MOV     ADDR,#SLAVE_DEV_ADDR
        MOV     Stage,#I2C_Disable
        CLR     completeAByte

        RET

;-------------------------------------------------------------------------
; inital TWSI's     SDA  (STAF &     STOF) edge detection
```

```
;-------------------------------------------------------------------
SystemFlag_ISR:
        PUSH    ACC
        PUSH    PSW

        MOV     ACC, AUXR1                      ; check STAF or   STOF ?
        JB      ACC.3, STAF_ROUTINE
        JB      ACC.2, STOF_ROUTINE

EXIT_FLAG_ISR:
        POP     PSW
        POP     ACC
        RETI

STAF_ROUTINE:                                   ; start of TWSI
        ORL     AUXR0, #INT1H   ; initial EX0 for raising edge detection and enable EX0 interrupt
        NOP
        CLR     IE1
        SETB    SDA
        SETB    EX1

        CLR     DisableTWSI                     ; clear DisableTWSI flag to 0 (= active TWSI)
        ANL     AUXR1, #~STAF                   ; clear STAF flag

        CLR     Slave_RW                        ; clear for receive a byte or address
        MOV     Stage,#I2C_SlaveStandby
        SETB    firstByte                       ; address byte flag
        SETB    StartTWSI
        JMP     EXIT_FLAG_ISR

STOF_ROUTINE:                                   ; stop of TSWI
        CLR     EX1                             ; disable EX0 interupt service routine
        ANL     AUXR1,#~STOF                    ; clear STOF flag
        SETB    DisableTWSI                     ; disable DisableTWSI (= inactive TWSI)
        MOV     Stage,#I2C_Disable
        JMP     EXIT_FLAG_ISR

;-------------------------------------------------------------------------
;   access SDA by EX1 interrupt
;-------------------------------------------------------------------------
SCL_DETECT_ISR:
        PUSH    ACC
        PUSH    PSW

        JNB     Slave_RW, SLAVE_WRITE
        JMP     SLAVE_READ
        ;-----------------------------------------------------------------------------
SLAVE_WRITE:
        JNB     StartTWSI,$+5
        CLR     StartTWSI
        MOV     C, SDA                          ; MSB - bit  7
        MOV     A, tempByte                     ; left shift SDA to  tempByte.0
        RLC     A
        MOV     tempByte, A

        CLR     IE1                             ; wait for IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3                ; avoid STOF event

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                          ; bit 6
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3
```

```
    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR
    MOV     C, SDA                      ; bit 5
    MOV     A, tempByte
    RLC     A
    MOV     tempByte, A
    CLR     IE1
    JB      IE1, $+6
    JNB     DisableTWSI, $-3

    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR
    MOV     C, SDA                      ; bit 4
    MOV     A, tempByte
    RLC     A
    MOV     tempByte, A
    CLR     IE1
    JB      IE1, $+6
    JNB     DisableTWSI, $-3

    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR
    MOV     C, SDA                      ; bit 3
    MOV     A, tempByte
    RLC     A
    MOV     tempByte, A
    CLR     IE1
    JB      IE1, $+6
    JNB     DisableTWSI, $-3

    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR
    MOV     C, SDA                      ; bit 2
    MOV     A, tempByte
    RLC     A
    MOV     tempByte, A
    CLR     IE1
    JB      IE1, $+6
    JNB     DisableTWSI, $-3

    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR
    MOV     C, SDA                      ; bit 1
    MOV     A, tempByte
    RLC     A
    MOV     tempByte, A
    CLR     IE1
    JB      IE1, $+6
    JNB     DisableTWSI, $-3

    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR
    MOV     C, SDA                      ; bit 0
    MOV     A, tempByte
    RLC     A
    MOV     tempByte, A
    CLR     IE1

    JB      DisableTWSI, EXIT_WITHOUT_COMPLETE_FLAG

    JNB     StartTWSI,$+6
    LJMP    EXIT_SCL_DETECT_ISR

    ANL     AUXR0, #~INT1H              ; set EX1 to falling edge detection
    NOP
    CLR     IE1
    JB      IE1, $+6
    JNB     DisableTWSI, $-3
```

```
        JNB     firstByte,SLAVE_WRITE_RESPONSE_ACK

        MOV     ACC,tempByte
        CLR     C
        RRC     A
        CJNE    A,ADDR,NOT_SLAVE_ADDR
SLAVE_WRITE_RESPONSE_ACK:
        CLR     SDA
        JMP     COMPLETE_WRITE_ONE_BYTE

NOT_SLAVE_ADDR:
        CLR     EX1
        MOV     Stage,#I2C_Disable
        JMP     EXIT_SCL_DETECT_ISR

COMPLETE_WRITE_ONE_BYTE:                ; 9th falling edge
        CLR     IE1
        JB      IE1, $+6                ; wait for the 9th bit
        JNB     DisableTWSI, $-3

        SETB    SDA                     ; set SDA for input

        ORL     AUXR0, #INT1H           ; set EX1 to edge detection
        NOP
        CLR     IE1

        SETB    completeAByte           ; set '1' when it receives a byte
        JNB     firstByte,REPEAT_RECEIVE_MODE
        CLR     firstByte

        ; SLA+W   or SLA+R ?
        CLR     Slave_RW
        MOV     ACC,tempByte
        JNB     ACC.0,SET_IN_SLAW_MODE
        SETB    Slave_RW

        ANL     AUXR0,#~INT1H           ; the falling edge of ACK signal
        NOP
        CLR     IE1

        MOV     Stage,#I2C_SLA_with_R
        JMP     EXIT_SCL_DETECT_ISR

SET_IN_SLAW_MODE:
        MOV     Stage,#I2C_SLA_with_W
        JMP     EXIT_SCL_DETECT_ISR

REPEAT_RECEIVE_MODE:
        MOV     IICByte,tempByte
        MOV     Stage,#I2C_SL_W_ACK
        ;-------------------------------------------------------------------------
EXIT_SCL_DETECT_ISR:
        POP     PSW
        POP     ACC
        RETI

EXIT_WITHOUT_COMPLETE_FLAG:
        CLR     completeAByte
        JMP     EXIT_SCL_DETECT_ISR
        ;-------------------------------------------------------------------------
SLAVE_READ:
        ; must transfer tempByte.7 in main routine
        ; and set EX1(SCL) for falling edge  detection
        JNB     StartTWSI,$+5
        CLR     StartTWSI

        MOV     A,IICByte               ; transfer IICByte.6
        RL      A
        MOV     IICByte,A
```

```
RLC     A
MOV     SDA,C

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
CLR     IE1                     ; wait for IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3        ; avoid STOF event

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte               ; transfer IICByte.5
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte               ; transfer IICByte.4
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte               ; transfer IICByte.3
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte               ; transfer IICByte.2
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte               ; transfer IICByte.1
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte               ; transfer IICByte.0
RL      A
MOV     IICByte,A
```

```
        RLC     A
        MOV     SDA,C
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP EXIT_SCL_DETECT_ISR

        SETB SDA                                ; for 9th bit - ACK / NAK
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     SDA,SET_I2C_SLAVE_READ_ACK
        MOV     Stage,#I2C_SL_R_NAK
        JMP     COMPLETE_READ_ONE_BYTE

SET_I2C_SLAVE_READ_ACK:
        MOV     Stage,#I2C_SL_R_ACK

COMPLETE_READ_ONE_BYTE:
        CLR     IE1
        SETB    completeAByte                   ; set '1' when it tranfer a byte

        JMP     EXIT_SCL_DETECT_ISR

;-------------------------------------------------------------------------------
        END
```

C Code Example:

```
#include <REG_MG86FE508.H>
#include <intrins.h>

#define SLAVE_DEV_ADDR  0x20                    // declare slave device address
#define DATA_LENGTH     32                      // declare buffer size

//-------------------------------------------------------------------------
// declare I2C stage
//-------------------------------------------------------------------------
#define I2C_SlaveStandby    0x00
#define I2C_SLA_with_W      0x01
#define I2C_SLA_with_R      0x02
#define I2C_Disable         0x03
#define I2C_SL_W_ACK        0x04                // SLA_W with data ACK
#define I2C_SL_R_ACK        0x05                // SLA_R with data ACK
#define I2C_SL_R_NAK        0x06                // SLA_R with data NAK

//-------------------------------------------------------------------------
// declare global variable
//-------------------------------------------------------------------------
typedef     struct {
        unsigned char ADDR;
        unsigned char IICByte;
        unsigned char Stage:8;
        unsigned char completeAByte:1;
        unsigned char Slave_RW:1;
} _TWSI;

_TWSI twsi;
unsigned char tempByte;
bit firstByte,DisableTWSI,StartTWSI;

//-------------------------------------------------------------------------
// declare the TWSI       pin
//-------------------------------------------------------------------------
sbit SDA = P3^4;
sbit SCL = P3^3;
```

```
//---------------------------------------------------------------------------
// initial TWSI       interrupt (priority) & trigger mode
//---------------------------------------------------------------------------
void INITIAL_TWSI ()
{
        // System Flag have the highest priority
        EIP1H |= 0x08;
        EIP1L |= 0x08;

        // the EX1 have normal priority
        IP0H |= 0x08;
        IP0L &= ~0x08;

        EIE1 |= ESF;                                    // enable ETWSI
        SFIE |= SDIFIE;
        EA = 1;

        // P33 & P34 is open drain mode for TWSI
        P3M0 = 0x18;
        P3M1 = 0x18;

        IT1 = 1;
        AUXR0 |= INT1H;

        // declare slave device address
        twsi.ADDR = SLAVE_DEV_ADDR;
        twsi.completeAByte = 0;
        twsi.Stage = I2C_Disable;
}

//---------------------------------------------------------------------------
// main()
//---------------------------------------------------------------------------
void main(void)
{
        unsigned char BufferIndex;
        unsigned char ReceiveString  [DATA_LENGTH];

        CKCON0 &= ~0x07;                                // system clock / 1
        INITIAL_TWSI ();                                // initial interrupt and priority

        while (1) {
                if (twsi.Stage != I2C_Disable) {
                        if (twsi.completeAByte == 1) {
                                switch (twsi.Stage)       {
                                case I2C_SLA_with_W:
                                        BufferIndex = 0;          // initial BufferIndex
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SLA_with_R:
                                        // prepare MSB on SDA pin
                                        twsi.IICByte = ReceiveString [0];
                                        SDA = twsi.IICByte & 0x80;

                                        BufferIndex = 0;          // initial BufferIndex
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SL_W_ACK:
                                        ReceiveString [BufferIndex] = twsi.IICByte;
                                        twsi.completeAByte = 0;

                                        BufferIndex ++;           // limit BufferIndex 0~31
                                        BufferIndex &= 0x1F;
                                        twsi.Stage = I2C_SlaveStandby;
```

```
                                    break;

                                case I2C_SL_R_ACK:
                                    BufferIndex ++;             // limit BufferIndex 0~31
                                    BufferIndex &= 0x1F;

                                    twsi.IICByte = ReceiveString [BufferIndex];
                                    SDA = twsi.IICByte & 0x80;
                                    twsi.completeAByte = 0;
                                    twsi.Stage = I2C_SlaveStandby;
                                    break;

                                case I2C_SL_R_NAK:
                                    SDA = 1;
                                    twsi.completeAByte = 0;
                                    twsi.Stage = I2C_SlaveStandby;
                                    break;

                                }
                            }

                        }

                        // to do ...
                    }

}

//--------------------------------------------------------------------------
// inital TWSI's SDA (STAF & STOF) edge detection

//--------------------------------------------------------------------------
void SystemFlag_ISR (void) interrupt 7
{
    unsigned char tempReg;

    tempReg = AUXR1;
    AUXR1 &= ~(STAF+STOF);                  // clear STAF & STOF flag

    if (tempReg & STOF) {
        EX1 = 0;
        DisableTWSI = 1;
        twsi.Stage = I2C_Disable;

    } else if (tempReg & STAF){
        AUXR0 |= INT1H;
        _nop_ ();
        IE1 = 0;
        SDA = 1;
        EX1 = 1;
        DisableTWSI = 0;                    // avoid mistake

        twsi.Slave_RW = 0;                  // clear for receive a byte or address

        twsi.Stage = I2C_SlaveStandby;
        firstByte = 1;
        StartTWSI = 1;
    }

}

// ---------------------------------------------------------------------
//    access SDA by EX1 interrupt
// ---------------------------------------------------------------------
void TWSI_EX1_ISR(void) interrupt 2
{
    if (twsi.Slave_RW == 0) {
        if (StartTWSI) {
            StartTWSI = 0;
```

```
    }
    tempByte = tempByte << 1;                   // bit 7
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);
                                                // bit 6
    if (StartTWSI) return;
    tempByte = tempByte << 1;
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   // bit 5
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   // bit 4
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   // bit 3
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   // bit 2
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   // bit 1
    tempByte |= SDA;
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   // bit 0
    tempByte |= SDA;

    AUXR0 &= ~INT1H;                            // for SCL edge detection
    _nop_ ();
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);           // 0th falling edge

    if (StartTWSI) return;
    if (DisableTWSI) return;

    if (firstByte) {
        if ((tempByte >> 1) == SLAVE_DEV_ADDR) {
            SDA = 0;
        } else {
            EX1 = 0;
            twsi.Stage = I2C_Disable;
        }
    } else {
        SDA = 0;
    }

    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);
    SDA = 1;
```

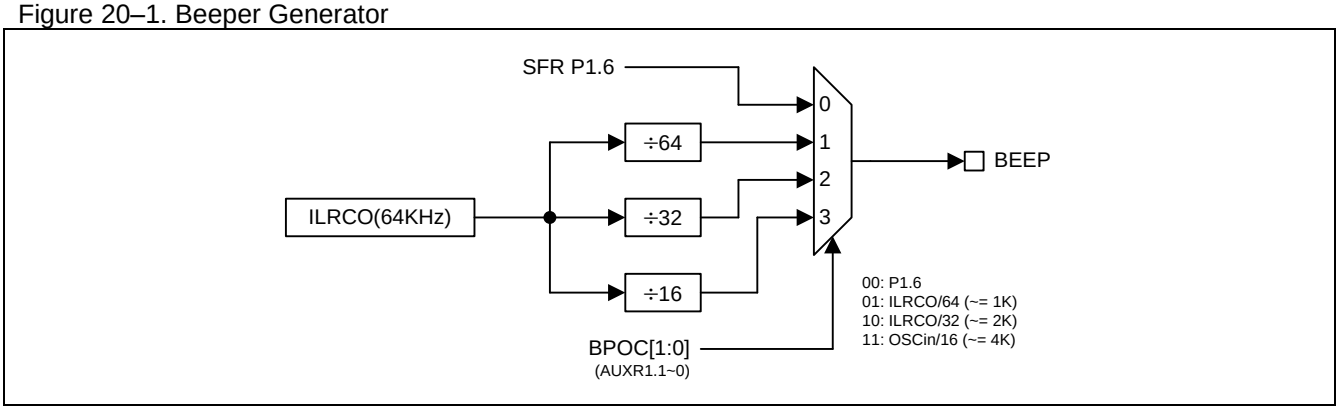
```
        AUXR0 |= INT1H;                 // for SCL raising edge detection
        _nop_ ();
        IE1 = 0;

        if (firstByte) {
            firstByte = 0;

            twsi.Slave_RW = (tempByte & 0x01);
            if (tempByte & 0x01) {
                twsi.Slave_RW = 1;
                twsi.Stage = I2C_SLA_with_R;

                // for SCL falling edge detection
                AUXR0 &= ~INT1H;
                _nop_ ();
                IE1 = 0;

            } else {
                twsi.Slave_RW = 0;
                twsi.Stage = I2C_SLA_with_W;
            }
        } else {
            twsi.Stage = I2C_SL_W_ACK;
        }

        twsi.IICByte = tempByte;
        if (DisableTWSI) return;
        twsi.completeAByte = 1;         // set '1' when it tranfer a byte
        P35 = 1;

    } else {
        if (StartTWSI) {
            StartTWSI = 0;
        }
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;      // bit 6
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;      // bit 5
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;      // bit 4
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;      // bit 3
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;      // bit 2
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;      // bit 1
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
```

```
            if (StartTWSI) return;
            twsi.IICByte = twsi.IICByte << 1;
            SDA = twsi.IICByte & 0x80;                  // bit 0
            IE1 = 0;
            while ((IE1 | DisableTWSI) == 0);

            SDA = 1;                                    // ACK
            IE1 = 0;
            while ((IE1 | DisableTWSI) == 0);
            IE1 = 0;

            if (DisableTWSI) return;
            if (SDA) {
                twsi.Stage = I2C_SL_R_NAK;
            } else {
                twsi.Stage = I2C_SL_R_ACK;
            }

            twsi.completeAByte = 1;
        }

}
```

20. Beeper

The beeper function outputs a signal on the BEEP pin for sound generation. The signal is in the range about 1, 2 or 4 kHz which is divided from ILRCO. Figure 20–1 shows the beeper generator circuit. But ILRCO is not the precision clock source. Please refer Section "27.5 ILRCO Characteristics" for more detailed ILRCO frequency deviation range.

Figure 20–1. Beeper Generator



20.1. Beeper Register

AUXR1: Auxiliary Control Register 1

SFR Page = Normal

SFR Address = 0xA2 RESET = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	**BPOC1**	**BPOC0**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1~0: BPOC1~0, Beeper output control bits.

BPOC[1:0]	P1.6 function	I/O mode
00	P1.6	By P1M0.6
01	ILRCO/64	By P1M0.6
10	ILRCO/32	By P1M0.6
11	ILRCO/16	By P1M0.6

For beeper on P1.6 function, it is recommended to set P1M0.6 to "1" which selects P1.6 as push-push output mode.

20.2. Beeper Sample Code

(1). Required Function: Set Beeper output 1KHz

Assembly Code Example:

```
ORL     P1M0,#40H                       ; Set P1.6 to push-pull output mode

ANL     AUXR1,#~(BPOC1|BPOC0)           ; Set P1.6 as GPIO function
ORL     AUXR1,#BPOC0                    ; BEEP = ILRCO/64 ~= 1KHz
```

C Code Example:

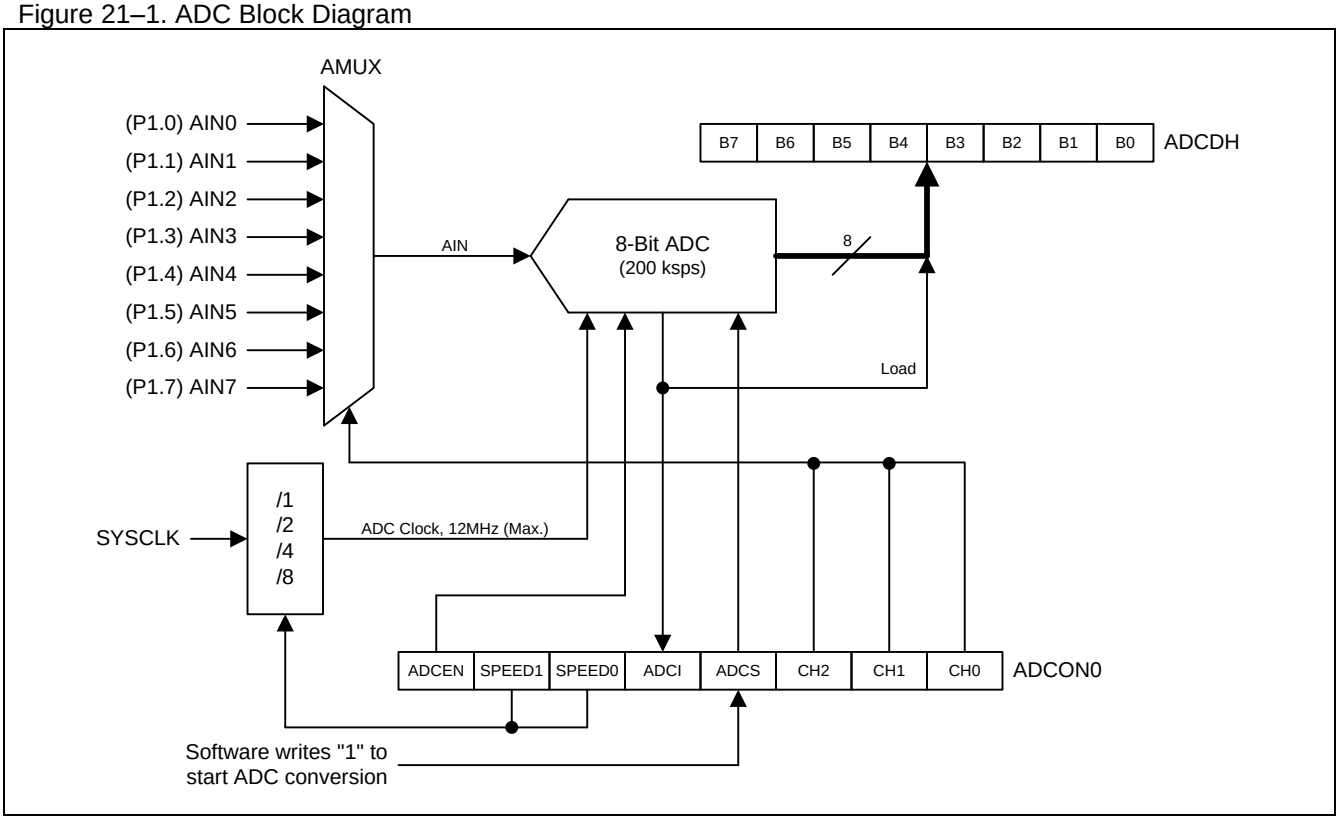
```
P1M0 = P1M0 | 0x40;                     // Set P1.6 to push-pull output mode

AUXR1 &= ~(BPOC1 | BPOC0);              // Set P1.6 as GPIO function
AUXR1 |= BPOC0;                         // BEEP = ILRCO/64 ~= 1KHz
```

21. 8-Bit ADC

The ADC subsystem for the **MG86FE/L508** consists of an analog multiplexer (AMUX), and a **200** ksps, **8**-bit successive-approximation-register ADC. The AMUX can be configured via the Special Function Registers shown in Figure 21–1. ADC operates in Single-ended mode, and may be configured to measure any of the pins on Port 1. The ADC subsystem is enabled only when the ADCEN bit in the ADC Control register (ADCON0) is set to logic 1. The ADC subsystem is in low power shutdown when this bit is logic 0.

21.1. ADC Structure

Figure 21–1. ADC Block Diagram



21.2. ADC Operation

ADC has a maximum conversion speed of 200 ksps. The ADC conversion clock is a divided version of the system clock, determined by the SPEED1~0 bits in the ADCON0 register. The ADC conversion clock should be no more than **12MHz**.

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

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

21.2.1. 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 ADCON0 register as described in Figure 21–1. The selected pin is measured with respect to GND.

21.2.2. Starting a Conversion

Prior to using the ADC function, the user should:

1) Turn on the ADC hardware by setting the ADCEN 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 Analog-Input-Only mode by P1, P1M0 and P1AIO 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 ADCH 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 EADC (in EIE1 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.

21.2.3. ADC Conversion Time

The user can select the appropriate conversion speed according to the frequency of the analog input signal. The maximum input clock of the ADC is **12MHz** and it operates a fixed conversion time with **60** ADC clocks. User can configure the SPEED1~0 in ADCON0 to specify the conversion rate. For example, if SYSCLK =12MHz and the ADCKS = SYSCLK/1 is selected, then the frequency of the analog input should be no more than 200KHz to maintain the conversion accuracy. (Conversion rate = 12MHz/1/60 = 200KHz.)

21.2.4. 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 “13 Configurable I/O Ports”.

When an analog signal is applied to the ADCI7~0 pin and the digital input from this pin is not needed, software could set the corresponding pin to analog input only in P1AIO to reduce power consumption in the digital input buffer.

21.2.5. 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 (ADCEN=0) before entering Idle mode and Power-Down mode.

21.3. ADC Register

ADCON0: ADC Control Register 0

SFR Page = Normal
SFR Address = 0xC4 RESET = 0000-0000

7	6	5	4	3	2	1	0
ADCEN	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: ADCEN, 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/1
0 1	SYSCLK/2
1 0	SYSCLK/4
1 1	SYSCLK/8

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.

In Single-ended mode:

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)

ADCDH: ADC Date High Byte Register

SFR Page = Normal
SFR Address = 0xC6 RESET = xxxx-xxxx

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

In **MG86FE/L508**, 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	ADCDH
VDD x 255/256	0xFF
VDD x 128/256	0x80
VDD x 64/256	0x40
VDD x 32/256	0x20
0	0x00

P1AIO: Port 1 Analog Input Only

SFR Page = Normal

SFR Address = 0x92 RESET = 0000-0000

7	6	5	4	3	2	1	0
P17AIO	P16AIO	P15AIO	P14AIO	P13AIO	P12AIO	P11AIO	P10AIO
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: Port pin has digital and analog input capability.

1: Port pin only has analog input only. The corresponding Port PIN Register bit will always read as zero when this bit is set.

21.4. ADC Sample Code

(1). Required Function: ADC sample code for SYSCLK=24MHz (200K), transfer analog input on P1.0/P1.1/P1.2 with P1AIO setting:

Assembly Code Example:

```
ADC_SAMPLE_P1x:
INITIAL_ADC_PIN:
        ORL     P1AIO,#00000111B            ; P1.0, P1.1, P1.2 = Analog-input only
        ANL     CKCON0,#11111000B           ; system clock / 1

        MOV     ADCON0,#ADCEN               ; Enable ADC block
        ; delay 5us
        ; call ....

Get_P10:
        MOV     ADCON0,#(ADCEN | SPEED1 | ADCS)
                                            ; Enable ADC block & start conversion
                                            ; Speed at 200k @ 24MHz, select P1.0 for ADC input pin

        MOV     ACC,ADCON0                  ; check ready?
        JNB     ACC.4,$-3
        ANL     ADCON0,#~(ADCI | ADCS)      ; clear ADCI & ADCS
        MOV     AIN0_data,ADCDH             ; reserve P1.0 ADC data
        ; to do ...

Get_P11:
        MOV     ADCON0,#(ADCEN | SPEED1 | ADCS | CHS0)   ; select P1.1

        MOV     ACC,ADCON0                  ; check ready?
        JNB     ACC.4,$-3
        ANL     ADCON0,#~(ADCI | ADCS)      ; clear ADCI & ADCS
        MOV     AIN1_data,ADCDH
        ; to do ...

Get_P12:
        MOV     ADCON0,#(ADCEN | SPEED1 | ADCS | CHS1)   ; select P1.2

        MOV     ACC,ADCON0                  ; check ready?
        JNB     ACC.4,$-3
        ANL     ADCON0,#~(ADCI | ADCS)      ; clear ADCI & ADCS
        MOV     AIN2_data,ADCDH

        ; to do ...

        RET
```

C Code Example:

```
void main(void)
{
        unsigned char AIN0_data, AIN1_data, AIN2_data;

        P1AIO = 0x07;                       // P1.0, P1.1, P1.2 = Analog-input only
        CKCON0 &= ~0x07;                    // system clock / 1

        ADCON0 = ADCEN;                     // Enable ADC block
        // delay 5us
        // ...


        // select P1.0
        ADCON0 = ADCEN | SPEED1 | ADCS;
                                            // Enable ADC block & start conversion
                                            // Speed at 200k @ 24MHz, select P1.0 for ADC input pin

        while ((ADCON0 & ADCI) == 0x00);    //wait for complete
        ADCON0 &= ~(ADCI | ADCS);
```

```
    AIN0_data = ADCDH;
    // to do ...

    // select P1.1
    ADCON0 = ADCEN | SPEED1 | ADCS | CHS0;   // select P1.1

    while ((ADCON0 & ADCI) == 0x00);                //wait for complete
    ADCON0 &= ~(ADCI | ADCS);
    AIN1_data = ADCDH;
    // to do ...

    // select P1.2
    ADCON0 = ADCEN | SPEED1 | ADCS | CHS1;  // select P1.2

    while ((ADCON0 & ADCI) == 0x00);                //wait for complete
    ADCON0 &= ~(ADCI | ADCS);
    AIN2_data = ADCDH;

    // to do ...

    while (1);

}
```

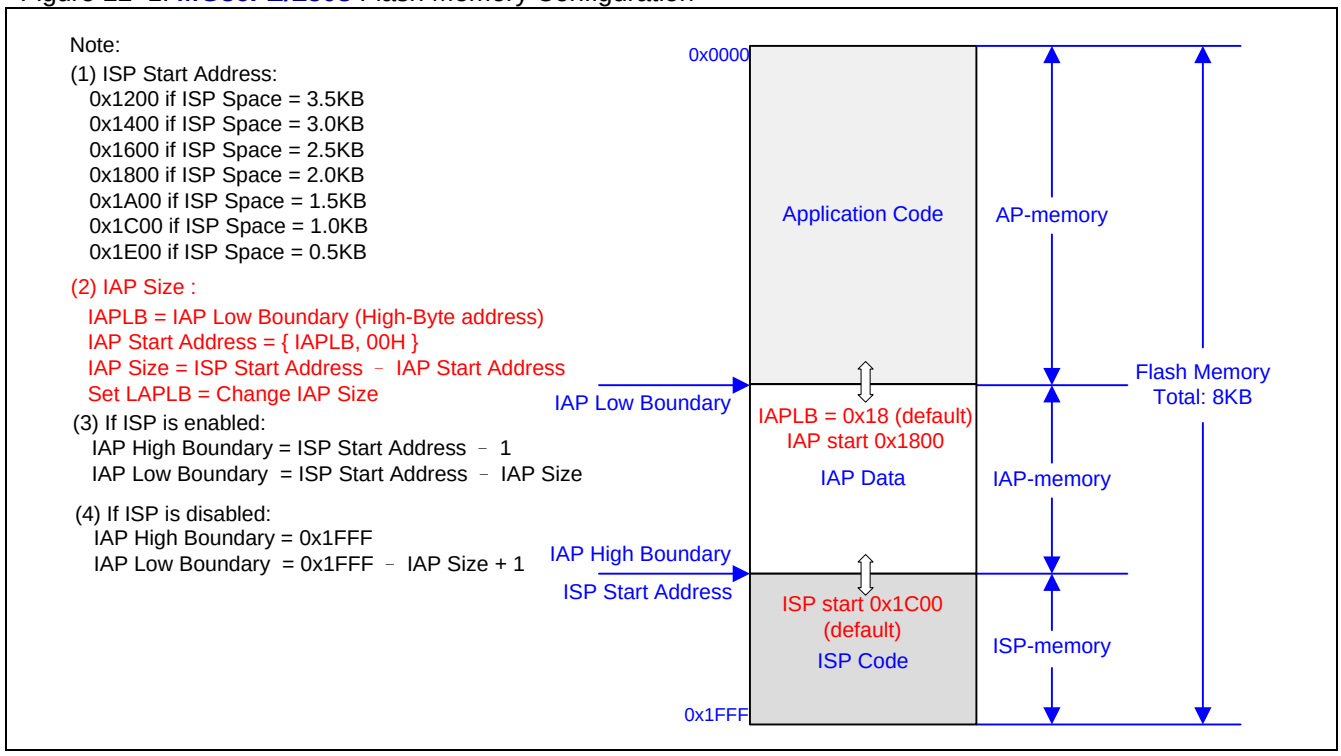
22. ISP and IAP

The flash memory of **MG86FE/L508** 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.

22.1. MG86FE/L508 Flash Memory Configuration

There are total 4K bytes of Flash Memory in **MG86FE/L508** and Figure 22–1 shows the device flash configuration of **MG86FE/L508**. The ISP-memory can be configured as disabled or up to 3.5K 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 or AP software programming. All of the AP, IAP and ISP memory are shared the total 8K bytes flash memory.

Figure 22–1. **MG86FE/L508** Flash Memory Configuration



Note:

*In default, the **MG86FE/L508** 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 1K IAP size can be re-configured by software for application required.*

22.2. MG86FE/L508 Flash Access in ISP/IAP

There are 2 flash access modes are provided in **MG86FE/L508** for ISP and IAP application: program mode and read mode. MCU software uses these two 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.

22.2.1. ISP/IAP Flash Program Mode

The “program” mode of **MG86FE/L508** 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 22–2 shows the flash byte program flow in ISP/IAP operation.

Figure 22–2. ISP/IAP byte Program Flow

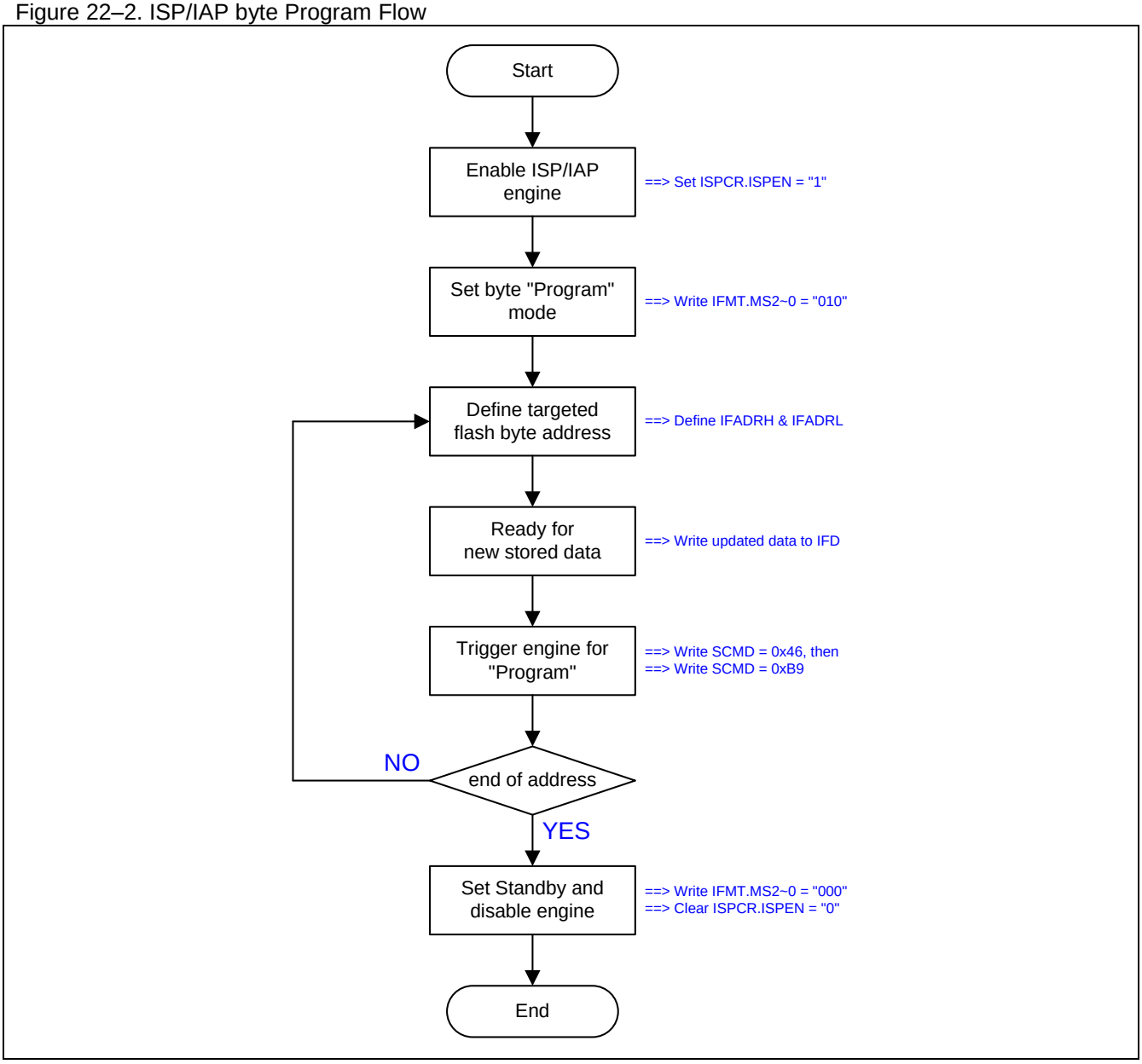


Figure 22–3 shows the demo code of the ISP/IAP byte program operation.

Figure 22–3. Demo Code for ISP/IAP byte Program

```
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
```

22.2.2. ISP/IAP Flash Read Mode

The “read” mode of **MG86FE/L508** 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 22–4 shows the flash byte read flow in ISP/IAP operation.

Figure 22–4. ISP/IAP byte Read Flow

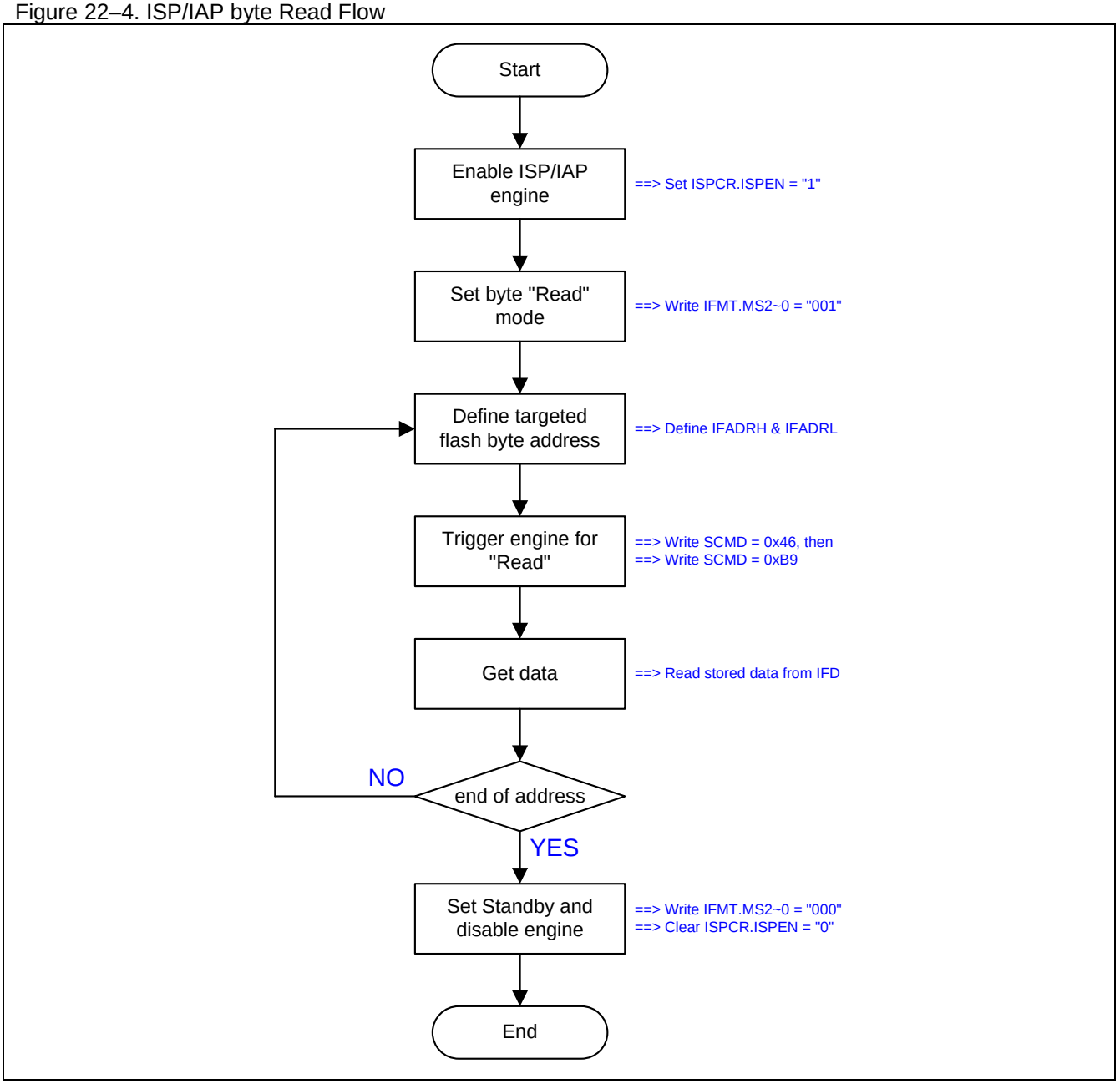


Figure 22–5 shows the demo code of the ISP/IAP byte read operation.

Figure 22–5. Demo Code for ISP/IAP byte Read

```
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
```

22.3. 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.

22.3.1. 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..

If the additional hardware option, HWBS2, is enabled with HWBS and ISP Memory, the MCU will always boot from ISP memory after power-on or **external reset finished**. It provides another hardware approached way to enter ISP mode by external reset signal. After first time power-on, **MG86FE/L508** can perform ISP operation by external reset trigger and doesn't wait for next time power-on, which suits the non-power-off system to apply the hardware approached ISP function.

22.3.2. 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.

22.3.3. 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 22–1), 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, nINTx, should keep activated until the ISP is completed, or they will be neglected.

ISP and Idle mode

MG86FE/L508 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 100 write cycles, that is to say, the write cycles shouldn't exceed 100 times. Thus the user should pay attention to it in the application which needs to frequently update the AP-memory and IAP-memory.

22.4. IAP Operation

The **MG86FE/L508** 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 **MG86FE/L508** 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.*

22.4.1. 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 size.

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 = 0x1FFF – IAP size + 1.

For example, if ISP-memory is 1K, so that ISP start address is 0x0C00, and IAP-memory is 1K, then the IAP-memory range is located at 0x1800 ~ 0x1BFF. The IAP low boundary in **MG86FE/L508** is defined by IAPLB register which can be modified by software to adjust the IAP size in user's AP program.

22.4.2. Update data in IAP-memory

The special function registers are related to ISP/IAP would be shown in Section "22.5 ISP/IAP Register".

To update "one byte" in the IAP-memory, users can directly program the new datum into that byte. To read the data in the IAP-memory, users can use the ***ISP/IAP Flash Read mode*** to get the targeted data.

22.4.3. 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 can not be in-time serviced during the MCU halts for IAP processing.
(2) The low/high-level triggered external interrupts, nINTx, should keep activated until the IAP is completed, or they will be neglected.

IAP and Idle mode

MG86FE/L508 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 100 write cycles, that is to say, the write cycles shouldn’t exceed 100 times. Thus the user should pay attention to it in the application which needs to frequently update the IAP-memory.

22.5. ISP/IAP Register

The following special function registers are related to the access of ISP, IAP and Page-P SFR:

IFD: ISP/IAP Flash Data Register

SFR Page = Normal
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/Page-P operation. The data in IFD will be written into the desired address in operating ISP/IAP/Page-P write and it is the data window of readout in operating ISP/IAP read.

IFADRH: ISP/IAP Address for High-byte addressing

SFR Page = Normal
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. It is not defined in Page-P mode.

IFADRL: ISP/IAP Address for Low-byte addressing

SFR Page = Normal
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/Page-P modes.

IFMT: ISP/IAP Flash Mode Table

SFR Page = Normal
SFR Address = 0xE5 RESET = xxxx-x000

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

Bit 7~4: Reserved. Software must write “0000_0” on these bits when IFMT is written.

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

MS[2:0]	Mode
0 0 0	Standby
0 0 1	Flash byte read of AP/IAP-memory
0 1 0	Flash byte program of AP/IAP-memory
0 1 1	Reserved
1 0 0	Page-P SFR Write
Others	Reserved

IFMT is used to select the flash mode for performing numerous ISP/IAP function or to select page P SFR access.

IAPLB: IAP Low Boundary

SFR Page = **Page P Only**
SFR Address = 0x03 RESET = 0001-1000

7	6	5	4	3	2	1	0
IAPLB							0
W	W	W	W	W	W	W	W

Bit 7~0: The IAPLB determines the IAP-memory lower boundary. Since a Flash page has 512 bytes, the IAPLB must be an even number.
To read IAPLB, MCU need to define the IMFT for mode selection on IAPLB Read and set ISPCR.ISPEN. And then write 0x46h & 0xB9h sequentially into SCMD. The IAPLB content is available in IFD. If write IAPLB, MCU will put new IAPLB setting value in IFD firstly. And then select IMFT, enable ISPCR.ISPEN and then set SCMD. The IAPLB content has already finished the updated sequence.

The range of the IAP-memory is determined by IAPLB and the ISP start address as listed below.
IAP lower boundary = IAPLBx256, and
IAP higher boundary = ISP start address – 1.

For example, if IAPLB=0x12 and ISP start address is 0x1C00, then the IAP-memory range is located at 0x1200 ~ 0x1BFF.

Additional attention point, the IAP low boundary address must not be higher than ISP start address.

SCMD: Sequential Command Data register
SFR Page = Normal
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/Page-P activity. If SCMD is filled with sequential 0x46h, 0xB9h and if ISPCR.7 = 1, ISP/IAP/Page-P activity will be triggered.

ISPCR: ISP Control Register
SFR Page = Normal
SFR Address = 0xE7 RESET = 0000-0xxx

7	6	5	4	3	2	1	0
ISPEN	SWBS	SWRST	CFAIL	--	--	--	--
R/W	R/W	R/W	R/W	W	W	W	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 3~0: Reserved. Software must write “0” on these bits when ISPCR is written.

22.6. ISP/IAP Sample Code

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

Assembly Code Example:

```
_ixp_read:
ixp_read:
    MOV     ISPCR,#ISPEN                ; Enable Function
    MOV     IFMT,#MS0                   ; ixp_read=0x01

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

    MOV     IFMT,#000h                  ; Flash_Standby=0x00
    ANL     ISPCR,#~ISPEN               ; Disable Function

    RET
```

C Code Example:

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

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

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

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

Assembly Code Example:

```
_ixp_program:
ixp_program:
    PUSH    ACC

    PUSH    IFADRH                      ;
    PUSH    IFADRL                      ;
    PUSH    IFD                         ;

    MOV     IFADRL,#WDTCR               ;
    MOV     IFD,WDTCR                   ;
    ORL     IFD,#CLRW                   ;
    CALL    page_p_sfr_write            ;

    POP     IFD                         ;
    POP     IFADRL                      ;
    POP     IFADRH                      ;

    MOV     ISPCR,#ISPEN                ; Enable Function
    MOV     IFMT,#MS1                   ; IxP_Write=0x02
    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;

    PUSH    IFD                         ;
    MOV     A,IFD                       ;
    CPL     A                           ;
    MOV     IFD,A                       ;

    MOV     IFMT,#MS0                   ; IxP_Read=0x01
    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;
```

```
;       if(reg[2] == IFD)
        POP         ACC                                 ;
        CJNE        A,IFD,ixp_first_progrma_fail        ;
        JMP         ixp_progrma_Pass                    ;

ixp_first_progrma_fail:
;       page_p_sfr_write (WDTCR_P,(WDTCR | CLRW)); //
        MOV         IFD,A                               ;
        PUSH        IFADRH                              ;
        PUSH        IFADRL                              ;
        PUSH        IFD                                 ;

        MOV         IFADRL,#WDTCR                       ;
        MOV         IFD,WDTCR                           ;
        ORL         IFD,#CLRW                           ;
        CALL        page_p_sfr_write                    ;

        POP         IFD                                 ;
        POP         IFADRL                              ;
        POP         IFADRH                              ;

        ANL         ISPCR,#~CFAIL                       ;
        MOV         IFMT,#MS1                           ; ixp_write=0x02
        MOV         SCMD,#046h                          ;
        MOV         SCMD,#0B9h                          ;

        PUSH        IFD                                 ;
        MOV         A,IFD                               ;
        CPL         A                                   ;
        MOV         IFD,A                               ;

        MOV         IFMT,#MS0                           ; IxP_Read=0x01
        MOV         SCMD,#046h                          ;
        MOV         SCMD,#0B9h                          ;

;       if(reg[2] == IFD)
        POP         ACC
        CJNE        A,IFD,ixp_second_progrma_Fail
ixp_progrma_Pass:
;       SETB        ixp_program_state                   ; Pass equ 1
        CLR         ixp_program_state                   ; Pass equ 0

end_ixp_program:
        MOV         IFMT,#000h                          ;
        ANL         ISPCR,#~ISPEN                       ;

        POP         ACC                                 ;
        RET                                             ;

ixp_second_progrma_Fail:
;       CLR         ixp_program_state                   ; Fail equ 0
        SETB        ixp_program_state                   ; Fail equ 1
        JMP         end_ixp_program                     ;
```

C Code Example:

```
bit ixp_program(void)
{     unsigned char reg[3];

        reg[0] = IFADRH;                                //
        reg[1] = IFADRL;                                //
        reg[2] = IFD;

        IFADRL = WDTCR_P;                               //
        IFD = (WDTCR | CLRW);                           //
        page_p_sfr_write ();                            //
```

```
    IFADRH = reg[0];                    //
    IFADRL = reg[1];                    //
    IFD = reg[2];                       //

    ISPCR = ISPEN;                      // Enable Function
    IFMT = IxP_Flash_Program;           // IxP_Write=0x02

    SCMD = 0x46;
    SCMD = 0xB9;

    IFD = ~reg[2];                      //

    IFMT = IxP_Flash_Read;              // IxP_Read=0x01
    SCMD = 0x46;                        //
    SCMD = 0xB9;                        //

    if(reg[2] == IFD)                   //
       {
         IFMT = Flash_Standby;           // Flash_Standby=0x00
         ISPCR &= ~ISPEN;                //
         return(Pass);                   //
       }
    else                                //
    {    IFADRL = WDTCR_P;              //
         IFD = (WDTCR | CLRW);          //
         page_p_sfr_write ();           //

         IFADRH = reg[0];               //
         IFADRL = reg[1];               //
         IFD = reg[2];                  //

         ISPCR &= ~CFAIL;               //
         IFMT = IxP_Flash_Program;      //IxP_Write=0x02

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

         IFD = ~reg[2];                 //

         IFMT = IxP_Flash_Read;         //IxP_Read=0x01
         SCMD = 0x46;                   //
         SCMD = 0xB9;                   //

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

         if(reg[2] != IFD)              //
         {
              return(Fail);             //
          }
                   else                 //
          {
              return(Pass);             //
          }
    }
}
```

(3). Required Function: ISP Sample code with CFAIL Checked and CLR WDT by hardware approached mode, Write twice if CFAIL true.

Assembly Code Example:

```
isp_hw_approached:
      MOV       DPH,#High(00000h)                ;
      MOV       DPL,#LOW(00000h)                 ;

isp_hw_write_loop:
      MOV       IFADRL,DPL                       ;
      MOV       IFADRH,DPH                       ;
      MOV       IFD,#055h                        ; ISP Program Data
      CALL      ixp_program                      ;

;     JNB       ixp_program_state,isp_hw_write_fail
                                                 ; Fail EQU 0

      JB        ixp_program_state,isp_hw_write_fail
                                                 ; Fail EQU 1
isp_hw_write_next:
      INC       DPTR                             ;

      MOV       A,DPH                            ;
      CJNE      A,#01Ch,isp_hw_write_loop        ;

      ANL       ISPCR,#~SWBS                     ;
      ORL       ISPCR,#SWRST                     ;

isp_hw_write_fail:
;           to do ...
      JMP       isp_hw_write_next                ;
```

C Code Example:

```
      unsigned short addr=0x0000;                //
      do{ IFADRH = (unsigned char)(addr >>8);
            IFADRL = (unsigned char)addr ;       //
            IFD = 0x55;                          //
            if(ixp_program() == Fail);           //
            {
//                to do ...
            }
      }while(++addr != 0x1C00);                  //
      ISPCR = SWRST;                             // Select AP Boot & SoftWave Reset
```

(4). Required Function: ISP Sample code by Sofrware approached entrance

Assembly Code Example:

```
      MOV       ISPCR,#(SWBS|SWRST)              ;
```

C Code Example:

```
      ISPCR = (SWBS | SWRST) ;                   //
```

(5). Required Function: IAPLB Change to on-chip flash 4K boundary

Assembly Code Example:

```
MOV       IFADRL,#IAPLB                    ;
MOV       IFD,#(HIGH(4096))&0xFE           ;
CALL      page_p_sfr_write                 ;
```

C Code Example:

```
IFADRL = IAPLB;                                  //
IFD = (((unsigned char)(4096 >> 8)) & 0xFE);     //
page_p_sfr_write();                              //
```

23. Page P SFR Access

MG86FE/L508 builds a special SFR page (Page P) to store the control registers for MCU operation. These SFRs can be accessed by the ISP/IAP operation with different IFMT. In page P access, IFADRH must set to “00” and IFADRL indexes the SFR address in page P. If IFMT= 04H for Page P writing, the content in IFD will be loaded to the SFR in IFADRL indexed after the SCMD triggered. If IFMT = 05H for Page P reading, the content in IFD is stored the SFR value in IFADRL indexed after the SCMD triggered.

Following descriptions are the SFR function definition in Page P:

IAPLB: IAP Low Boundary

SFR Page = Page P Only
SFR Address = 0x03 RESET = 0001-1000

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

Bit 7~0: The IAPLB determines the IAP-memory lower boundary.

To read IAPLB, MCU need to define the IFADRL for SFR address in Page-P, the IMFT for mode selection on IAPLB Read and set ISPCR.ISPEN. And then write 0x46h & 0xB9h sequentially into SCMD. The IAPLB content is available in IFD. If write IAPLB, MCU will put new IAPLB setting value in IFD firstly. And index IFADRL, select IMFT, enable ISPCR.ISPEN and then set SCMD. The IAPLB content has already finished the updated sequence.

The range of the IAP-memory is determined by IAPLB and the ISP Start address as listed below.
IAP lower boundary = IAPLBx256, and
IAP higher boundary = ISP start address – 1.

For example, if IAPLB=0x12 and ISP start address is 0x1C00, then the IAP-memory range is located at 0x1200 ~ 0x1BFF.

Additional attention point, the IAP low boundary address must not be higher than ISP start address.

CKCON2: Clock Control Register 2

SFR Page = Page P Only
SFR Address = 0x40 RESET = 1101-xx00

7	6	5	4	3	2	1	0
XTGS1	XTGS0	XTALE	IHRCOE	**0**	**0**	OSCS1	OSCS0
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**R/W**	**R/W**

Bit 7~6: XTGS1~XTGS0, OSC Driving control Register.

XTGS1, XTGS0	Gain Define
0, 0	Gain for 32.768K
1, 1	Gain for 2MHz ~ 25MHz
Others	Reserved

Bit 5: XTALE, external Crystal (XTAL) Enable.
0: Disable XTAL oscillating circuit. In this case, XTAL2 and XTAL1 behave as Port 4.0 and Port 4.1.
1: Enable XTAL oscillating circuit. If this bit is set by CPU software, software polls the **XTOR** (AUXR1.4) **true** to indicate the crystal oscillator is ready for clock selected.

Bit 4: IHRCOE, Internal High frequency RC Oscillator Enable. The default value is set for MCU clock source.
0: Disable internal high frequency RC oscillator.
1: Enable internal high frequency RC oscillator. If this bit is set by CPU software, it needs **32 us** to have stable output after IHRCOE enabled.

Bit 3~2: Reserved. Software must write “0” on these bits when CKCON2 is written.

Bit 1~0: OSCS[1:0], OSCin source selection. The default selection of OSCin is IHRCO.

OSCS[1:0]	OSCin source Selection
0 0	IHRCO
0 1	XTAL
1 0	ILRCO
1 1	ECKI, External Clock Input (P4.0) as OSCin.

PCON2: Power Control Register 2

SFR Page = Page P Only
SFR Address = 0x44 RESET = x0xx-xx01

7	6	5	4	3	2	1	0
--	AWBOD0	--	--	--	--	BO0RE	**1**
W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**W**

Bit 7: Reserved. Software must write “0” on this bit when PCON2 is written.

Bit 2: AWBOD0, Awaked BOD0 in PD mode. This function is only supported in **MG86FL508.**
0: BOD0 is disabled in power-down mode.
1: BOD0 keeps operation in power-down mode.

Bit 5~2: Reserved. Software must write “0” on these bits when PCON2 is written.

Bit 1: BO0RE, BOD0 Reset Enabled. Its initial is loaded from OR1.BO0REO inverted.
0: Disable BOD0 to trigger a system reset when BOF0 is set.
1: Enable BOD0 to trigger a system reset when BOF0 is set by VDD meets 4.2V(E) or 2.4V(L).

Bit 0: Reserved. Software must write “**1**” on this bit when PCON2 is written.

SPCON0: SFR Page Control 0

SFR Page = Page P Only
SFR Address = 0x48 RESET = xxx0-x000

7	6	5	4	3	2	1	0
RTCCTL	--	--	WRCTL	--	CKCTL0	PWCTL1	PWCTL0
W	**W**	**W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: RTCCTL. RTCCR SFR access Control.
If RTCCTL is set, it will disable the RTCCR SFR modified in general Page. RTCCR in general Page only keeps the SFR read function. But software always owns the modification capability in SFR Page P.

Bit 6~5: Reserved. Software must write “0” on these bits when SPCON0 is written.

Bit 4: WRCTL. WDTCR SFR access Control.
If WRCTL is set, it will disable the WDTCR SFR modified in general Page. WDTCR in general Page only keeps the SFR read function. But software always owns the modification capability in SFR Page P.

Bit 3: Reserved. Software must write “0” on this bit when SPCON0 is written.

Bit 2: CKCTL0. CKCON0 SFR access Control.
If CKCTL0 is set, it will disable the CKCON0 SFR modified in general Page. CKCON0 in general Page only keeps the SFR read function. But software always owns the modification capability in SFR Page P.

Bit 1: PWCTL1. PCON1 SFR access Control.
If PWCTL1 is set, it will disable the PCON1 SFR modified in general Page. PCON1 in general Page only keeps the SFR read function. But software always owns the modification capability in SFR Page P.

Bit 0: PWCTL0. PCON0 SFR access Control.
If PWCTL0 is set, it will disable the PCON0 SFR modified in general Page. PCON0 in general Page only keeps the SFR read function. But software always owns the modification capability in SFR Page P.

DCON0: Device Control 0

SFR Page = Page P Only
SFR Address = 0x4C RESET = 10xx-xx1x

7	6	5	4	3	2	1	0
HSE	IAPO	--	--	--	--	RSTIO	--
R/W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**W**

Bit 7: HSE, High Speed operation Enable.
0: Select MCU running in low speed mode which is slow down internal circuit to reduce power consumption.
1: Enable MCU full speed operation if F_{SYSCLK} > 6MHz.

Bit 6: IAPO, IAP function Only.
0: Maintain IAP region to service IAP function and code execution.
1: Disable the code execution in IAP region and the region only service IAP function.

Bit 5~0: Reserved. Software must write “0” on these bits when DCON0 is written.

Bit 1: RSTIO, RST function on I/O,
1: Select I/O pad function for RST.
0: Select I/O pad function for P36.

23.1. Page-P Sample Code

(1). Required Function: General function call of Page-P SFR Read

Assembly Code Example:

```
_page_p_sfr_read:
page_p_sfr_read:
    MOV     IFADRH,000h
    MOV     IFMT,#(MS2|MS0)             ; PageP_Read=0x05

    ANL     ISPCR,#CFAIL                ;
    ORL     ISPCR,#ISPEN                ; Enable Function

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

    MOV     IFMT,#000h                  ; Flash_Standby=0x00
    ANL     ISPCR,#~ISPEN               ; Disable Function

    RET
```

C Code Example:

```
void page_p_sfr_read (void)
{
    IFADRH = 0x00;                      //

    ISPCR = ISPEN;                      // Enable Function
    IFMT = (MS0 | MS2);                 // PageP_Read=0x05

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

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

(2). Required Function: General function call of Page-P SFR Write

Assembly Code Example:

```
_page_p_sfr_write:
page_p_sfr_write:
    MOV     IFADRH,000h                 ;

    MOV     ISPCR,#ISPEN                ; Enable Function
    MOV     IFMT,#MS2                   ; PageP_Write=0x04

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

    MOV     IFMT,#000h                  ; Flash_Standby=0x00
    ANL     ISPCR,#~ISPEN               ; Disable Function

    RET
```

C Code Example:

```
void page_p_sfr_write (void)
{
    IFADRH = 0x00;

    ISPCR = ISPEN;                      // Enable Function
    IFMT = MS2;                         // PageP_Write=0x04

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

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

(3). Required Function: Enable PWCTL0 for PCON0.PD control in Page-P

Assembly Code Example:

```
    MOV     IFADRL,#SPCON0          ;
    CALL    page_p_sfr_read         ;

    ORL     IFD,#PWCTL0             ; Set PWCTL0
    CALL    page_p_sfr_write        ;

    MOV     IFD,PCON0               ; Read PCON0

    ORL     IFD,#PD                 ; Write PCON0 and Power-Down
    MOV     IFADRL,#PCON0_P         ;
    CALL    page_p_sfr_write        ;
```

C Code Example:

```
    IFADRL = SPCON0;                //
    page_p_sfr_read();              //

    IFD |= PWCTL0;                  // Set PWCTL0
    page_p_sfr_write();             //

    IFD = PCON0;                    // Read PCON0

    IFD |= PD;                      // Write PCON0
    IFADRL = PCON0_P;               //
    page_p_sfr_write();             //
```

(4). Required Function: Enable CKCTL0 for SYSCLK divider (CKCON0) changed in Page-P

Assembly Code Example:

```
    MOV     IFADRL,#SPCON0          ;
    CALL    page_p_sfr_read         ;

    ORL     IFD,#CKCTL0             ; Set CKCTL0
    CALL    page_p_sfr_write        ;

    MOV     IFD,CKCON0              ; Read CKCON0

    ORL     IFD,#(AFS | SCKS0)      ; Write CKCON0 and Set AFS
    MOV     IFADRL,#CKCON0_P        ; SYSCLK / 2
    CALL    page_p_sfr_write
```

C Code Example:

```
    IFADRL = SPCON0;                //
    page_p_sfr_read ();             //

    IFD |= CKCTL0;                  // Set CKCTL
    page_p_sfr_write();             //

    IFD = CKCON0;                   // read CKCON0

    IFD |= (AFS | SCKS0);           //
    IFADRL = CKCON0_P;              //
    page_p_sfr_write();             // Write CKCON0
```

24. Auxiliary SFRs

AUXR0: Auxiliary Register 0

SFR Page = Normal
SFR Address = 0xA1 RESET = 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P4.0 function configured control bit 1 and 0. The two bits only act when IHRCO is selected for system clock source. In crystal mode, XTAL2 and XTAL1 are the alternated function of P4.0 and P4.1. In external clock input mode, P4.0 is the dedicated clock input pin. In IHRCO operating, P4.0 provides the following selections for GPIO or clock source generator. When P40OC[1:0] index to non-P4.0 GPIO function, P4.0 will drive the IHRCO output to provide the clock source for other devices.

P40OC[1:0]	P4.0 function	I/O mode
00	P4.0	By P4M0.0
01	IHRCO	By P4M0.0
10	IHRCO/2	By P4M0.0
11	IHRCO/4	By P4M0.0

For clock-out on P4.0 function, it is recommended to set P4M0.0 to “1” which selects P4.0 as push-push output mode.

Bit 5: P40FD, P4.0 Fast Driving.
0: P4.0 output with default driving.
1: P4.0 output with fast driving enabled. If P4.0 is configured to clock output, enable this bit when P4.0 output frequency is more than 12MHz at 5V application or more than 6MHz at 3V application.

Bit 4: T0XL is the Timer 0 per-scaler control bit. Please refer T0X12 (AUXR2.2) for T0XL function definition.

Bit 3~2: P1.4 and P1.5 alternated function selection.

P1FS[1:0]	P1.4	P1.5
00	P1.4	P1.5
01	Input for RXD0	Output for TXD0
10	Input for nINT0	Input for nINT1
11	Input for T0 or Output for T0CKO	Input for T1 or Output for T1CKO

Bit 1: INT1H, INT1 High/Rising trigger enable.
0: Remain INT1 triggered on low level or falling edge on nINT1 port pin.
1: Set INT1 triggered on high level or rising edge on nINT1 port pin.

Bit 0: INT0H, INT0 High/Rising trigger enable.
0: Remain INT0 triggered on low level or falling edge on nINT0 port pin.
1: Set INT0 triggered on high level or rising edge on nINT0 port pin.

AUXR1: Auxiliary Control Register 1

SFR Page = Normal
SFR Address = 0xA2 RESET = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	BPOC1	BPOC0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P3TWI, TWI interface in P3.0~P3.1.
0: Disable TWI function moved to P30 & P31.
1: Set TWI function in P3 as following definition.
‘TWI_SCL’ function in P3.3 is moved to P3.0.
‘TWI_SDA’ function in P3.4 is moved to P3.1.

Bit 6: P4S0MI, S0MI of Serial Port SPI master mode input in P4.1.
0: Disable S0MI function moved to P4.
1: Set S0MI in P4.1 as following definition.
'S0MI' function in P3.7 is moved to P4.1.

Bit 5: P2PCA, all PCA signals Port 2.
0: Disable PCA function moved to P4.
1: Set PCA in Port 2 as following definition.
'ECI' function in P3.4 is moved to P2.7.
'CEX0' function in P3.7 is moved to P2.6.
'CEX1' function in P3.5 is moved to P2.5.
'CEX2' function in P2.0 is kept in P2.0.
'CEX3' function in P2.4 is kept in P2.4.

Bit 1: XTOR, Crystal Oscillating Ready. Read Only.
0: Crystal Oscillating not Ready.
1: Crystal Oscillating Ready. When XTALE is enabled, XTOR reports the crystal oscillator reached start-up count.

Bit 3: STAF, Start Flag detection of TWI.
0: Clear by firmware by writing "0" on it.
1: Set by hardware to indicate the START condition occurred on TWI bus.

Bit 2: STOF, Stop Flag detection of TWI.
0: Clear by firmware by writing "0" on it.
1: Set by hardware to indicate the START condition occurred on TWI bus.

Bit 1~0: BPOC1~0, Beeper output control bits.

BPOC[1:0]	P1.6 function	I/O mode
00	P1.6	By P1M0.6
01	ILRCO/64	By P1M0.6
10	ILRCO/32	By P1M0.6
11	ILRCO/16	By P1M0.6

For beeper on P1.6 function, it is recommended to set P1M0.6 to "1" which selects P1.6 as push-push output mode.

AUXR2: Auxiliary Register 2

SFR Page = Normal
SFR Address = 0xA3 RESET = 0000-0000

7	6	5	4	3	2	1	0
--	BTI	URM0X3	SM3	T1X12	T0X12	T1CKOE	T0CKOE
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 AUXR2 is written.

Bit 6: BTI, Block TI in Serial Port Interrupt.
0: Retain the TI to be a source of Serial Port Interrupt.
1: Block TI to be a source of Serial Port Interrupt.

Bit 5: URM0X3, 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/4 as the baud rate for UART Mode 0.

Bit 4: SM3, Serial Port Mode control bit 3.
0: Disable Serial Prot Mode 4.
1: Enable SM3 to control Serial Port Mode 4, SPI Master.

Bit 3: 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 2: T0X12, Timer 0 clock source selector while C/T=0.

T0XL, T0X12	Timer 0 Clock Selection
0 0	SYSCLK/12
0 1	SYSCLK
1 0	SYSCLK/48
1 1	SYSCLK/192

Bit 1: T1CKOE, Timer 1 Clock Output Enable.
0: Disable Timer 1 clock output.
1: Enable Timer 1 clock output on P3.5.

Bit 0: T0CKOE, Timer 0 Clock Output Enable.
0: Disable Timer 0 clock output.
1: Enable Timer 0 clock output on P3.4.

BOREV: Bit Order Reversed Register

SFR Page = Normal
SFR Address = 0x96 RESET = 0000-0000

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

This register serves data read as a Bit-Order **Reversed** function with data written into.

25. 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, the "Megawin 8051 Writer U1" or the "Megawin 8051 ICE Adapter" (The ICE adapter also supports ICP programming function. Refer Section "26.4 ICP Interface Circuit"). After whole-chip erased, all the hardware options are left in "disabled" state and there is no ISP-memory and IAP-memory configured. The **MG86FE/L508** has the following Hardware Options:

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

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., 0x0FFF. The following table lists the ISP space option in this chip. In default setting, **MG86FE/L508** 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
3.5K bytes	0x1200
3K bytes	0x1400
2.5K bytes	0x1600
2K bytes	0x1800
1.5K bytes	0x1A00
1K bytes	0x1C00
0.5K bytes	0x1E00
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.

HWBS2:
☑: Enabled. Not only power-up but also any reset will cause MCU to boot from ISP-memory if ISP-memory is configured.
☐: Disabled. Where MCU boots from is determined by HWBS.

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

BO0REO:
☑: Enabled. BOD0 will trigger a RESET event to CPU on AP program start address. (2.2V)
☐: Disabled. BOD0 can not trigger a RESET to CPU.

WRENO:
☑: Enabled. Set WDTCR.WREN to enable a system reset function by WDTF.
☐: Disabled. Clear WDTCR.WREN to disable the system reset function by WDTF.

NSWDT: Non-Stopped WDT
☑: Enabled. Set WDTCR.NSW to enable the WDT running in power down mode (watch mode).
☐: Disabled. Clear WDTCR.NSW to disable the WDT running in power down mode (disable Watch mode).

HWENW: Hardware loaded for "ENW" of WDTCR.
☑: Enabled. Enable WDT and load the content of WRENO, NSWDT, 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.

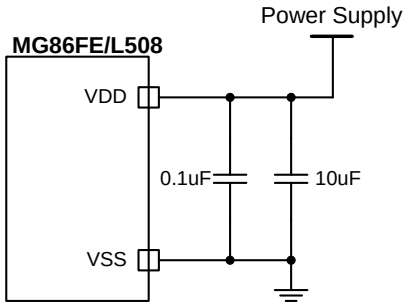
P36EN:
☑: Enabled. RSTIO (DCON0.1) will be cleared and P3.6 function behaves on RST pin.
☐: Disabled. RSTIO (DCON0.1) will be set and reserve RST pin function.

26. Application Notes

26.1. Power Supply Circuit

To have the **MG86FE/L508** work with power supply varying from 4.2V 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 26–1.

Figure 26–1. Power Supplied Circuit



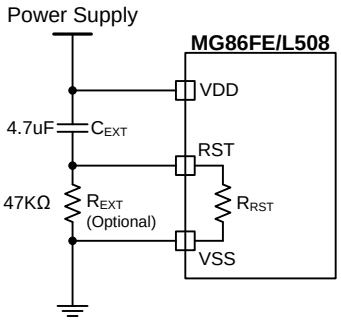
26.2. Reset Circuit

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 26–2 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 "27.2 DC Characteristics" for R_{RST} value.

Figure 26–2. Reset Circuit



26.3. XTAL Oscillating Circuit

To achieve successful and exact oscillating (up to 24MHz), the capacitors C1 and C2 are necessary, as shown in Figure 26–3. Normally, C1 and C2 have the same value. Table 26–1 lists the C1 & C2 value for the different frequency crystal application.

Figure 26–3. XTAL Oscillating Circuit

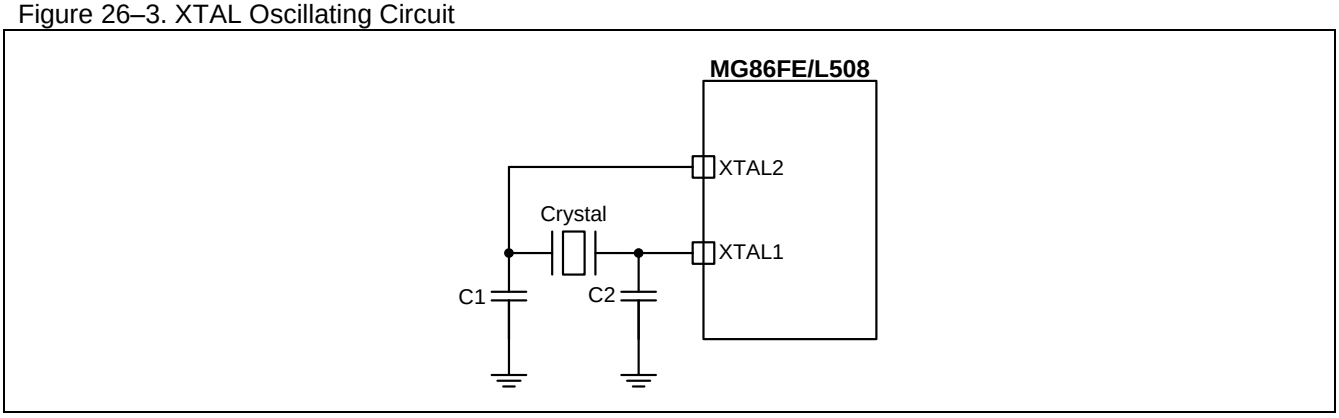


Table 26–1. Reference Capacitance of C1 & C2 for crystal oscillating circuit

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

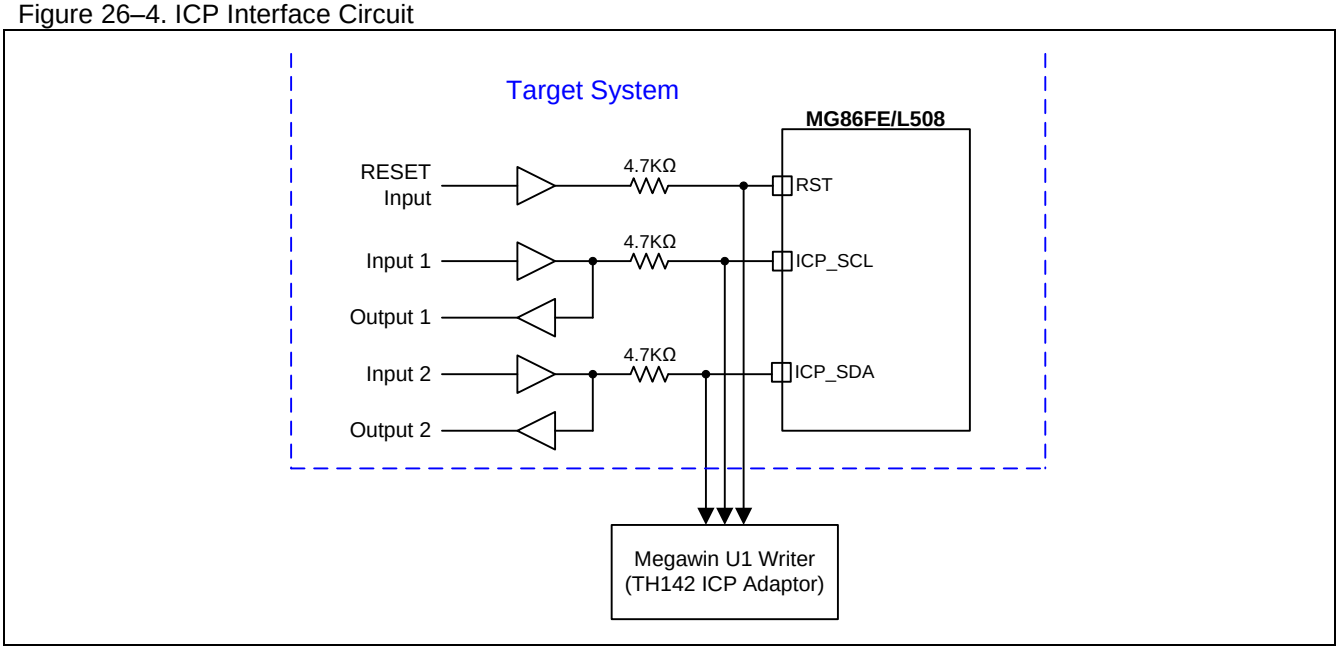
26.4. ICP Interface Circuit

MG86FE/L508 devices include an on-chip Megawin proprietary programming interface to allow In-Chip-Programming (ICP) with the production part installed in the end application. The ICP interface uses a clock signal (ICP_SCL) and a bi-directional data signal (ICP_SDA) to perform the device programming from a host instruction.

The ICP interface allows the ICP_SCL/ICP_SDA pins to be shared with user functions so that In-Chip Flash Programming function could be performed. This is practicable because ICP communication is performed when the device is in the halt state, where the on-chip peripherals and user software are stalled. In this halted state, the ICP interface can safely 'borrow' the ICP_SCL (P1.6) and ICP_SDA (P1.5) pins. In most applications, external resistors are required to isolate ICP interface traffic from the user application. A typical isolation configuration is shown in Figure 26–4.

It is strongly recommended to build the ICP interface circuit on target system. It will reserve the whole capability for software programming and device options configured.

Figure 26–4. ICP Interface Circuit



27. Electrical Characteristics

27.1. Absolute Maximum Rating

For MG86FE508:

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	200	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 MG86FL508:

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	200	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.

27.2. DC Characteristics

For MG86FE508:

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)	Except RST, P4.0, P4.1	2.0			V
V_{IH2}	Input High voltage (P4.0, P4.1)		3.5			V
V_{IH3}	Input High voltage (RST/P3.6)		4.0			
V_{IL1}	Input Low voltage (All I/O Ports)	Except RST, P4.0, P4.1			0.8	V
V_{IL2}	Input Low voltage (P4.0, P4.1)				1.0	V
V_{IL3}	Input Low voltage (RST/P3.6)				1.0	V
I_{IH}	Input High Leakage current (All I/O Ports)	V_{PIN} = VDD		0	10	uA
I_{IL1}	Logic 0 input current (P3 in quasi-mode or Input port with on-chip pull-up resistor)	V_{PIN} = 0.4V		28	60	uA
I_{IL2}	Logic 0 input current (All Input only or open-drain Ports)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	Logic 1 to 0 input transition current (P3 in quasi-mode or Input port with on-chip pull-up resistor)	V_{PIN} =1.8V		330	500	uA
I_{OH1}	Output High current (P3 in quasi-Mode)	V_{PIN} =2.4V	150	200		uA
I_{OH2}	Output High current (All push-pull output ports)	V_{PIN} =2.4V	12	29		mA
I_{OL1}	Output Low current (All I/O Ports)	V_{PIN} =0.4V	12	20		mA
R_{RST}	Internal reset pull-down resistance			77		Kohm
Power Consumption						
I_{OP1}	Normal mode operating current	SYSCLK = 24MHz @ IHRCO		11		mA
I_{OP2}		SYSCLK = 12MHz @ IHRCO		6.8		mA
I_{OP3}		SYSCLK = 6MHz @ IHRCO & HSE = 0		4.8		mA
I_{OP4}		SYSCLK = 3MHz @ IHRCO & HSE = 0		2.9		mA
I_{OP5}		SYSCLK = 24MHz @ XTAL		11.5		mA
I_{OP6}		SYSCLK = 12MHz @ XTAL		7.2		mA
I_{OP7}		SYSCLK = 6MHz @ XTAL & HSE = 0		5		mA
I_{OP8}		SYSCLK = 2MHz @ XTAL & HSE = 0		2.6		mA
I_{OPS1}	Slow mode operating current	SYSCLK = 24MHz/128 @ IHRCO & HSE = 0		1.1		mA
I_{IDLE1}	Idle mode operating current	SYSCLK = 24MHz @ IHRCO		4		mA
I_{IDLE2}		SYSCLK = 24MHz @ XTAL		4.9		mA
I_{IDLE3}		SYSCLK = 24MHz/128 @ IHRCO		1		mA
I_{IDLE4}		SYSCLK = 24MHz/128 @ XTAL		1.9		mA
I_{IDLE5}		SYSCLK = 64KHz @ ILRCO		22		uA

I_{IDLE6}		SYSCLK = 64KHz/128 @ ILRCO		13		uA
I_{SUB1}	Sub-clock mode operating current	SYSCLK = 64KHz @ ILRCO & HSE = 0		60		uA
I_{SUB2}		SYSCLK = 64KHz/128 @ ILRCO & HSE = 0		13		uA
I_{SUB2}		SYSCLK = 32768Hz @ XTAL & HSE = 0		58		uA
I_{WAT}	Watch mode operating current	WDT = 64KHz @ ILRCO in PD mode		2.5		uA
I_{MON1}	Monitor Mode operating current	BOD0 enabled in PD mode		10		uA
I_{RTC1}	RTC Mode operating current	RTC operating in PD mode		4.7		uA
I_{PD1}	Power down mode current			0.1	5	uA
BOD0 Characteristics						
V_{BOD0}	BOD0 detection level	T_A = -40℃ to +85℃	4.0(1)	4.2	4.4(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 MG86FL508:

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)	Except RST, P4.0, P4.1	2.0			V
V_{IH2}	Input High voltage (P4.0, P4.1)		2.4			V
V_{IH3}	Input High voltage (RST/P3.6)		2.7			V
V_{IL1}	Input Low voltage (All I/O Ports)	Except RST, P4.0, P4.1			0.8	V
V_{IL2}	Input Low voltage (P4.0, P4.1)				0.8	V
V_{IL3}	Input Low voltage (RST/P3.6)				0.8	V
I_{IH}	Input High Leakage current (All I/O Ports)	V_{PIN} = VDD		0	10	uA
I_{IL1}	Logic 0 input current (P3 in quasi-mode or Input port with on-chip pull-up resistor)	V_{PIN} = 0.4V		10	30	uA
I_{IL2}	Logic 0 input current (All Input only or open-drain Ports)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	Logic 1 to 0 input transition current (P3 in quasi-mode or Input port with on-chip pull-up resistor)	V_{PIN} =1.8V		120	250	uA
I_{OH1}	Output High current (P3 in quasi-Mode)	V_{PIN} =2.4V	40	80		uA
I_{OH2}	Output High current (All push-pull output ports)	V_{PIN} =2.4V	8	10		mA
I_{OL1}	Output Low current (All I/O Ports)	V_{PIN} =0.4V	8	12		mA
R_{RST}	Internal reset pull-down resistance			93		Kohm
Power Consumption						
I_{OP1}	Normal mode operating current	SYSCLK = 24MHz @ IHRCO		11		mA
I_{OP2}		SYSCLK = 12MHz @ IHRCO		6.8		mA
I_{OP3}		SYSCLK = 6MHz @ IHRCO & HSE = 0		4.8		mA
I_{OP4}		SYSCLK = 3MHz @ IHRCO & HSE = 0		2.9		mA
I_{OP5}		SYSCLK = 24MHz @ XTAL		11		mA
I_{OP6}		SYSCLK = 12MHz @ XTAL		6.7		mA
I_{OP7}		SYSCLK = 6MHz @ XTAL & HSE = 0		4.3		mA
I_{OP8}		SYSCLK = 2MHz @ XTAL & HSE = 0		1.6		mA
I_{OPS1}	Slow mode operating current	SYSCLK = 24MHz/128 @ IHRCO & HSE = 0		1.1		mA
I_{IDLE1}	Idle mode operating current	SYSCLK = 24MHz @ IHRCO		4		mA
I_{IDLE2}		SYSCLK = 24MHz @ XTAL		3.8		mA
I_{IDLE3}		SYSCLK = 24MHz/128 @ IHRCO		1		mA
I_{IDLE4}		SYSCLK = 24MHz/128 @ XTAL		0.75		mA
I_{IDLE5}		SYSCLK = 64KHz @ ILRCO		22		uA
I_{IDLE6}		SYSCLK = 64KHz/128 @ ILRCO		13		uA

I_{SUB1}	Sub-clock mode operating current	SYSCLK = 64KHz @ ILRCO & HSE = 0		60		uA
I_{SUB2}		SYSCLK = 64KHz/128 @ ILRCO & HSE = 0		13		uA
I_{SUB3}		SYSCLK = 32768Hz @ XTAL & HSE = 0		60		uA
I_{WAT}	Watch mode operating current	WDT = 64KHz @ ILRCO in PD mode		2.5		uA
I_{MON1}	Monitor Mode operating current	BOD0 enabled in PD mode		10		uA
I_{RTC1}	RTC Mode operating current	RTC operating in PD mode		1.8		uA
I_{PD1}	Power down mode current			0.1	5	uA
BOD0 Characteristics						
V_{BOD0}	BOD0 detection level	T_A = -40℃ to +85℃	2.45(1)	2.6	2.75(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.

27.3. External Clock Characteristics

For MG86FE508:

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

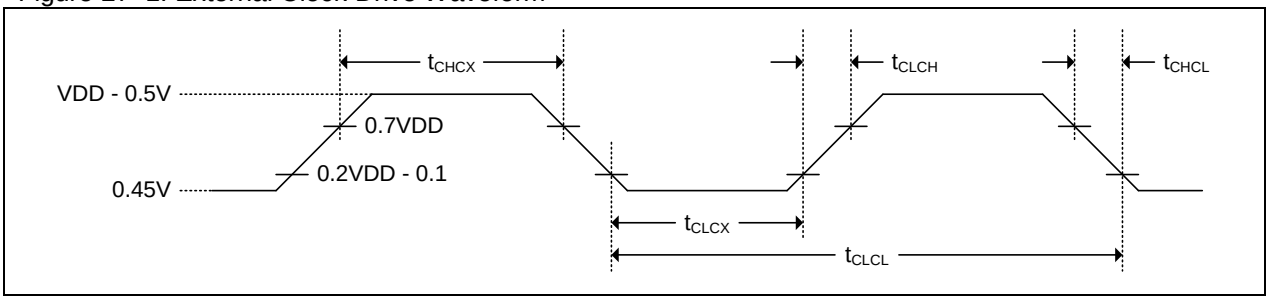
Symbol	Parameter	Oscillator				Unit
		Crystal Mode		ECKI Mode		
		Min.	Max	Min.	Max	
$1/t_{CLCL}$	Oscillator Frequency	2	25	0	25	MHz
$1/t_{CLCL}$	Oscillator Frequency (VDD = 4.2V ~ 5.5V)	2	12	0	12	MHz
t_{CLCL}	Clock Period	40		40		ns
t_{CHCX}	High Time	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCX}	Low Time	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCH}	Rise Time		5		5	ns
t_{CHCL}	Fall Time		5		5	ns

For MG86FL508:

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

Symbol	Parameter	Oscillator				Unit
		Crystal Mode		ECKI Mode		
		Min.	Max	Min.	Max	
$1/t_{CLCL}$	Oscillator Frequency	2	25	0	25	MHz
$1/t_{CLCL}$	Oscillator Frequency (VDD = 2.4V ~ 3.6V)	2	12	0	12	MHz
t_{CLCL}	Clock Period	40		40		ns
t_{CHCX}	High Time	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCX}	Low Time	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCH}	Rise Time		5		5	ns
t_{CHCL}	Fall Time		5		5	ns

Figure 27–1. External Clock Drive Waveform



27.4. IHRCO Characteristics

For MG86FE508:

Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Voltage		4.5		5.5	V
IHRCO Frequency	TA = +25℃, AFS = 0		24		MHz
	TA = +25℃, AFS = 1		22.118		MHz
IHRCO Frequency Deviation (factory calibrated)	TA = +25℃	-1.0		+1.0	%
	TA = -40℃ to +85℃	-2.5(1)		+2.5(1)	%
IHRCO Start-up Time	TA = -40℃ to +85℃		1(1)	32(1)	us
IHRCO Power Consumption	TA = +25℃, VDD=5.0V		770		uA

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

For MG86FL508:

Parameter	Test Condition	Limits			Unit
		min	Typ	max	
Supply Voltage		2.7		3.6	V
IHRCO Frequency	TA = +25℃, AFS = 0		24		MHz
	TA = +25℃, AFS = 1		22.118		MHz
IHRCO Frequency Deviation (factory calibrated)	TA = +25℃	-1.0		+1.0	%
	TA = -40℃ to +85℃	-2.5(1)		+2.5(1)	%
IHRCO Start-up Time	TA = -40℃ to +85℃		1(1)	32(1)	us
IHRCO Power Consumption	TA = +25℃, VDD=3.3V		770		uA

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

27.5. ILRCO Characteristics

For MG86FE508:

Parameter	Test Condition	Limits			Unit
		min	Typ	max	
Supply Voltage		4.2		5.5	V
ILRCO Frequency	TA = +25℃		64		KHz
ILRCO Frequency Deviation	TA = +25℃	-30(1)		+30(1)	%
	TA = -40℃ to +85℃	-50(1)		+50(1)	%
ILRCO Power Consumption	TA = +25℃, VDD=5.0V		2		uA

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

For MG86FL508:

Parameter	Test Condition	Limits			Unit
		min	Typ	max	
Supply Voltage		2.4		3.6	V
ILRCO Frequency	TA = +25℃		64		KHz
ILRCO Frequency Deviation	TA = +25℃	-30(1)		+30(1)	%
	TA = -40℃ to +85℃	-50(1)		+50(1)	%
IHRCO Power Consumption	TA = +25℃, VDD=3.3V		2		uA

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

27.6. Flash Characteristics

For MG86FE508:

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℃	100			times
Flash Data Retention	TA = +25℃	100			year

For MG86FL508:

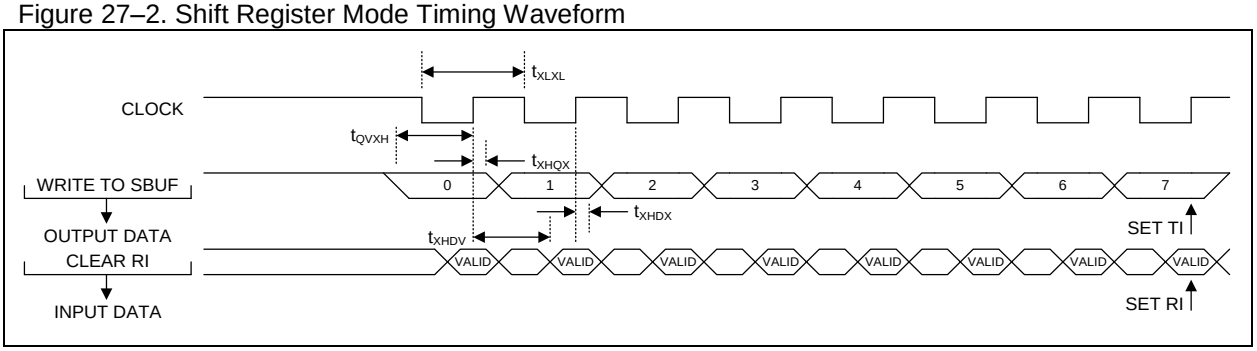
Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Voltage	TA = -40℃ to +85℃	2.4		3.6	V
Flash Write (Erase/Program) Voltage	TA = -40℃ to +85℃	2.7		3.6	V
Flash Erase/Program Cycle	TA = -40℃ to +85℃	100			times
Flash Data Retention	TA = +25℃	100			year

27.7. Serial Port Timing Characteristics

For MG86FE/L508 E-Type: VDD = 5.0V±10%, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified
For MG86FE/L508 L-Type: 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 27–2. Shift Register Mode Timing Waveform



27.8. ADC Characteristics

For MG86FE508:

VDD = 5.0V, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Range					
Supply Voltage		4.2		5.5	V
DC Accuracy					
Resolution			8		bits
Integral Nonlinearity	VDD= 5.0V (200 ksps)		±0.5	±1	LSB
	VDD= 4.2V~5.5V (200 ksps)		±1	±1.5	LSB
	VDD= 4.2V~5.5V (25 ksps)		±1	±1.5	LSB
Differential Nonlinearity	VDD= 4.2V ~ 5.5V		±0.5	±1	LSB
Offset Error	VDD= 4.2V ~ 5.5V			±1	LSB
Conversion Rate					
SAR Conversion Clock				12	MHz
Conversion Time in SAR Clocks			60		clocks
Throughput Rate				200	ksps
Analog Inputs					
ADC Input Voltage Range		0		VDD	V
Input Capacitance			1		pF
Power Consumption					
Power Supply Current			1		mA

For MG86FL508:

VDD = 3.3V, VSS = 0V, T_A = -40℃ to +85℃, unless otherwise specified

Parameter	Test Condition	Limits			Unit
		min	typ	max	
Supply Range					
Supply Voltage		2.4		3.6	V
DC Accuracy					
Resolution			8		bits
Integral Nonlinearity	VDD= 3.0V~3.6V (200 ksps)		±1.5	±2	LSB
	VDD= 2.7V~3.6V (200 ksps)		±2.5	±3	LSB
	VDD= 2.4V~3.6V (200 ksps)		±8	±10	LSB
	VDD= 3.0V~3.6V (25 ksps)		±1	±1.5	LSB
	VDD= 2.7V~3.6V (25 ksps)		±1	±1.5	LSB
	VDD= 2.4V~3.6V (25 ksps)		±3	±4	LSB
Differential Nonlinearity	VDD= 3.0V~3.6V (200 ksps)		±0.5	±1	LSB
	VDD= 2.7V~3.6V (200 ksps)		±1.5	±2	LSB
	VDD= 2.4V~3.6V (200 ksps)		±8	±10	LSB
	VDD= 3.0V~3.6V (25 ksps)		±0.5	±1	LSB
	VDD= 2.7V~3.6V (25 ksps)		±0.5	±1	LSB
	VDD= 2.4V~3.6V (25 ksps)		±3	±4	LSB
Offset Error	VDD= 2.4V ~ 3.6V			±1	LSB
Conversion Rate					
SAR Conversion Clock				12	MHz
Conversion Time in SAR Clocks			60		clocks
Throughput Rate				200	ksps
Analog Inputs					
ADC Input Voltage Range		0		VDD	V
Input Capacitance			1		pF
Power Consumption					
Power Supply Current			0.8		mA

28. Instruction Set

Table 28–1. 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	Not Support
MOVX A,@DPTR	Move on-chip auxiliary RAM(16-bit address) to Acc	1	Not Support
MOVX @Ri,A	Move Acc to on-chip auxiliary RAM(8-bit address)	1	Not Support
MOVX @DPTR,A	Move Acc to on-chip auxiliary RAM(16-bit address)	1	Not Support
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

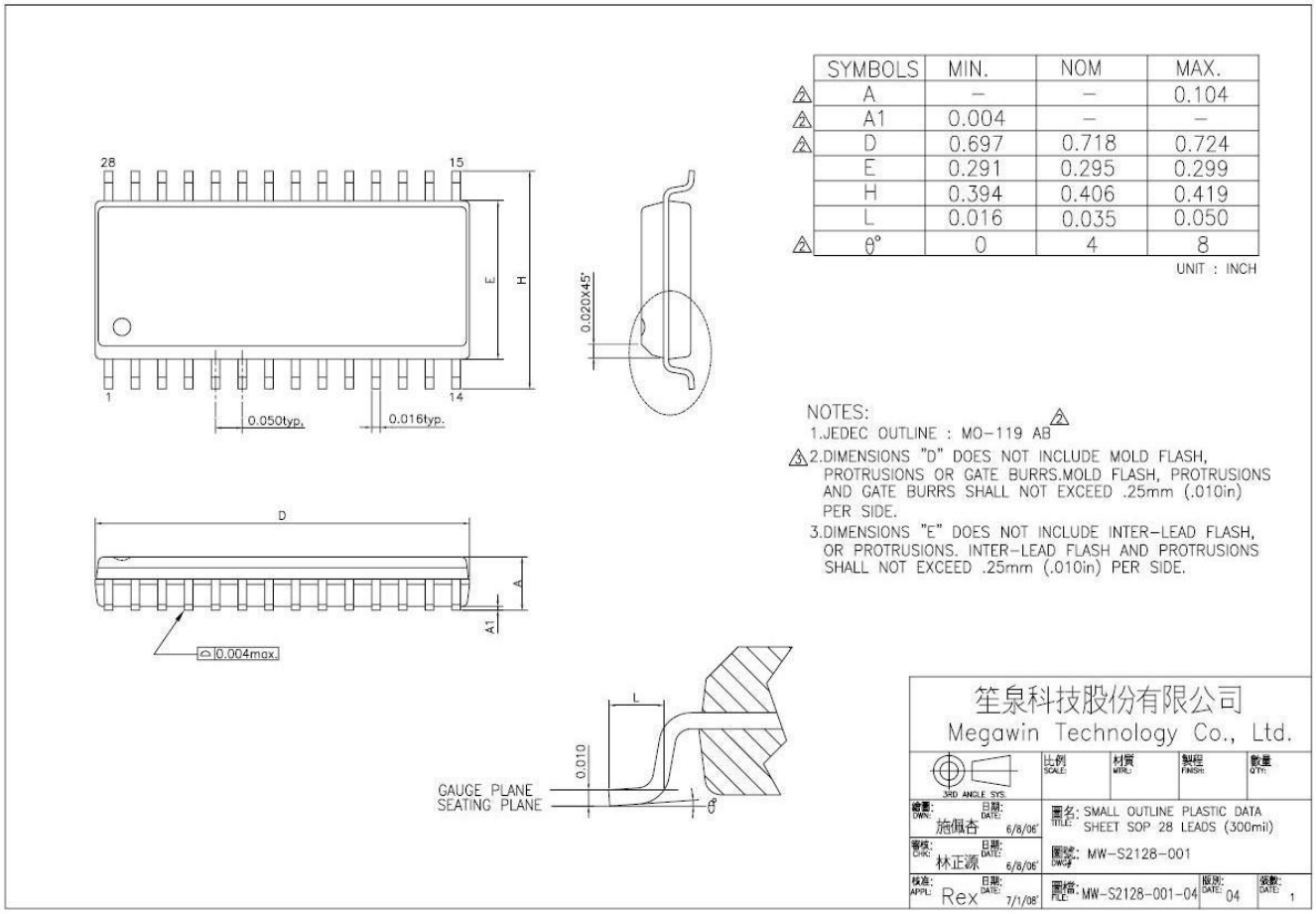
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	3
SUBB A,#data	Subtract immediate data from Acc with borrow	2	2
INC A	Increment Acc	1	2
INC Rn	Increment register	1	3
INC direct	Increment direct byte	2	4
INC @Ri	Increment indirect RAM	1	4
DEC A	Decrement Acc	1	2
DEC Rn	Decrement register	1	3
DEC 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

29. Package Dimension

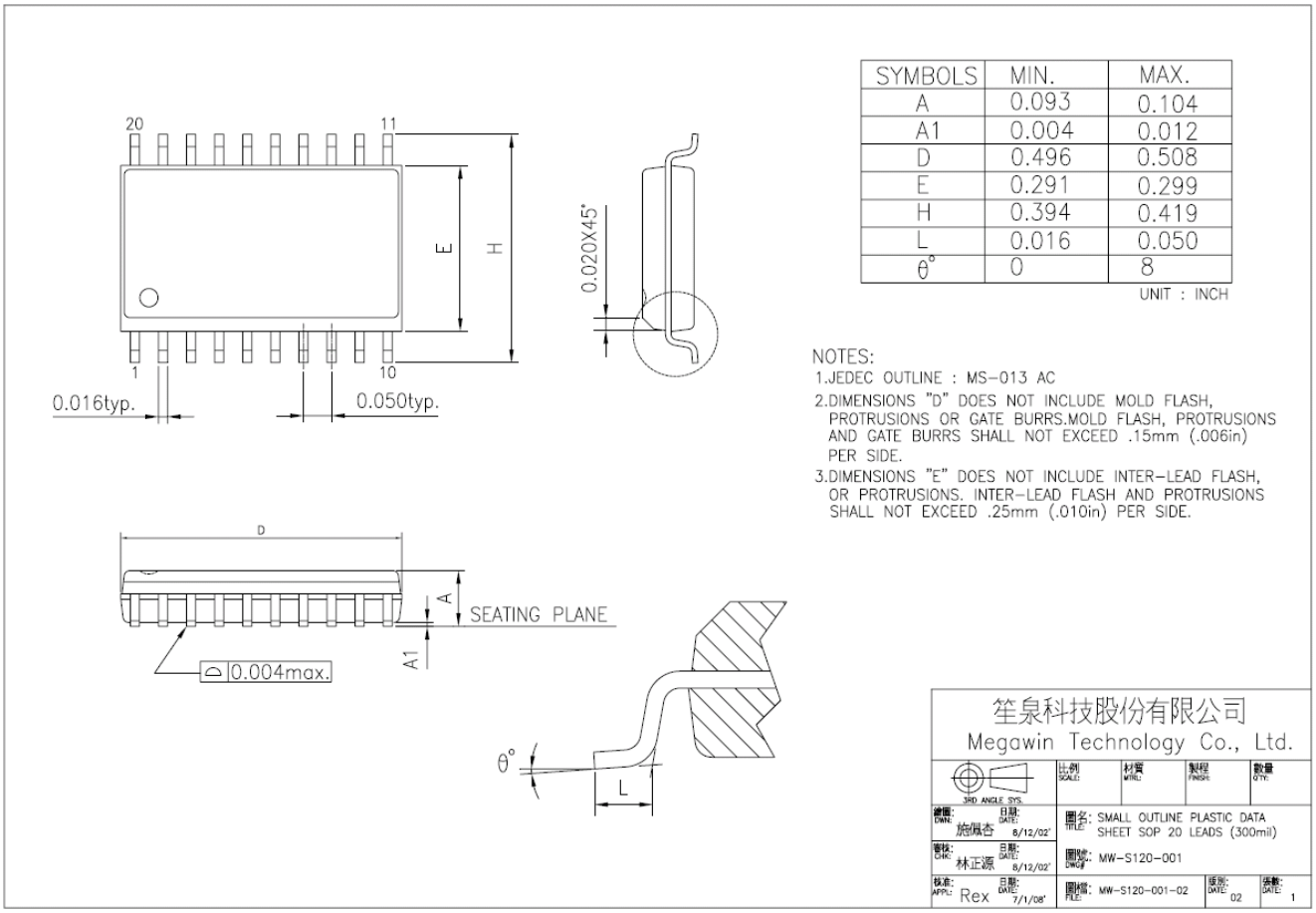
29.1. SOP-28

Figure 29–1. SOP-28



29.2. SOP-20

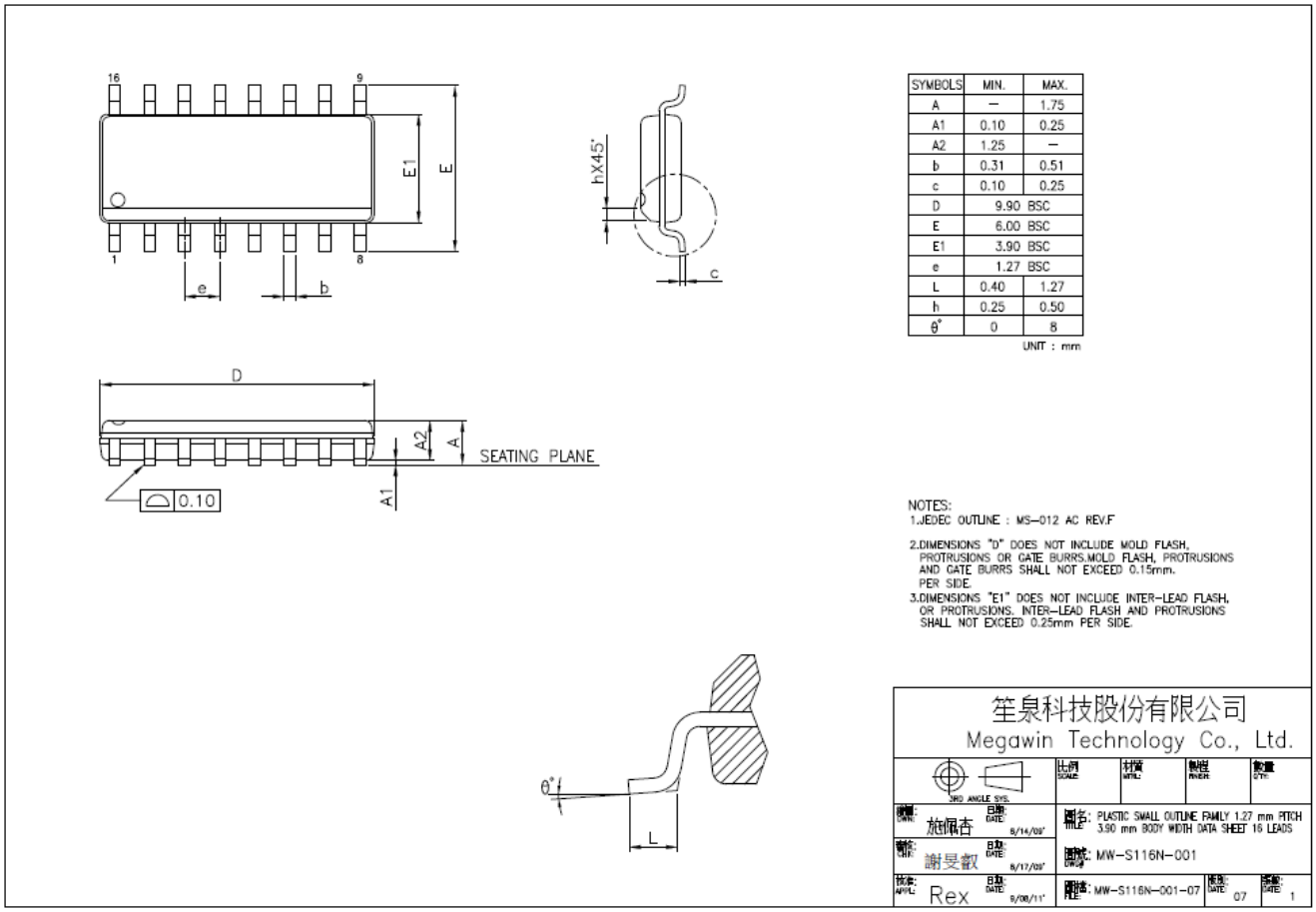
Figure 29–2. SOP-20



SYMBOLS	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.496	0.508
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ°	0	8

29.3. SOP-16

Figure 29–3. SOP-16



SYMBOLS	MIN.	MAX.
A	—	1.75
A1	0.10	0.25
A2	1.25	—
b	0.31	0.51
c	0.10	0.25
D	9.90 BSC	
E	6.00 BSC	
E1	3.90 BSC	
e	1.27 BSC	
L	0.40	1.27
h	0.25	0.50
θ°	0	8

UNIT : mm

NOTES:
1.JEDEC OUTLINE : MS-012 AC REV.F

2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED 0.15mm. PER SIDE.

3.DIMENSIONS "E1" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED 0.25mm PER SIDE.

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

3RD ANGLE SYS.

比例 SCALE:	材質 MTRL:	版號 REVISION:	數量 QTY:
繪圖: 施佩杏 日期: 8/14/09'	圖名: PLASTIC SMALL OUTLINE FAMILY 1.27 mm PITCH 3.90 mm BODY WIDTH DATA SHEET 16 LEADS		
審核: 謝旻叡 日期: 8/17/09'	圖號: MW-S116N-001		
核准: Rex 日期: 9/08/11'	檔名: MW-S116N-001-07	版次: DATE: 07	張次: DATE: 1

30. Revision History

Table 30–1. Revision History

Rev	Descriptions	Date
v0.70	1. Release preliminary version, v0.70.	2012/07/16
v0.80	1. In feature section, modify PCA, BOD & Clock Source description.	2012/08/22
	2. In feature section, add P2 in GPIO description.	2012/08/22
	3. In System Block Diagram, add PWM channel number in PCA.	2012/08/22
	4. In System Clock section, finalize XTOR function and add DON0.HSE description in Section 8.2, Clock Register. And modify IHRCO performance description which is referred to IHRCO characteristics.	2012/08/22
	5. Modify Figure 7-1 for IHRCO output on P4.0.	2012/08/22
	6. In RTCTM register description, modify P6.0 to P4.0.	2012/08/22
	7. Modify slowest speed of sub-clock mode from 250Hz to 500Hz.	2012/08/22
	8. Modify WDT diagram.	2012/08/29
	9. Finalize T0XL function for Timer 0 clock pre-scaler.	2012/08/29
	10. Modify Timer 0/1 Mode 0/1 description and Timer 0 clock output function for T0XL.	2012/08/29
	11. In section 15.1.6, modify TMOD register for Mode 0, PWM generator.	2012/08/29
	12. Add description for clock polarity selection of UART Mode 0.	2012/08/29
	13. Modify Figure 15-4 & 15-5 for UART Mode 0 clock polarity selection.	2012/08/29
	14. Add CPS2 to select SYSCLK/1 to PCA module.	2012/08/29
	15. Add PAOE description and modify Section ISP/IAP description.	2012/08/29
	16. Add port pin input mode defined in KBI description.	2012/08/29
	17. Add Serial-Interface-Detection and Beeper function description.	2012/08/29
	18. Update Figure 21-1 and Section 26, Electrical Characterisitcs	2012/08/29
	19. Modify system flag interrupt description in Section 14.2.	2012/08/29
	20. Add sample code for each function.	2012/08/29
A1	New Datasheet format New edition	2013/05/21
A1.1	Modify MTP write access cycle.	2013/09/04
A1.2	Modify IE SFR Address error E8h as A8h.	2013/10/09
A1.3	Support SOP16 Package	2015/05/27

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).