# MA801/MA802
# 说明书

版本: A10

This document contains information on a new product under development by Megawin. Megawin reserves the right to change or discontinue this product without notice.
© Megawin Technology Co., Ltd. 2005 All rights reserved.

# 特性

- 1-T 80C51 中央处理器
- **MA801_MA802 8K** 字节 Flash ROM
  - ISP 存储空间可以选择 1.0KB, 2.0KB 或 3.0KB
  - 用于 Flash 存储器访问的两级代码保护
  - Flash 写/擦除周期: 20,000
  - Flash 数据保留时间: 在 25℃下 100 年
  - MA801/MA802 出厂默认空间设置
    - AP 程序空间（0000h~17FFh）
    - IAP 数据空间（1800h~1BFFh）
    - ISP 引导码空间（1C00h~1FFFh）（保留在线烧录用，用户程序无法更动）
- 片上 256 字节暂存 RAM
- 中断控制器
  - 7 个中断源，4 级优先级
  - 两个外部中断输入，INT0 和 INT1
- 两个 16 位定时/计数器， Timer 0 和 Timer 1.
  - Timer 0/1 可以选择 X12 模式
- 可编程的 16 位计数/定时阵列(PCA) 支持 2 通道 PWM
  - 捕获模式
  - 16 位软件定时器模式
  - 高速输出模式
  - 8 位 PWM 模式
- 增强型 UART (S0)
  - 帧错误检测
  - 自动地址识别
- 8 位 ADC
  - 可编程吞吐率达到 100 ksps
  - 8 通道单端输入
- 主/从 SPI 串行接口
- 可编程看门狗定时器，由 CPU 或上电一次使能。
- 在 20 脚封装下最多 15 个 GPIO。
  - 可以配置成双向口，推挽输出，漏极开路和仅输入
- 多种电源控制模式：空闲模式和掉电模式
  - 所有中断可以唤醒空闲模式
  - **2** 个源可以唤醒掉电模式
- 低压侦测器：对于 MA801 是 VDD 3.7V ，对于 MA802 是 VDD 2.4V
- 工作电压范围 Operating voltage range:
  - **MA801**: 4.5V~5.5V, 要求 Flash 写操作最小是 4.5V (ISP/IAP)
  - **MA802**: 2.4V~3.6V, 要求 Flash 写操作最小是 2.7V (ISP/IAP)
- 工作频率范围: 25MHz(最大)
  - **MA801**: 0 – 25MHz @ 4.5V – 5.5V
  - **MA802**: 0 – 12MHz @ 2.4V – 3.6V and 0 – 25MHz @ 2.7V – 3.6V
- 时钟源
  - 外部晶振模式和内置 RC 振荡器 (IRCO, 6MHz)
- 工作温度:
  - 工业级 (-40℃ to +85℃)*
- 封装类型:
  - PDIP20: MA801AE/MA802AE
  - SOP20: MA801AS/MA802AS
  - TSSOP20: MA801AT/MA802AT

*: 抽样检测

# 目录


特性 ..... 3
目录 ..... 6
1. 概述 ..... 10
2. 方框图 ..... 11
3. 特殊功能寄存器 ..... 12
3.1. SFR 地址表 ..... 12
3.2. SFR 位分配表 ..... 13
4. 引脚结构 ..... 15
4.1. 封装指南 ..... 15
4.2. 引脚描述 ..... 16
5. 8051 CPU 功能描述 ..... 17
5.1. CPU 寄存器 ..... 17
5.2. CPU 时序 ..... 18
5.3. CPU 寻址模式 ..... 19
6. 存储器组织 ..... 20
6.1. 片上程序存储器 ..... 20
6.2. 片上数据存储器 ..... 21
6.3. 关于 C51 编译器的声明标识符 ..... 23
7. 数据指针寄存器(DPTR) ..... 24
8. 系统时钟 ..... 25
8.1. 时钟结构 ..... 25
8.2. 时钟寄存器 ..... 26
8.3. 时钟示例代码 ..... 27
9. 看门狗定时器 (WDT) ..... 28
9.1. WDT 结构 ..... 28
9.2. WDT 在掉电模式和空闲模式期间 ..... 28
9.3. WDT 寄存器 ..... 29
9.4. WDT 硬件选项 ..... 30
9.5. WDT 示例代码 ..... 31
10. 系统复位 ..... 32
10.1. 复位源 ..... 32
10.2. 上电复位 ..... 32
10.3. 外部复位 ..... 33
10.4. 软件复位 ..... 33
10.5. 低电压复位 ..... 33
10.6. WDT 复位 ..... 34
10.7. 非法地址复位 ..... 34
10.8. 复位示例代码 ..... 35
11. 电源管理 ..... 36
11.1. 低电压检测器 ..... 36
11.2. 省电模式 ..... 37
11.3. 空闲模式 ..... 37

11.4. 掉电模式 ..... 37
11.4.1. 中断唤醒掉电模式 ..... 38
11.4.2. 复位唤醒掉电模式 ..... 38
11.5. 电源控制寄存器 ..... 39
11.6. 电源控制示例代码 ..... 40
12.I/O 端口配置 ..... 41
12.1. IO 结构 ..... 41
12.2. 准双向口结构 ..... 41
12.3. 推挽输出结构 ..... 42
12.4. 仅输入（高阻抗输入）模式结构 ..... 42
12.5. 3 漏极开路输出结构 ..... 43
12.6. I/O 端口寄存器 ..... 44
12.7. 端口 1 寄存器 ..... 44
12.8. 端口 3 寄存器 ..... 44
12.9. GPIO 示例代码 ..... 46
13.中断 ..... 47
13.1. 中断结构 ..... 47
13.2. 中断源 ..... 49
13.3. 中断使能 ..... 50
13.4. 中断优先级 ..... 50
13.5. 中断处理 ..... 51
13.6. 中断寄存器 ..... 52
13.7. 中断示例代码 ..... 54
14.定时器/计数器 ..... 55
14.1. 定时器 0 和定时器 1 ..... 55
14.2. 模式 0 结构 ..... 55
14.2.1. 模式 1 结构 ..... 56
14.2.2. 模式 2 结构 ..... 57
14.2.3. 模式 3 结构 ..... 58
14.2.4. 定时器 0/1 寄存器 ..... 59
14.2.5. 定时器 0/1 示例代码 ..... 61
15.串行口(UART) ..... 63
15.1. 串行口模式 0 ..... 64
15.2. 串行口模式 1 ..... 66
15.3. 串行口模式 2 和模式 3 ..... 67
15.4. 帧错误检测 ..... 67
15.5. 多处理器通讯 ..... 68
15.6. 自动地址识别 ..... 68
15.7. 波特率设置 ..... 70
15.7.1. 模式 0 的波特率 ..... 70
15.7.2. 模式 2 的波特率 ..... 70
15.7.3. 模式 1 和 3 的波特率 ..... 70
15.8. 串行口寄存器 ..... 73
15.9. 串行口示例代码 ..... 75
16.可编程计数器阵列(PCA) ..... 77
16.1. PCA 概述 ..... 77
16.2. PCA 定时器/计数器 ..... 78
16.3. 比较/捕获模块 ..... 80

16.4. PCA 工作模式 ..... 82
16.4.1. 捕获模式 ..... 82
16.4.2. 16 位软件定时器模式 ..... 83
16.4.3. 高速输出模式 ..... 84
16.4.4. PWM 模式 ..... 85
16.5. PCA 示例代码 ..... 86
17. 串行外设接口 (SPI) ..... 87
17.1. 典型 SPI 配置 ..... 88
17.1.1. 单主机和单从机 ..... 88
17.1.2. 双驱动器,可以是主机或从机 ..... 88
17.1.3. 单主机和多从机 ..... 89
17.2. SPI 配置 ..... 90
17.2.1. 从机注意事项 ..... 90
17.2.2. 主机注意事项 ..... 90
17.2.3. nSS 引脚的模式改变 ..... 91
17.2.4. 数据冲突 ..... 91
17.2.5. SPI 时钟频率选择 ..... 91
17.3. 数据模式 ..... 92
17.4. SPI 寄存器 ..... 94
17.5. SPI 示例代码 ..... 96
18. 8-位 ADC ..... 100
18.1. ADC 结构 ..... 100
18.2. ADC 工作 ..... 100
18.2.1. ADC 输入通道 ..... 101
18.2.2. 启动一个转换 ..... 101
18.2.3. ADC 转换时间 ..... 101
18.2.4. I/O 引脚用于 ADC 功能 ..... 101
18.2.5. 空闲和掉电模式 ..... 101
18.3. ADC 寄存器 ..... 102
18.4. ADC 示例代码 ..... 103
19. ISP 和 IAP ..... 105
19.1. MA801_MA802 Flash 存储空间配置 ..... 105
19.2. MA801_MA802 Flash 在 ISP/IAP 的访问 ..... 106
19.2.1. ISP/IAP Flash 页檫除模式 ..... 106
19.2.2. ISP/IAP Flash 写模式 ..... 108
19.2.3. ISP/IAP Flash 读取模式 ..... 110
19.3. ISP 操作 ..... 112
19.3.1. 硬件访问 ISP ..... 112
19.3.2. 软件访问 ISP ..... 113
19.3.3. ISP 注意事项 ..... 114
19.3.4. MA801_MA802 默认 ISP 代码 ..... 115
19.4. 在应用可编程(IAP) ..... 116
19.4.1. IAP-存储空间边界/范围 ..... 116
19.4.2. IAP-存储空间更新数据 ..... 116
19.4.3. IAP 注意事项 ..... 117
19.5. ISP/IAP 寄存器 ..... 118
19.6. ISP/IAP 示例代码 ..... 120
20. 辅助特殊功能寄存器 ..... 123
21. 硬件选项 ..... 124

22.应用说明 ..... 126
22.1. 电源电路 ..... 126
22.2. 复位电路 ..... 126
22.3. XTAL 振荡电路 ..... 127
22.4. ISP 接口电路 ..... 128
23.电气特性 ..... 129
23.1. 最大绝对额定参数 ..... 129
23.2. 直流电气特性 ..... 130
23.3. 外部时钟特征 ..... 132
23.4. IRCO 特征 ..... 133
23.5. Flash 特征 ..... 133
23.6. 串口时间特征 ..... 134
23.7. SPI 时间特性 ..... 135
24.指令集 ..... 137
25.封装尺寸 ..... 140
25.1. DIP-20 ..... 140
25.2. SOP-20 ..... 141
25.3. TSSOP-20 ..... 142
26.修订历史 ..... 143

# 1. 概述

**MA801_MA802** 是基于 80C51 的高效 1-T 结构的单芯片微处理器，执行指令需要要 1~6 个时钟信号（比标准的 8051 快 6~7 倍），与标准 8051 指令兼容。因此在与标准 8051 有同样的处理能力的情况下，**MA801_MA802** 只需要要非常低的运行速度，同时由此能很大程度的减少耗电量。

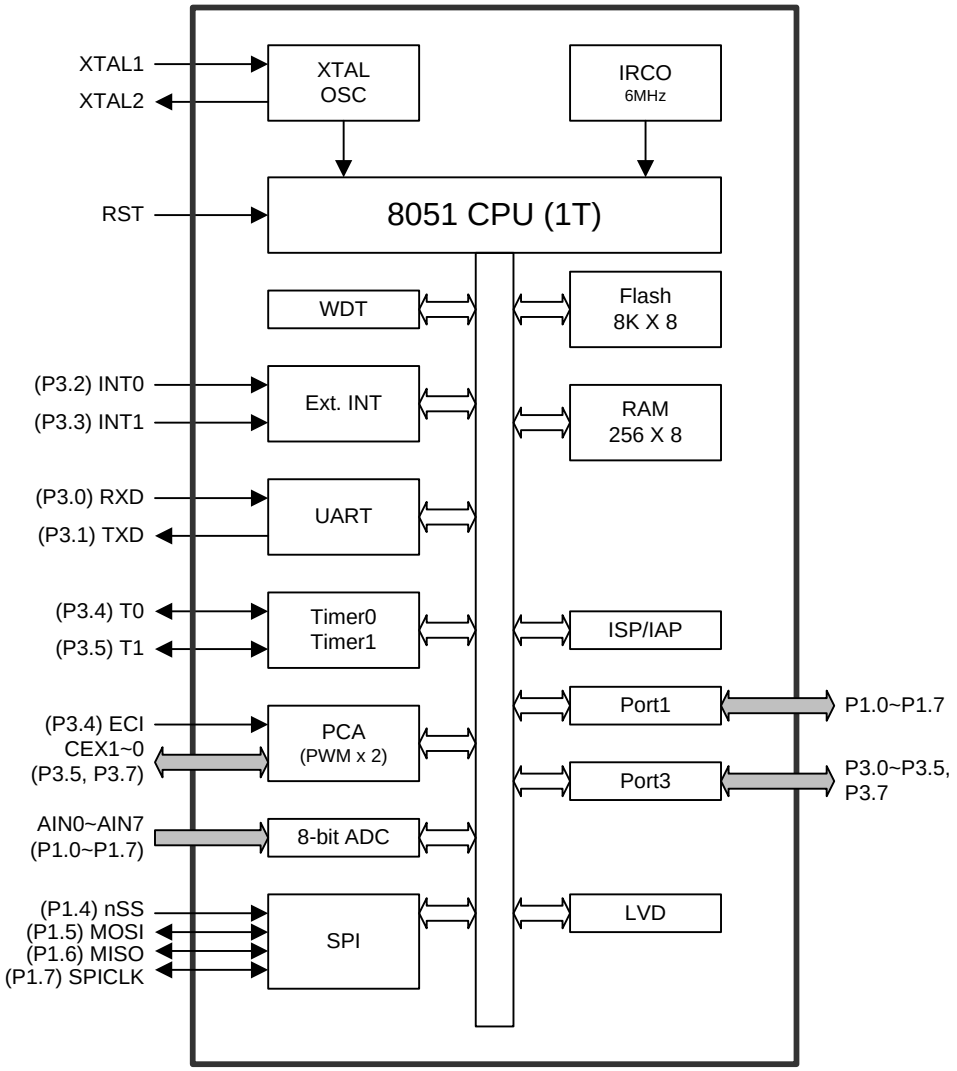
**MA801_MA802** 有 **8K** 字节的内置 Flash 存储器用于保存代码和数据。Flash 存储器可以通过并行编程模式或者 ISP 模式（在系统编程）进行编程。同时，它也提供了在应用编程（IAP）的能力。ISP 让使用者无需从产品中取下微控制器就可以下载新的代码；IAP 意味着应用程序正在运行时，微控制器能够在 Flash 中写入非易失数据。这些功能都由内建的电荷泵提供编程用的高压，而无需外部提供。

**MA801_MA802** 除了标准 80C52 的所有功能包括 256 字节的随机存储器，两组 I/O 端口，两个外部中断，一个多源 4 级中断控制器和两个 16 位定时/计数器。另外，**MA801_MA802** 还有一个 8 位 ADC，一个两通道 PCA，一个 SPI，一个 Watchdog 定时器，一个低电压侦测器，一个片上晶体振荡器，一个内置振荡器(IRCO,6MHz)，和一个改进了多处理器通信的更通用的串口（EUART）。

**MA801_MA802** 有多种工作模式可以减少功耗：空闲模式和掉电模式。在空闲模式下，CPU 被冻结而外围和中断系统依然运行。在掉电模式下，RAM 和特殊功能寄存器 SFR 的值被保存，而其它所有功能无效。最重要的是，在掉电模式下，芯片可以被多种中断或复位源唤醒。

## 2. 方框图

图 2–1. 方框图


XTAL1
XTAL2
XTAL OSC
IRCO 6MHz
RST
8051 CPU (1T)
WDT
Flash 8K X 8
(P3.2) INT0
(P3.3) INT1
Ext. INT
RAM 256 X 8
(P3.0) RXD
(P3.1) TXD
UART
(P3.4) T0
(P3.5) T1
Timer0 Timer1
ISP/IAP
Port1
P1.0~P1.7
(P3.4) ECI
CEX1~0 (P3.5, P3.7)
PCA (PWM x 2)
Port3
P3.0~P3.5, P3.7
AIN0~AIN7 (P1.0~P1.7)
8-bit ADC
(P1.4) nSS
(P1.5) MOSI
(P1.6) MISO
(P1.7) SPICLK
SPI
LVD

# 3. 特殊功能寄存器

## 3.1. SFR 地址表

表 3–1. SFR 地址表

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

## 3.2. SFR 位分配表

表 3–2. SFR 位分配表

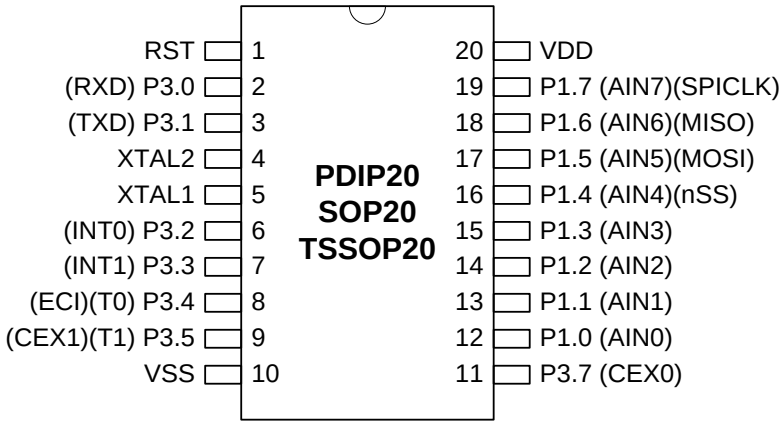
| 符号 | 描述 | 地址 | 位地址及符号 | | | | | | | | 复位值 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 | |
| SP | 堆栈指针 | 81H | | | | | | | | | 00000111B |
| DPL | 数据指针低 8 位 | 82H | | | | | | | | | 00000000B |
| DPH | 数据指针高 8 位 | 83H | | | | | | | | | 00000000B |
| SPISTAT | SPI 状态寄存器 | 84H | SPIF | WCOL | -- | -- | -- | -- | -- | -- | 00xxxxxxB |
| SPICON | SPI 控制寄存器 | 85H | SSIG | SPEN | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 | 00000100B |
| SPIDAT | SPI 数据寄存器 | 86H | | | | | | | | | 00000000B |
| PCON | 电源控制寄存器. | 87H | SMOD | SMOD0 | LVF | POF | GF1 | GF0 | PD | IDL | 00110000B |
| TCON | 定时器控制寄存器 | 88H | TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 | 00000000B |
| TMOD | 定时器模式 | 89H | GATE | C/T | M1 | M0 | GATE | C/T | M1 | M0 | 00000000B |
| TL0 | 定时器 0 低 8 位 | 8AH | | | | | | | | | 00000000B |
| TL1 | 定时器 1 低 8 位 | 8BH | | | | | | | | | 00000000B |
| TH0 | 定时器 0 高 8 位 | 8CH | | | | | | | | | 00000000B |
| TH1 | 定时器 1 高 8 位 | 8DH | | | | | | | | | 00000000B |
| AUXR | 辅助寄存器 | 8EH | T0X12 | T1X12 | URM0X6 | EADCI | ESPI | ENLVFI | -- | -- | 000000xxB |
| P1 | 端口 1 | 90H | P1.7 | P1.6 | P1.5 | P1.4 | P1.3 | P1.2 | P1.1 | P1.0 | 11111111B |
| P1M0 | P1 模式寄存器 0 | 91H | P1M0.7 | P1M0.6 | P1M0.5 | P1M0.4 | P1M0.3 | P1M0.2 | P1M0.1 | P1M0.0 | 00000000B |
| P1M1 | P1 模式寄存器 1 | 92H | P1M1.7 | P1M1.6 | P1M1.5 | P1M1.4 | P1M1.3 | P1M1.2 | P1M1.1 | P1M1.0 | 00000000B |
| SCON | 串口控制寄存器 | 98H | SM0/FE | SM1 | SM2 | REN | TB8 | RB8 | TI | RI | 00000000B |
| SBUF | 串口缓存寄存器 | 99H | | | | | | | | | xxxxxxxxB |
| IE | 中断使能寄存器 | A8H | EA | EPCA_LVD | ESPI_ADC | ES | ET1 | EX1 | ET0 | EX0 | 00000000B |
| SADDR | 从机地址寄存器 | A9H | | | | | | | | | 00000000B |
| P3 | 端口 3 | B0H | *P3.7* | -- | P3.5 | P3.4 | P3.3 | P3.2 | P3.1 | P3.0 | 1x111111B |
| P3M0 | P3 模式寄存器 0 | B1H | P3M0.7 | -- | P3M0.5 | P3M0.4 | P3M0.3 | P3M0.2 | P3M0.1 | P3M0.0 | 0x000000B |
| P3M1 | P3 模式寄存器 1 | B2H | P3M1.7 | -- | P3M1.5 | P3M1.4 | P3M1.3 | P3M1.2 | P3M1.1 | P3M1.0 | 0x000000B |
| IPH | 中断优先级高 | B7H | -- | PPCA_LVD | PSPI_ADC | PSH | PT1H | PX1H | PT0H | PX0H | x0000000B |
| IP | 中断优先级低 | B8H | -- | PPCA_LVD | PSPI_ADC | PSL | PT1L | PX1L | PT0L | PX0L | x0000000B |
| SADEN | 从地址屏蔽寄存器 | B9H | | | | | | | | | 00000000B |
| ADCTL | ADC 控制寄存器 | C5H | ADCON | SPEED1 | SPEED0 | ADCI | ADCS | CHS2 | CHS1 | CHS0 | 00000000B |
| ADCV | ADC 结果寄存器 | C6H | ADCV.7 | ADCV.6 | ADCV.5 | ADCV.4 | ADCV.3 | ADCV.2 | ADCV.1 | ADCV.0 | xxxxxxxxB |
| PCON2 | 电源控制 2 寄存器 | C7H | -- | -- | -- | -- | -- | CKS2 | CKS1 | CKS0 | xxxxx000B |
| PSW | 程序状态字 | D0H | CY | AC | F0 | RS1 | RS0 | OV | *F1* | P | 00000000B |
| CCON | PCA 控制寄存器 | D8H | CF | CR | -- | -- | -- | -- | CCF1 | CCF0 | 00xxxx00B |
| CMOD | PCA 模式寄存器 | D9H | CIDL | -- | -- | -- | -- | CPS1 | CPS0 | ECF | 0xxxx000B |
| CCAPM0 | PCA 模块 0 模式 | DAH | -- | ECOM0 | CAPP0 | CAPN0 | MAT0 | TOG0 | PWM0 | ECCF0 | x0000000B |
| CCAPM1 | PCA 模块 1 模式 | DBH | -- | ECOM1 | CAPP1 | CAPN1 | MAT1 | TOG1 | PWM1 | ECCF1 | x0000000B |
| ACC | 累加器 | E0H | ACC.7 | ACC.6 | *ACC.5* | ACC.4 | *ACC.3* | ACC.2 | ACC.1 | ACC.0 | 00000000B |
| WDTCR | 看门狗控制寄存器 | E1H | WRF | -- | ENW | CLW | WIDL | PS2 | PS1 | PS0 | 0x000000B<br>xxx0xxxxB |
| IFD | ISP Flash 数据 | E2H | | | | | | | | | 11111111B |
| IFADRH | ISP Flash 地址高 8 位 | E3H | | | | | | | | | 00000000B |
| IFADRL | ISP Flash 地址低 8 位 | E4H | | | | | | | | | 00000000B |
| IFMT | ISP 模式表 | E5H | -- | -- | -- | -- | -- | -- | MS1 | MS0 | xxxxxx00B |
| SCMD | ISP 串行命令 | E6H | | | | | | | | | xxxxxxxxB |
| ISPCR | ISP 控制寄存器 | E7H | ISPEN | SWBS | SWRST | CFAIL | -- | WAIT.2 | WAIT.1 | WAIT.0 | 0000x000B |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CL | PCA 基准定时器低 8 位 | E9H | CL.7 | CL.6 | CL.5 | CL.4 | CL.3 | CL.2 | CL.1 | CL.0 | 00000000B |
| CCAP0L | PCA 模块 0 捕获低 8 位 | EAH | | | | | | | | | 00000000B |
| CCAP1L | PCA 模块 1 捕获低 8 位 | EBH | | | | | | | | | 00000000B |
| B | B 寄存器 | F0H | B.7 | B.6 | B.5 | B.4 | B.3 | B.2 | B.1 | B.0 | 00000000B |
| PCAPWM0 | PCA PWM0 模式 | F2H | -- | -- | -- | -- | -- | -- | EPC0H | EPC0L | xxxxxx00B |
| PCAPWM1 | PCA PWM1 模式 | F3H | -- | -- | -- | -- | -- | -- | EPC1H | EPC1L | xxxxxx00B |
| CH | PCA 基准定时器高 8 位 | F9H | CH.7 | CH.6 | CH.5 | CH.4 | CH.3 | CH.2 | CH.1 | CH.0 | 00000000B |
| CCAP0H | PCA 模块 0 捕获高 8 位 | FAH | | | | | | | | | 00000000B |
| CCAP1H | PCA 模块 1 捕获高 8 位 | FBH | | | | | | | | | 00000000B |

# 4. 引脚结构

## 4.1. 封装指南

图 4–1. MA801_MA802 20 脚 顶视图



## 4.2. 引脚描述

表 4–1. 引脚描述

| 助记符 | 引脚号 | | | I/O 类型 | 描述 |
|---|---|---|---|---|---|
| | 20-Pin PDIP | 20-Pin SOP | 20-Pin TSSOP | | |
| P1.0<br>(AIN0) | 12 | 12 | 12 | I/O | * 端口 1.0.<br>* AIN0: ADC 模拟输入通道 0 |
| P1.1<br>(AIN1) | 13 | 13 | 13 | I/O | *端口 1.1.<br>* AIN1: ADC 模拟输入通道 1 |
| P1.2<br>(AIN2) | 14 | 14 | 14 | I/O | *端口 1.2.<br>* AIN2: ADC 模拟输入通道 2 |
| P1.3<br>(AIN3) | 15 | 15 | 15 | I/O | *端口 1.3.<br>* AIN3: ADC 模拟输入通道 3 |
| P1.4<br>(AIN4)<br>(nSS) | 16 | 16 | 16 | I/O | *端口 1.4.<br>* AIN4: ADC 模拟输入通道 4<br>* nSS: SPI 从机选择 |
| P1.5<br>(AIN5)<br>(MOSI) | 17 | 17 | 17 | I/O | * Port 1.5.<br>* AIN5: ADC 模拟输入通道 5<br>* MOSI: SPI 主机输出&从机输入 |
| P1.6<br>(AIN6)<br>(MISO) | 18 | 18 | 18 | I/O | *端口 1.6.<br>* AIN6: ADC 模拟输入通道 6<br>* MISO: SPI 主机输入&从机输出 |
| P1.7<br>(AIN7)<br>(SPICLK) | 19 | 19 | 19 | I/O | *端口 1.7.<br>* AIN7: ADC 模拟输入通道 7<br>* SPICLK: SPI 时钟，主机输出和从机输入 |
| P3.0<br>(RXD) | 2 | 2 | 2 | I/O | *端口 3.0.<br>* RXD: UART 串行输入口 |
| P3.1<br>(TXD) | 3 | 3 | 3 | I/O | *端口 3.1.<br>* TXD: UART 串行输出口 |
| P3.2<br>(INT0) | 6 | 6 | 6 | I/O | *端口 3.2.<br>* INT0: 外部中断 0 输入 |
| P3.3<br>(INT1) | 7 | 7 | 7 | I/O | *端口 3.3.<br>* INT1: 外部中断 1 输入 |
| P3.4<br>(T0)<br>(ECI) | 8 | 8 | 8 | I/O | *端口 3.4.<br>* T0: 定时器/计数器 0 外部输入<br>* ECI: PCA 外部时钟输入. |
| P3.5<br>(T1)<br>(CEX1) | 9 | 9 | 9 | I/O | *端口 3.5.<br>* T1: 定时器/计数器 1 外部输入<br>* CEX1: PCA 模块 1 外部 I/O. |
| P3.7<br>(CEX0) | 11 | 11 | 11 | I/O | *端口 3 .7.<br>* CEX0: PCA 模块 0 外部 l I/O. |
| XTAL2 | 4 | 4 | 4 | O | * XTAL2: 片上晶体振荡电路输出 |
| XTAL1 | 5 | 5 | 5 | I | * XTAL1: 片上晶体振荡电路输入 |
| RST | 1 | 1 | 1 | I | * RST: 外部复位输入，高有效 |
| VDD | 20 | 20 | 20 | P | 电源输入。MA801 为. 5V，MA802 为 3.3V |
| VSS | 10 | 10 | 10 | G | 地，0V 参照 |

# 5. 8051 CPU 功能描述

## 5.1. CPU 寄存器

PSW: 程序状态字
SFR 地址 = 0xD0 复位值 = 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: 进位标志位
AC: 辅助进位标志位
F0: 通用标志位 0.
RS1: 寄存器组选择位 1.
RS0: 寄存器组选择位 0.
OV: 溢出标志位
F1: 通用标志位 1.
P: 奇偶标志位

程序状态字 (PSW)包含反映 CPU 当前状态的几个状态位。如上所示，PSW 存在于特殊功能寄存器 SFR 区。它包含进位标志，辅助进位标志 (应用于 BCD 操作)， 两个寄存器组选择位，溢出标志， 奇偶标志和两个用户可设定的标志位。

进位标志，不仅有算术运算的进位功能，也充当许多布尔运算的“累加器”。
RS0 和 RS1 被用来选择 4 组中的任意一组寄存器组，详见内部 RAM 章节“6.2 片上数据”.。
奇偶位反映 1S 内累加器数字和的状况， 1S 内累加器中数字和是奇数则 P=1 否则 P=0

SP: 堆栈指针
SFR 地址 = 0x81 复位值 = 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 |

堆栈指针保持栈顶位置，每执行一个 PUSH 指令，SP 会自动增加，SP 缺省值为 0X07H。

DPL: 数据指针低字节
SFR 地址 = 0x82 复位值 = 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 |

DPL 是 DPTR 的低字节，DPTR 用来间接访问 XRAM 和程序空间。

DPH: 数据指针高字节
SFR 地址 = 0x83 复位值 = 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 |

DPH 是 16 位 DPTR 的高字节，DPTR 用来间接访问 XRAM 和程序空间。

ACC: 累加器

SFR 地址 = 0xE0 复位值= 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 |

算术运算的累加器。

B: B 寄存器

SFR 地址 = 0xF0 复位值 = 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 |

另一个算术运算的累加器。

.

## 5.2. CPU 时序

**MA801_MA802** 是基于 80C51 的高效 1-T 结构的单芯片微处理器，与 8051 指令集兼容，每条指令需要 1~6 个时钟信号(比标准 8051 快 6~7 倍)。使用流线型结构同标准的 8051 结构比较大大增加了指令完成的速度，指令的时序也和标准的 8051 不同。

多数 8051 执行指令，一个区别是建立在机器周期和时钟周期之间，机器周期来自 2 到 12 个时钟周期长度。然而，1-T 结构的 80C51 执行指令是基于单独的时钟周期时序。所有指令时序被指定在时钟周期期间。关于 1T-80C51 指令更详细的说明，请参考“24 指令集”，这里有每一条指令的助记符、字节数、时钟周期数。

## 5.3. CPU 寻址模式

***直接寻址(DIR)***
直接寻址时操作数用指令中一个 8 位地址的区域表示，只有内部数据存储器和特殊功能寄存器可以直接寻址。

***间接寻址(IND)***
间接寻址时指令用一个包含操作数地址的寄存器表示，内部和外部存储器均可间接寻址。
8 位地址的地址寄存器可以是选中区的 R0 或 R1 或堆栈指针。
16 位地址的地址寄存器只能是 16 位的“数据指针”寄存器，DPTR。

***寄存器操作 (REG)***
包含寄存器 R0 到 R7 的寄存器组，可以被某些指令寻址，该指令的操作码包含了 3 个位的寄存器参数。这种访问寄存器的操作方式有更高的代码效率，因为这种模式减少了一个地址字节。当指令被执行时，其中被选取的组里 8 个寄存器中的一个寄存器被存取。

***寄存器特别指令***
有些指令特定的指向某一寄存器。例如，一些指令总是操作累加器或数据指针等等。因此这些指令无需地址字节。操作码自己就行了。

***立即数(IMM)***
常量的值可以在程序存储器里紧跟着操作码。

***索引寻址***
索引寻址只能访问程序存储器，且只读。这种寻址模式用查表法读取程序存储器。一个 16 位基址寄存器（数据指针 DPTR 或程序计数器 PC）指向表的基地址，累加器提供偏移量。程序存储器中表项目地址由基地址加上累加器数据后形成。另一种索引寻址方式是利用条件指令。跳转指令中的目标地址是基地址加上累加器数据后的值。

# 6. 存储器组织

像所有的80C51 一样，**MA801_MA802** 的程序存储器和数据存储器的地址空间是分开的。这样 8 位微处理器可以通过一个 8 位的地址快速而有效的访问数据存储器。

程序存储器(ROM)只能读取，不能写入。最大可以达到 **8K** 字节。在 **MA801_MA802** 中，所有的程序存储器都是片上 Flash 存储器。因为没有设计外部程序使能 (/EA)和编程使能 (/PSEN) 信号，所以不允许外接程序存储器。

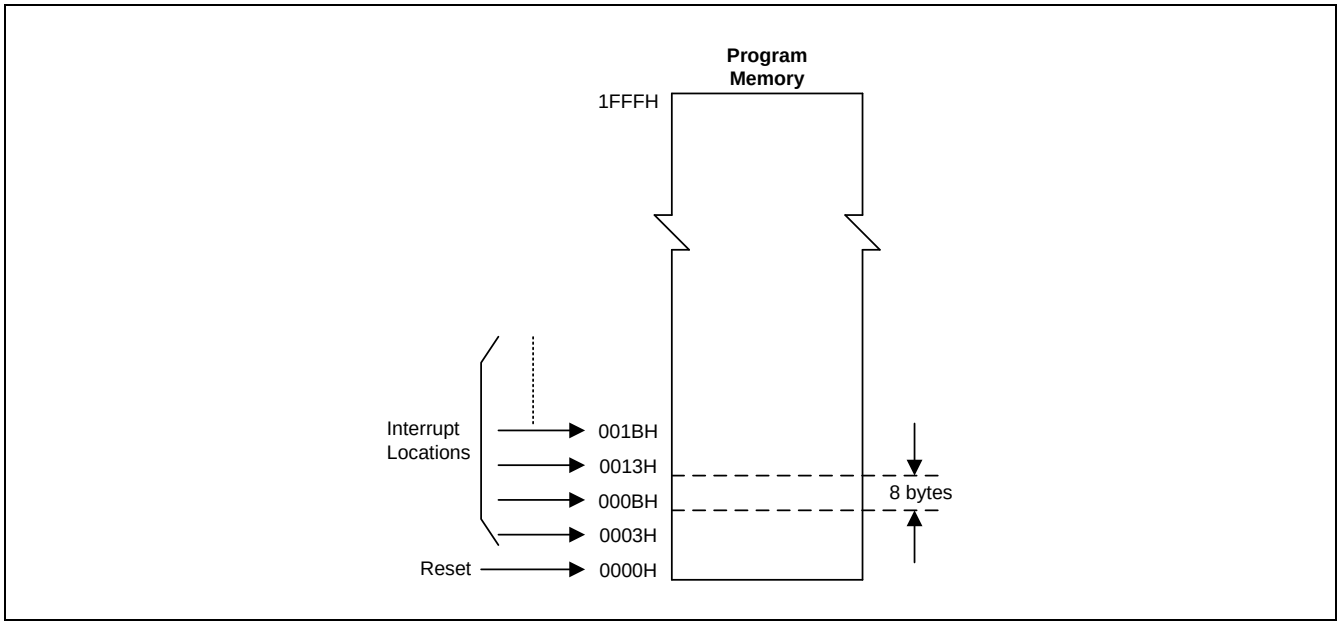
数据存储器使用与程序存储器不同的地址空间。**MA801_MA802** 只有 256 字节的内部暂存 RAM。

## 6.1. 片上程序存储器

程序存储器用来保存让 CPU 进行处理的程序代码，如图 6–1.所示。复位后，CPU 从地址为 0000H 的地方开始运行，用户应用代码的起始部分应该放在这里。为了响应中断，中断服务位置(被称为中断矢量)应该位于程序存储器。每个中断在程序存储器中有一个固定的起始地址，中断使 CPU 跳到这个地址运行中断服务程序。举例来说，外部中断 0 被指定到地址 0003H，如果使用外部中断 0，那么它的中断服务程序一定是从 0003H 开始的。如果中断未被使用，那么这些地址就可以被一般的程序使用。

中断服务程序的起始地址之间有 8 字节的地址间隔： *0003H* 用于外部中断 0；*000BH* 用于定时器 0；*0013H* 用于外部中断 1；*001BH* 用于定时器 1 等等。如果中断服务程序足够短，它完全可以放在这 8 字节的空间中。如果其他的中断也被使用的话，较长的中断服务程序可以通过一条跳转指令越过后面的中断服务起始地址。

图 6–1. 程序存储器



## 6.2. 片上数据存储器

图 6–2 向 MA801_MA802 使用者展示了内部和外部数据存储器的空间划分。内部数据存储器被划分为三部分，通常被称为低 128 字节 RAM，高 128 字节 RAM 和 128 字节 SFR 空间。内部数据存储器的地址线只有 8 位宽，因此地址空间只有 256 字节。SFR 空间的地址高于 7FH，用直接地址访问；而用间接访问的方法访问高 128 字节的 RAM。这样虽然 SFR 和高 128 字节 RAM 占用相同的地址空间（80H—FFH），但他们物理上是分开的。.

如图 6–3 所示，低 128 字节 RAM 与所有 80C51 一样。最低的 32 字节被划分为 4 组每组 8 字节的寄存器组。指令中称这些寄存器为 R0 到 R7。程序状态字 (PSW) 中的两位用于选择哪组寄存器被使用。这使得程序空间能够被更有效的使用，因为对寄存器访问的指令比使用直接地址的指令短。 接下来的 16 字节是可以位寻址的存储器空间。80C51 的指令集包含一个位操作指令集，这区域中的 128 位可以被这些指令直接使用。位地址从 00H 开始到 7FH 结束。

所有的低 128 字节 RAM 都可以用直接或间接地址访问，而高 128 字节 RAM 只能用间接地址访问。

图 6–4 给出了特殊功能寄存器 (SFR) 的概览。SFR 包括端口寄存器，定时器和外围器件控制器，这些寄存器只能用直接地址访问。SFR 空间中有 16 个地址同时支持位寻址和直接寻址。可以位寻址的 SFR 的地址末位是 0H 或 8H。

图 6–2. 数据存储器

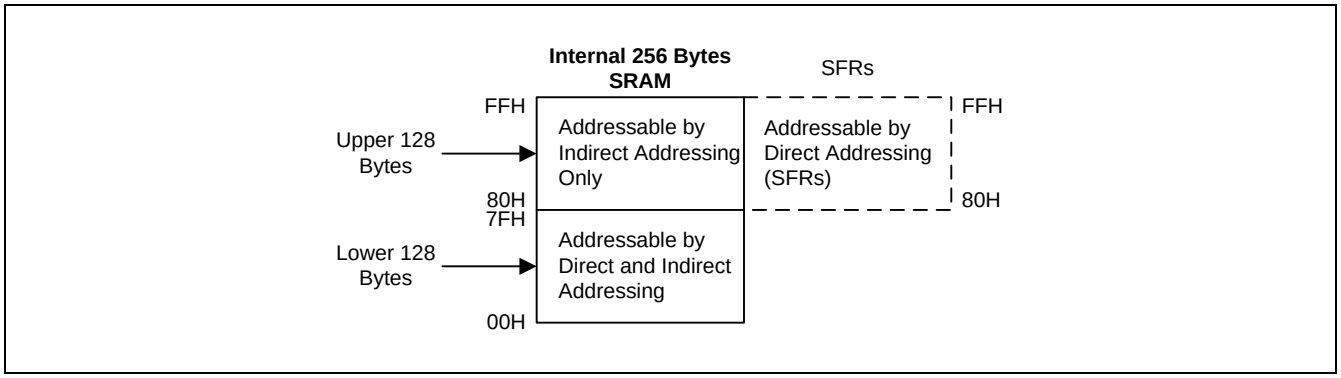



图 6–3.内部 RAM 的低 128 字节

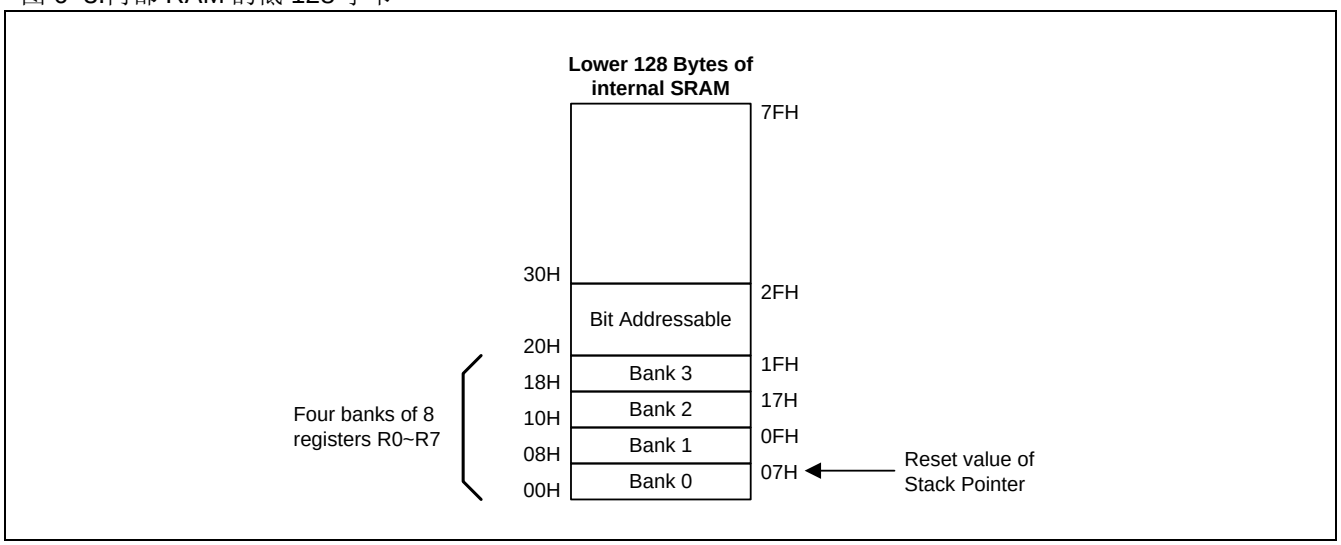
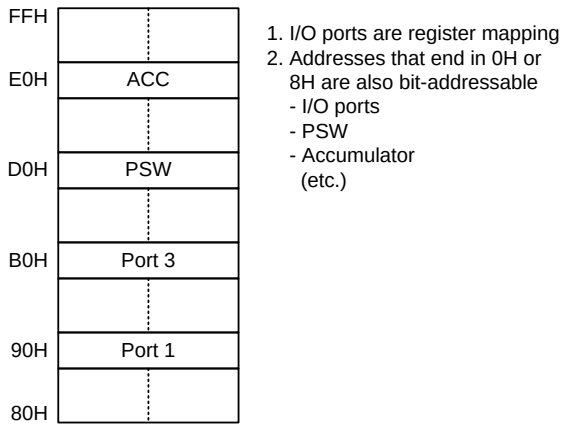


图 6–4. SFR 空间


FFH
E0H
ACC
D0H
PSW
B0H
Port 3
90H
Port 1
80H
1. I/O ports are register mapping
2. Addresses that end in 0H or 8H are also bit-addressable
- I/O ports
- PSW
- Accumulator
(etc.)

## 6.3. 关于 C51 编译器的声明标识符

下面说明了在 C51 编译器使用各样 **MA801_MA802** 存储空间的声明标识符。

***data***
128 字节的内部数据存储空间 (00h~7Fh)。使用除 MOVX 和 MOVC 以外的指令，可以直接或间接的访问。全部或部分的堆栈可能保存在此区域中。

***idata***
间接数据。256 字节的内部数据存储空间 (00h~FFh) 使用除 MOVX 和 MOVC 以外的指令间接访问。全部或部分的堆栈可能保存在此区域中。此区域包括 data 区 和 data 区以上的 128 字节。

***sfr***
特殊功能寄存器。CPU 寄存器和外围部件控制/状态寄存器，只能通过直接地址访问。

***xdata***
没有外部数据或片上的扩展 RAM (XRAM)。

***pdata***
没有分页的外部数据(256 字节) 或片上的扩展 RAM（XRAM）。

***code***
**8K** 程序存储空间。通过“MOVC @A+DTPR”访问，作为程序部分被读取。

# 7. 数据指针寄存器(DPTR)

**MA801_MA802** 的 DPTR 只有一种设置。 **MA801_MA802** 不支持访问外部存储器和 MOVX 指令。

# 8. 系统时钟

系统时钟有 2 个时钟源：外部晶振 ， 内部慢频 RC 震荡器(IRCO)。如图 8–1 所示 **MA801_MA802** 系统时钟结构。

**MA801_MA802** 可以由硬件选项控制位 ENROSC 选择从外部晶振或者内部 6MHz IRCO 启动。使用通用编程器或者笙泉专用的编程器使能或禁止 ENROSC。

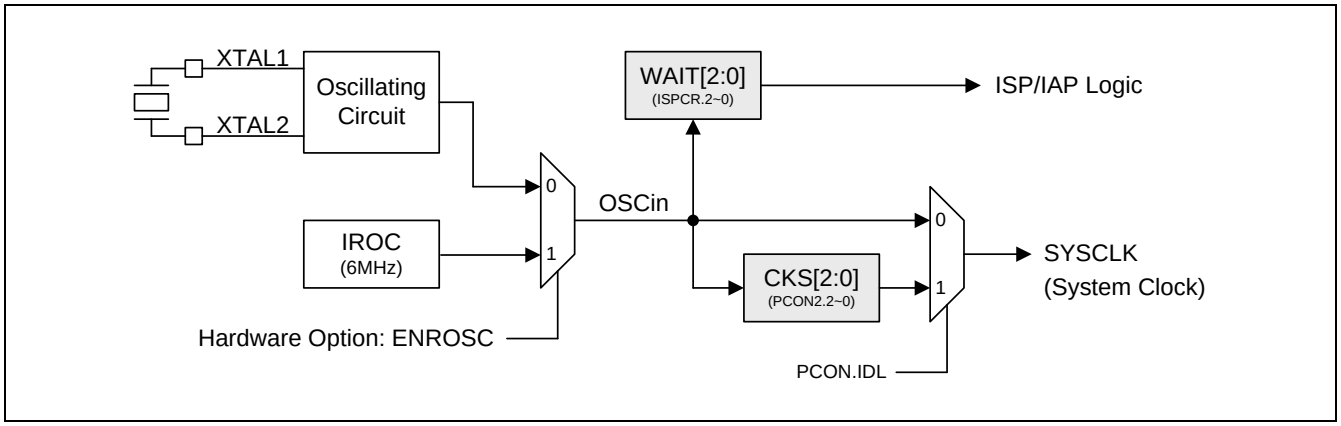
内置 IRCO 提供 6MHz 的频率。查找更多 IRCO 特性描述，请参见章节“23.4 IRCO ”。

如图 8–1 所示 在空闲模式下，系统时钟是从两个时钟源中的一个通过分频器获得。用户可编程分频器控制位 CKS2~CKS0 (在 PCON2 寄存器) 来获得想要的系统时钟。换言之，CKS2~CKS0 控制位仅在**空闲模式**有效。

## 8.1. 时钟结构

图 8–1 显示了 **MA801_MA802** 的时钟系统。系统时钟可以使用外部晶振或者内部 RC 振荡作为来源。

图 8–1. 系统时钟



## 8.2. 时钟寄存器

PCON2: 电源控制寄存器 2

SFR 地址 = 0xC7 复位值 = xxxx-x000

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

Bit 7~3: 保留。写 PCON2 时，这些位软件必须写“0”。

Bit 2~0: CKS2 ~ CKS0, 在**空闲模式**下可编程系统时钟选项。CKS[2:0] 缺省值“000” 选择系统时钟是 OSCin/1。

| CKS[2:0] | 空闲模式下系统时钟选项 |
|---|---|
| 0 0 0 | OSCin/1 |
| 0 0 1 | OSCin /2 |
| 0 1 0 | OSCin /4 |
| 0 1 1 | OSCin /8 |
| 1 0 0 | OSCin /16 |
| 1 0 1 | OSCin /32 |
| 1 1 0 | OSCin /64 |
| 1 1 1 | OSCin /128 |

## 8.3. 时钟示例代码

(1). 功能需求: 在空闲模式下 SYSCLK 切换到 OSCin/128 (缺省值是 OSCin/1)

汇编语言代码范例:

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

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

C 汇编语言代码范例:

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

   PCON2 &= (CKS2 | CKS1 | CKS0);        // 系统时钟分频 /128
                                // CKS[2:0], 系统时钟分频
                                //   0    | OSCin/1
                                //   1    | OSCin/2
                                //   2    | OSCin/4
                                //   3    | OSCin/8
                                //   4    | OSCin/16
                                //   5    | OSCin/32
                                //   6    | OSCin/64
                                //   7    | OSCin/128
```

# 9. 看门狗定时器 (WDT)

## 9.1. WDT 结构

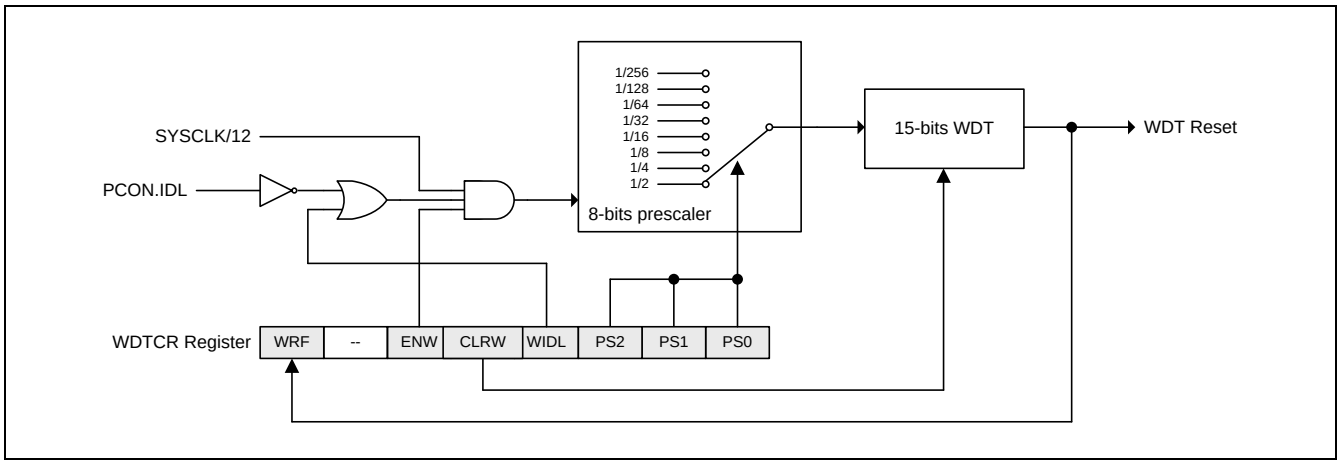
看门狗定时器 (WDT) 用来使程序从跑飞或死机状态恢复的一个手段。WDT 由一个 15 位独立定时器、一个 8 位预分频器和一个控制寄存器(WDTCR)组成。 图 9–1 显示了 **MA801_MA802** WDT 结构框图。

当 WDT 使能，它的时间基准是从 SYSCLK/12 而来。WDT 溢出会触发一个系统复位，并且置位 WDTCR.7 上的 WRF。为了阻止 WDT 溢出，软件必须在 WDT 溢出之前写“1”到 CLRW 位(WDTCR.4)去清它。

一旦置位 ENW 位使能 WDT，就没有方法去禁止它，除了上电复位以外。在外部复位（RST 引脚），软件复位和 WDT 复位后，WDTCR 将保持之前的值不变。

ENW 是一次性使能有效，仅写“1”有效。请参见章节 “9.3 WDT ” 获得更详细的信息。

图 9–1. 看门狗定时器




## 9.2. WDT 在掉电模式和空闲模式期间

在空闲模式下， WIDL 位 (WDTCR.3) 决定 WDT 是否计数。 设置这个位能让 WDT 在空闲模式一直计数。

在掉电模式下，因为 SYSCLK 停止导致 WDT 被冻结。在 MCU 唤醒后，WDT 将继续计数。

## 9.3. WDT 寄存器

WDTCR: 看门狗控制寄存器
SFR 地址 = 0xE1 上电复位值 = 0000-0111 (xxx0_xxxx 由硬件选项)

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

Bit 7: WRF, WDT 复位标志
0: 这位应由软件清除
1: 当 WDT 溢出，这位由硬件置位，表示发生了一个 WDT 复位

Bit 6: 保留。写 WDTCR 时，这位软件必须写“0”

Bit 5: ENW. 使能 WDT.
0: 禁止 WDT 运行.
1: 使能 WDT，一旦 ENW 被置位，软件将不能清除它

Bit 4: CLRW. WDT 计数器清零位
0: 在此位写“0”WDT 没有任何操作 .
1: 在此位写 “1” 会清除 9 位 WDT 计数器到 000H， 注意此位不能写“0”清除，设置此位清除 WDT 重新计数。

Bit 3: WIDL. WDT 空闲模式控制
0: WDT 停止计数 MCU 在空闲模式。.
1: WDT 保持计数 MCU 在空闲模式

Bit 2~0: PS2 ~ PS0, 选择用于 WDT 时钟基准输入的预分频器输出

| PS[2:0] | 预分频器 | WDT 周期(如果 SYSCLK= 12MHz) |
|---|---|---|
| 0 0 0 | 2 | 65.5ms |
| 0 0 1 | 4 | 131ms |
| 0 1 0 | 8 | 262ms |
| 0 1 1 | 16 | 524ms |
| 1 0 0 | 32 | 1.05S |
| 1 0 1 | 64 | 2.10S |
| 1 1 0 | 128 | 4.19S |
| 1 1 1 | 256 | 8.39S |

## 9.4. WDT 硬件选项

除了由软件初始化外，WDTCR 寄存器还能在上电的时候由硬件选项 HWENW,HWWIDL，HWPS[2:0]和 WDSFWP 来自动初始化，这些选项通过通用编程器来编程，如下所述。

如果 HWENW 编程为“使能”，则硬件在上电时为 WDTCR 寄存器作如下的初始化工作：（1）ENWI 位置 1。(2)载入 HWWIDL 的值到 WIDL 位。（3）载入 HWPS【2：0】的值到 PS【2：0】位。

如果 HWENW 和 WDSFWP 都被编程为“使能”，则硬件仍然会在上电时由 WDT 硬件选项初始化 WDTCR 寄存器的内容。之后，任何对 WDTCR 的位的写动作都会被忽略，除了写“1”到 WDTCR.4(CLRW)位来清 WDT 之外.

**HWENW:** 硬件加载 WDTCR 的“ENW”

☑:使能：上电时自动硬件使能看门狗定时器，并且自动加载 HWWIDL 和 HWPS2~0 的值到 WDTCR 中。

☐: 禁止： 上电时看门狗定时器（WDT）不自动使能。

**HWWIDL, HWPS2, HWPS1, HWPS0**:

当 HWENW 被使能，上电复位时，这四个保险丝位将被载入到特殊功能寄存器 WDTCR 中。

**WDSFWP**:

☑:使能. WDT 特殊功能寄存器 WDTCR 位 WIDL, PS2, PS1 和 PS0 软件写保护。

☐:禁止. WDT 特殊功能寄存器 WDTCR 位 WIDL, PS2, PS1 和 PS0 可以由软件写

## 9.5. WDT 示例代码

(1)功能需求: 使能 WDT 并且选择 WDT 预分频为 1/32

汇编语言代码范例:

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

   ANL    WDTCR,#(0FFh - WRF)          ; 清除 WRF 标志(写“0”)
   MOV    WDTCR,#(ENW + CLRW + PS2)    ; 使能 WDT 并且选择 WDT 预分频为 1/32
```

C 语言代码范例:

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

   WDTCR &= ~WRF;                       //清除 WRF 标志(写“0”)
   WDTCR = (ENW | CLRW | PS2);          //使能 WDT 并且选择 WDT 预分频为 1/32
                                // PS[2:0]  | WDT 预分频器选项
                                //   0    | 1/2
                                //   1    | 1/4
                                //   2    | 1/8
                                //   3    | 1/16
                                //   4    | 1/32
                                //   5    | 1/64
                                //   6    | 1/128
                                //   7    | 1/256
```

# 10. 系统复位

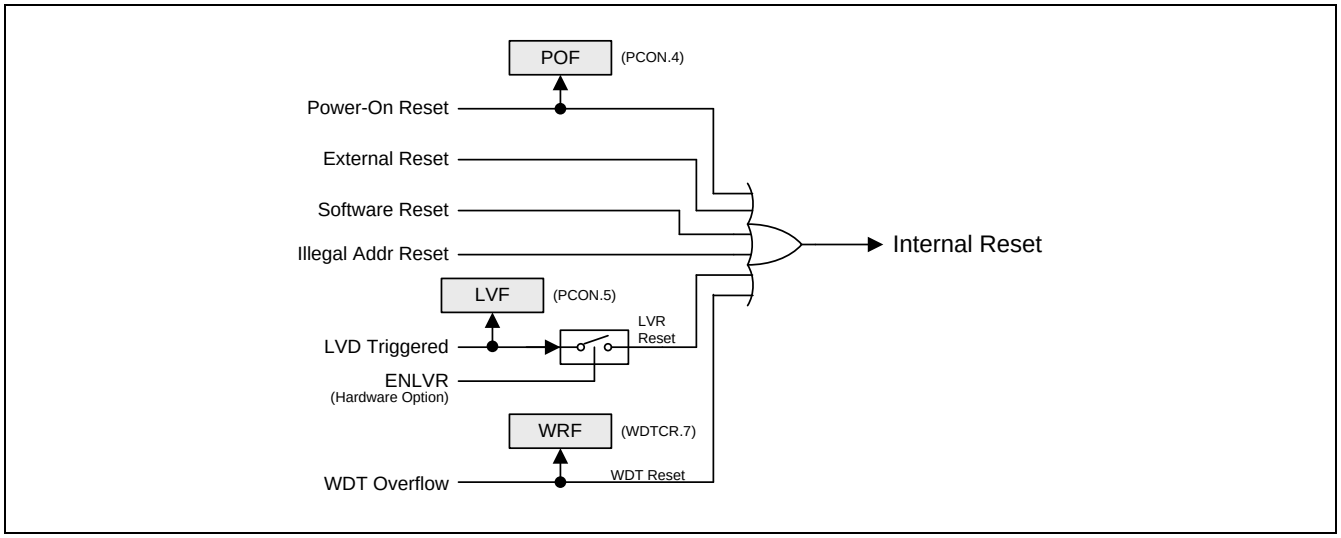
复位期间， 所有的 I/O 寄存器都设置为初始值，程序会根据 OR 设置选择从复位向量的 0000H 开始运行，或者根据硬件选项设置从 ISP 地址开始运行。The **MA801_MA802** 有 6 种复位源： 上电复位， 外部复位， 软件复位，非法地址复位， WDT 复位和 低电压复位。 如图所示：图 10–1 显示了 **MA801_MA802** 的系统复位源
.

下面的选项描述复位产生源及其相应的控制寄存器和指示标志。

## 10.1. 复位源

图 10–1 展示了 **MA801_MA802** 的复位系统，和所有复位源

图 10–1. 系统复位源




## 10.2. 上电复位

上电复位 (POR)用于在电源上电过程中产生一个复位信号。 微控制器在 VDD 电压上升到 $V_{POR}$ (POR 开始电压)电压之前将保持复位状态。 VDD 电压降到 $V_{POR}$ 之下后微控制器将再次进入复位状态。 在一个电源周期中， 如果需要再产生一次上电复位 VDD 必须降到 $V_{POR}$ 之下。

PCON: 电源控制寄存器
SFR 地址 = 0x87 上电复位值 = 0011-0000, 复位值 = 000X-0000

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

Bit 4: POF. 上电标志.
0: 这标志必须通过软件清零以便认出下一个复位类型
1: 当 VDD 从 0 伏上升到正常电压时硬件复位， POF 也能由软件置位。

上电标志 POF 在上电过程中由硬件置“1” 或当 VDD 电压降到 $V_{POR}$ 电压之下时由硬件置“1”。它能通过软件来清除但不受任何热复位（譬如：外部 RST 引脚复位、低电压复位、软件(ISPCR.5)复位和 WDT 复位）的影响。它帮助用户检测 CPU 是否从上电开始运行。注意：POF 必须由软件清除。

## 10.3. 外部复位

保持复位引脚 RST 至少 24 个振荡周期的高电平，将产生一个复位信号，为确保 MCU 正常工作，必须在 RET 引脚上连接可靠的硬件复位电路。

## 10.4. 软件复位

软件通过对 SWRST(ISPCR.5) 位写“1”触发一个系统热复位， 软件复位后，硬件置位 SWRF 标志(PCON1.7)。SWBS 标志决定 CPU 是从 ISP 还是 AP 区域开始运行程序

ISPCR: ISP 控制寄存器
SFR 地址 = 0xE5 复位值 = 0000-x000

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

Bit 6: SWBS, 软件引导控制
0: 复位后从 AP-空间启动
1: 复位后从 ISP-空间启动.

Bit 5: SWRST, 软件复位触发控制
0: 无操作
1: 产生软件系统复位，它将被硬件自动清除。

## 10.5. 低电压复位

在 **MA801_MA802** 里，有一个低压检测器（LVD）去监视 VDD 电源。LVD 固定检测电压对于 MA801 为 VDD=3.7V，对于 MA802 为 VDD=2.3V。如果 VDD 电源下降到低于 LVD 监视电压，关联的标志位 LVF 将被置位。如果 ENLVR（硬件选项）使能，LVD 将触发一个系统复位，并且置位 LVF 表示产生了一个 LVD 复位。

PCON: 电源控制寄存器
SFR 地址 = 0x87 上电复位值= 0011-0000, 复位值 = 000X-0000

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

Bit 4: LVF, 低电压标志
0: 这位必须由软件在上电后清零，以便继续检测。
1: 这位仅由硬件在 VDD 掉到 LVD 监视电平时置位

## 10.6. WDT 复位

当 WDT 使能开始计数， WDT 溢出时置位 WDTF 标志，并且将触发一个系统复位导致 CPU 重启。软件可以读 WRF 标志来确认 WDT 复位发生。

WDTCR: 看门狗定时器控制寄存器
SFR 地址 = 0xE1 上电复位值 = 0000-0111 (xxx0_xxxx 由硬件选项)

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

Bit 7: WRF, WDT 复位标志
0: 这位必须由软件清零
1: 当 WDT 溢出，这位由硬件置位表示产生了一个 WDT 复位。

## 10.7. 非法地址复位

**MA801_MA802** 中, 如果程序运行到非法地址，如超过程序 ROM 限制，它将触发 CPU 复位。

## 10.8. 复位示例代码

(1) 功能需求: 触发一个软件复位

汇编语言代码范例:

```
SWRST                   EQU                 20h

   ORL     ISPCR,#SWRST             ; 触发软件复位
```

C 语言代码范例:

```
#define SWRST                       0x20

   ISPCR |= SWRST;                  // 触发软件复位
```

# 11. 电源管理

**MA801_MA802** 支持一个电源监控模块，低电压检测器(LVD), 和两种节能模式：空闲模式和掉电模式

LVD 在标志位 LVF 上报告芯片的电源状态，通过软件配置它可以产生 CPU 中断，或通过硬件选项产生 CPU 复位。两种节能模式为芯片应用提供不同的节能方案。通过 PCON 和 PCON2 寄存器来操作这些模式。

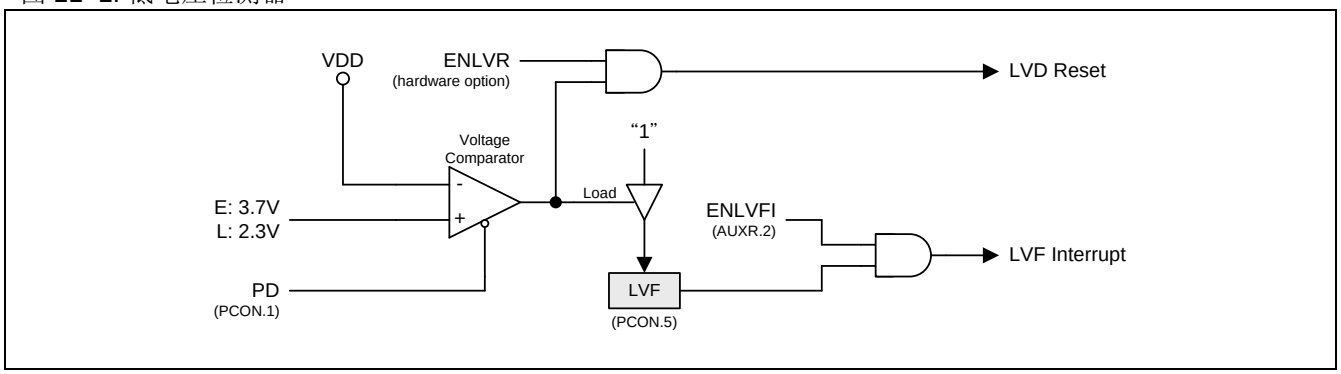
## 11.1. 低电压检测器

**MA801_MA802**, 有一个低压检测器（LVD）去监视 VDD 电源。图 11–1 是 LVD 的功能逻辑图。LVD 固定检测电压对于 MA801 为 VDD=3.7V，对于 MA802 为 VDD=2.3V。当 LVD 侦测到监测点电平时，相关联的位，LVF (PCON.5)将被置位。如果 ENLVFI (AUXR.2) 使能，置位 LVF 将会产生一个低电压检测中断。它不管 CPU 是在普通模式还是在空闲模式都会中断 CPU。

如果 ENLVR (硬件选项)使能，LVD 事件将会触发一个系统复位，并且置位 LVF 表示产生了一个 LVD 复位。LVD 不管 CPU 是在普通模式还是在空闲模式都会复位重启 CPU。

低电压检测器特别注意: 低压检测器不是一个精确的检测器。阀值电压会稍等温度变化的影响。随温度下降而下降，上升而上升。在温度范围内(-40℃, 85℃)，对于 MA802 阀值电压是在范围（2.7V,1.8V）之间，对于 MA801 是在（4.2V，3.2V）之间。为了控制低电压检测和复位，用户必须读另一笙泉文档 “*Initial Configuration.pdf*” ，它描述了寄存器设置初始化选项。

图 11–1. 低电压检测器



## 11.2. 省电模式

## 11.3. 空闲模式

置位 PCON 的 IDL 位进入空闲模式。空闲模式会停止内部 CPU 的时钟。CPU 状态全面的被保存，包括 RAM，堆栈指针，程序计数器，程序状态字和累加器。端口引脚保持在空闲模式被激活的那一刻的逻辑状态。空闲模式让外围设备继续运行，以允许它们产生中断时去唤醒 CPU。Timer 0, Timer 1, UART, SPI 和 LVD 将在空闲模式下继续工作。PCA 定时器和 WDT 在空闲模式下有条件的使能去唤醒 CPU。任何使能的中断源或复位都可以终止空闲模式。当使用一个中断退出空闲模式时，中断将立即被处理，紧跟 RETI 指令之后，下一条将被执行的指令是芯片进入空闲模式的那条指令之后的那一条指令。

应当注意当空闲模式被硬件复位终止时，芯片从它停止的地方恢复正常程序运行，在内部复位算法采取控制之前需两个机器周期以上。这一事件里，片上硬件会抑制内部 RAM 的访问，但不会抑制对端口引脚的操作。为了清除在复位终止空闲模式时对端口引脚的一些意外的写操作的可能性，紧跟着进入空闲模式指令的那条指令不应当写端口引脚或访问外部存储器。

选择空闲模式降低功耗是通过编程 CKS2~CKS0 位（在 PCON2 寄存器里，见“8 系统时钟”）为非 000 值来减缓 MCU 的工作速度。用户应该检查哪个程序段适合较低的操作速度。大体上，较低的运行速度应当不影响系统的一般功能。然后，由硬件恢复到一般速度进入普通模式。

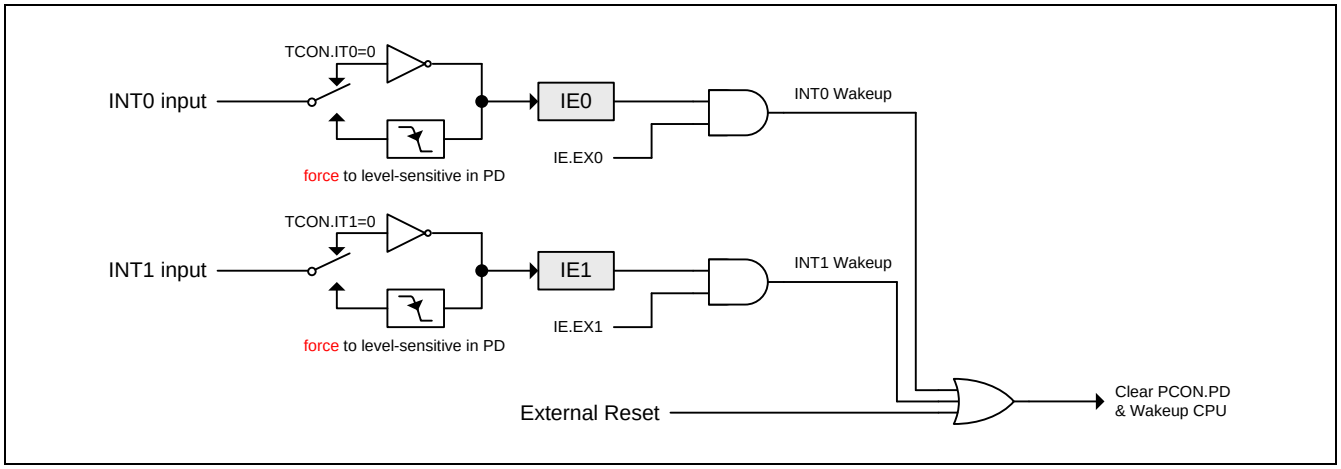
## 11.4. 掉电模式

置位 PCON 的 PD 位进入掉电模式。掉电模式下振荡停止并且 FLASH 存储器掉电，以达到最小化功耗。掉电模式下仅上电电路会继续耗电。在掉电期间，电源电压可以降低到维持 RAM 活性的电压。RAM 的内部被保存；然而，VDD 一但下降，SFR 内容不一定能被保持。掉电模式可以被外部复位，上电复位和使能的外部中断退出。

下列情况之一发生之后的 4uS 时间内用户不应当试着进入（或重复进入）掉电模式：开始执行代码（任何复位之后），或从掉电模式退出。

图 11–2 显示了 **MA801_MA802** 掉电模式的唤醒机制

图 11–2. 掉电模式唤醒结构



### 11.4.1. 中断唤醒掉电模式

两个外部中断可以被配置去终止掉电模式。外部中断 INT0 (P3.2), INT1 (P3.3) 可以用来退出掉电模式。为了能唤醒掉电模式， 中断 nINT0, nINT1 必须使能并且设置为电平触发操作。如果外部中断使能且设置是边沿触发（下降沿），他们会被硬件强置为电平触发（低电平）。.

一个中断终止掉电模式，唤醒时间取决内部定时。当中断引脚产生下降沿时 ， 掉电模式被终止，震荡重新启动，并且一个内部计数器开始计数， 在内部计数器没有计满之前内部时钟不允许被应用 CPU 也不能运行指令。 计数溢出后， 中断服务程序开始工作， 为了避免中断被重复触发， 中断服务程序在返回前应该被禁止，中断引脚应保持足够长的低电平时间直到芯片计时溢出并且开始执行程序。

### 11.4.2. 复位唤醒掉电模式

对于 RST 输入引脚，通过外部复位唤醒掉电模式，有点类似于中断唤醒。在 RST 的上升沿后，掉电模式退出，振荡开始起振，并且一个内部定时器开始计数。内部时钟在这个定时器计满之前不被允许应用到 CPU 单元上。复位引脚上的高电平应保持比定时器溢出周期更长的时间以确保芯片复位正确。一旦 RST 引脚变低电平，芯片将开始执行程序。

## 11.5. 电源控制寄存器

PCON: 电源控制寄存器
SFR 地址 = 0x87 上电复位值 = 0011-0000, 复位值 = 000x-0000

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

Bit 1: PD, 掉电控制位
0: 软件清零或任何一个退出掉电模式的事件发生时硬件清零。
1: 置位则激活掉电操作（即进入掉电模式）。

Bit 0: IDL, 空闲模式控制位
0: 软件清零或任何一个退出空闲模式的事件发生时硬件清零。
1: 置位则激活空闲操作（即进入空闲模式）。

PCON2: 电源控制寄存器 2
SFR 地址 = 0xC7 复位值 = xxxx-x000

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

Bit 7~3: 保留. 当写 PCON2 时，软件必须在这些位上写“0”

Bit 2~0: CKS2 ~ CKS0, 在**空闲模式**下可编程系统时钟选项。CKS[2:0] 缺省值“000” 选择系统时钟是 OSCin/1。

| CKS[2:0] | 空闲模式下系统时钟 |
|---|---|
| 0 0 0 | OSCin/1 |
| 0 0 1 | OSCin /2 |
| 0 1 0 | OSCin /4 |
| 0 1 1 | OSCin /8 |
| 1 0 0 | OSCin /16 |
| 1 0 1 | OSCin /32 |
| 1 1 0 | OSCin /64 |
| 1 1 1 | OSCin /128 |

## 11.6. 电源控制示例代码

(1) 功能需求: 选择系统时钟分频为 OSCin/128 的空闲模式 (默认为 OSCin/1)

汇编语言代码范例:

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

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

C 语言代码范例:

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

   PCON2 |= (CKS2 | CKS1 | CKS0);        // 系统时钟分频 /128
                                // CKS[2:0], 系统时钟分频
                                //   0    | OSCin/1
                                //   1    | OSCin/2
                                //   2    | OSCin/4
                                //   3    | OSCin/8
                                //   4    | OSCin/16
                                //   5    | OSCin/32
                                //   6    | OSCin/64
                                //   7    | OSCin/128
```

# 12. I/O 端口配置

**MA801_MA802** 有下列 I/O 端口: P1.0~P1.7, P3.0~P3.5 和 P3.7。 准确的可用 I/O 引脚数量由封装类型决定。见表 12–1.

表 12–1. 可用 I/O 引脚数量

| 封装类型 | I/O 引脚 | 引脚数量 |
|---|---|---|
| 20 脚 PDIP/SOP/TSSOP | P1.0~P1.7, P3.0~P3.5, P3.7 | 15 |

## 12.1. IO 结构

**MA801_MA802** 的 I/O 操作模式支持 4 种配置类型。它们是：准双向口（标准 8051 I/O 口），推挽输出，仅输入（高阻输入）和漏极开路输出。

下面描述这四种类型的 I/O 模式的配置

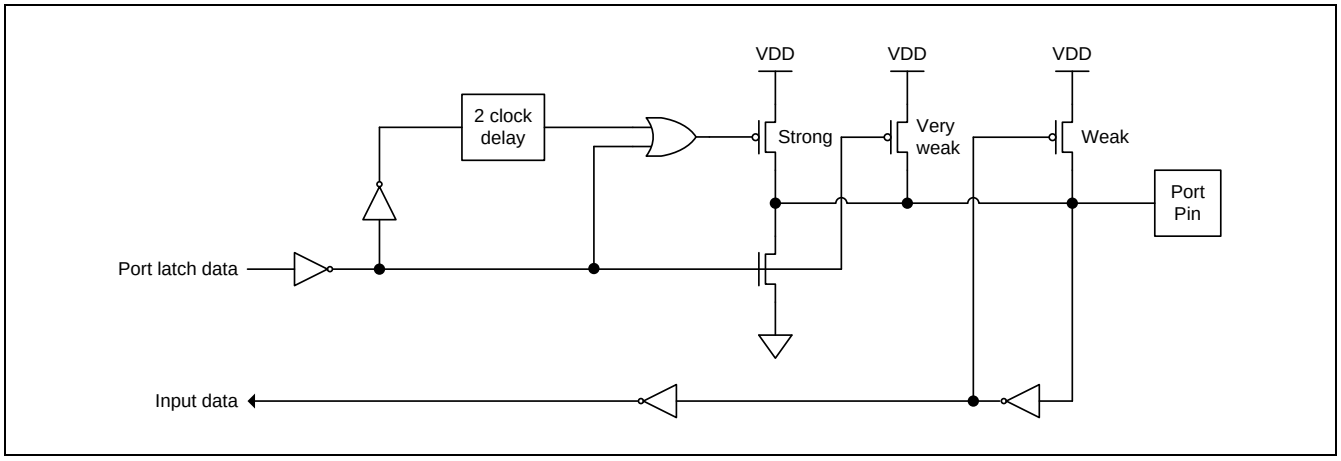
## 12.2. 准双向口结构

所有工作在准双向模式时的引脚与标准 8051 端口引脚类似。一个准双向端口用作输入和输出时不需要对端口重新配置。这是因为端口输出逻辑高时，弱上拉，允许外部器件拉低引脚。当输出低时，强的驱动能力可吸收大电流。在准双向输出时有三个上拉晶体管用于不同的目的。

其中的一种上拉，称为微上拉，只要端口寄存器的引脚包含逻辑 1 则打开。如果引脚悬空，则这种非常弱上拉提供一个非常小的电流将引脚拉高。第二种上拉称为“弱上拉”，端口寄存器的引脚包含逻辑 1 时且引脚自身也在逻辑电平时打开。这种上拉对准双向引脚提供主要的电流源输出为 1。如果引脚被外部器件拉低，这个弱上拉关闭，只剩一个微上拉。为了在这种条件下将引脚拉低，外部器件不得不吸收超过弱上拉功率的电流，且拉低引脚在输入的极限电压之下。第三种上拉称为“强”上拉。这种上拉用于加速准双向端口的上升沿跳变，当端口寄存器发生从逻辑 0 到逻辑 1 跳变时，强上拉打开两个 CPU 时钟，快速将端口引脚拉高。

准双向口配置如图 12–1.所示

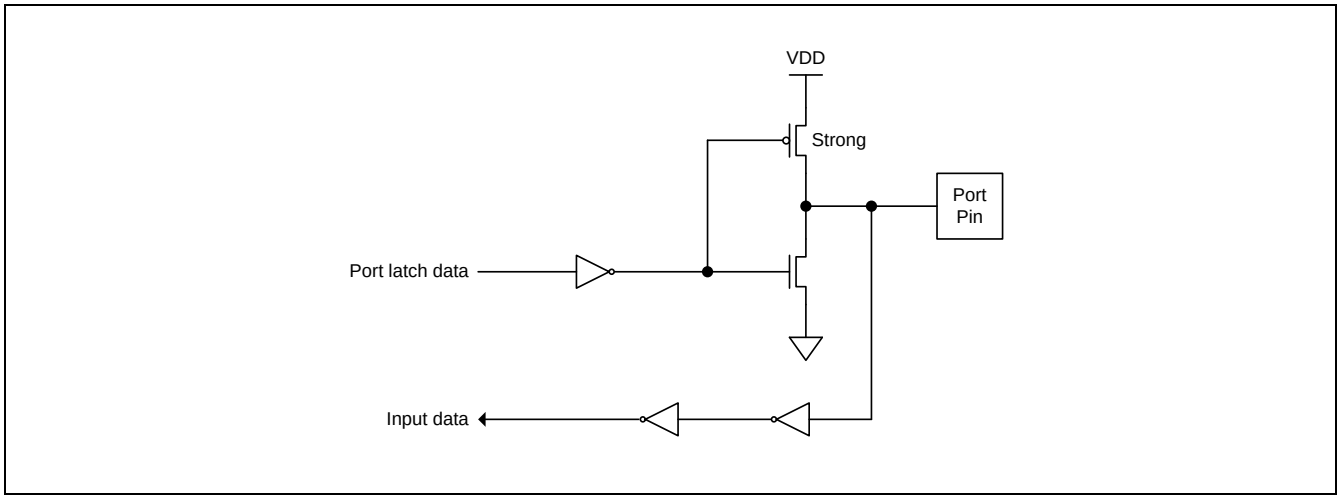
图 12–1. 准双向 I/O




## 12.3. 推挽输出结构

推挽输出配置与开漏输出、准双向输出模式有着相同的下拉结构，但是当端口寄存器包含逻辑 1 时提供一个连续的强上拉。当一个端口输出需要更大的电流时可配置为推挽输出模式。另外，在这种配置下端口的输入路径与准双向模式相同。
推挽输出配置见图 12–2.

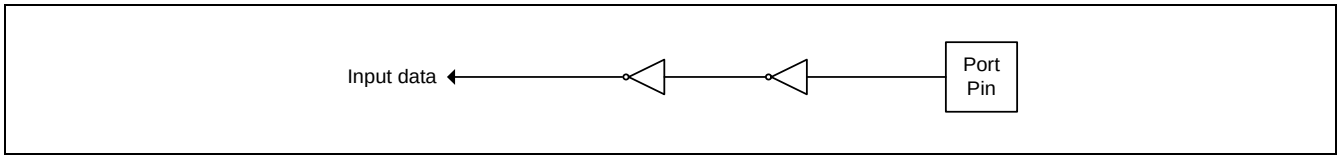
图 12–2. 推挽输出




## 12.4. 仅输入（高阻抗输入）模式结构

仅输入配置在引脚上没有任何上拉电阻，如图 12–3 所示。

图 12–3. 仅输入模式

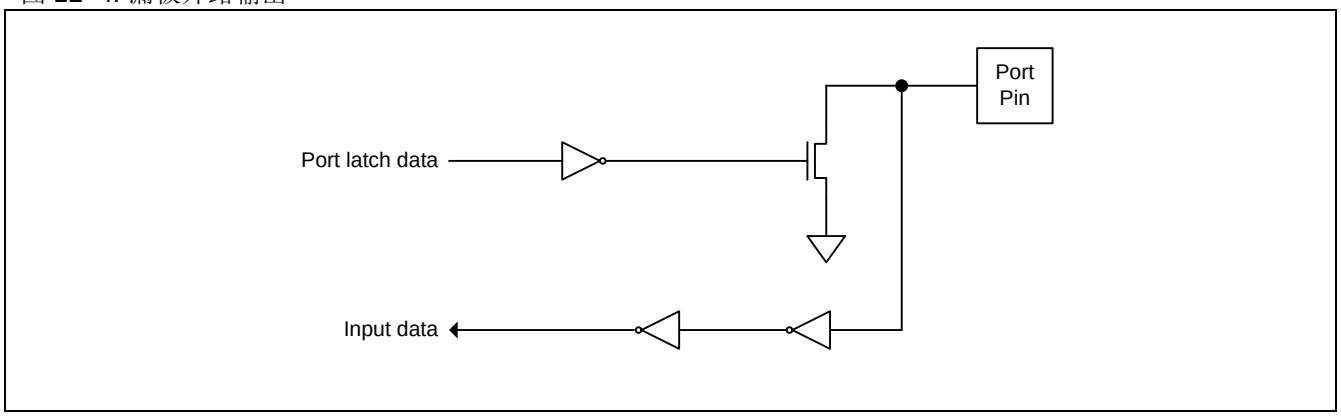


### 12.5. 3 漏极开路输出结构

开漏输出时，当端口寄存器包含逻辑 0 时，关闭所有上拉，只有端口引脚的下拉晶体管。在应用中使用这个配置，端口引脚必须有外部上拉，典型的是将电阻接到 VDD。这个模式的下拉和准双向端口的模式相同。另外，在这种配置下端口的输入路径与准双向模式相同。

漏极开路输出模式配置如 图 12–4.

图 12–4. 漏极开路输出



## 12.6. I/O 端口寄存器

**MA801_MA802** 所有的 I/O 端口都可以单独的由软件按位配置成 4 种模式中的一种。如表 12–2 所示。两个模式寄存器用来选择每个 I/O 引脚的输出类型。

表 12–2. I/O 端口配置设定

| PxM0.y | PxM1.y | 端口模式 |
|---|---|---|
| 0 | 0 | 准双向(默认) |
| 0 | 1 | 推挽输出 |
| 1 | 0 | 仅输入 (高阻输入) |
| 1 | 1 | 漏极开路输出 |

这里 x=1 或 3 (端口号)， y=0~7 (端口引脚).。寄存器 PxM0 和 PxM1 列举了每个引脚的描述。

## 12.7. 端口 1 寄存器

P1: 端口 1 寄存器

SFR 地址 = 0x90 复位值 = 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 可以由 CPU 置位或清零.

P1M0: 端口 1 模式寄存器 0

SFR 地址 = 0x91 POR+复位值 = 0000-0000

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

P1M1: 端口 1 模式寄存器 1

SFR 地址 = 0x92 POR+复位值 = 0000-0000

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

## 12.8. 端口 3 寄存器

P3: 端口 3 寄存器

SFR 地址 = 0xB0 复位值 = 1x11-1111

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

Bit 7: P3.7 可以由 CPU 置位或清零.

Bit 6: 保留. 当写 P3 时，软件必须在这位上写“0”。

Bit 5~0: P3.7~P3.0 可以由 CPU 置位或清零.

P3M0: 端口 3 模式寄存器 0

SFR 地址 = 0xB1 复位值 = 0x00-0000

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

Bit 5: 保留. 当写 P3M0 时，软件必须在这位上写“0”。

P3M1: 端口 3 模式寄存器 1

SFR 地址 = 0xB2 复位值 = 0000-0000

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

Bit 5: 保留. 当写 P3M1 时，软件必须在这位上写“0”。

## 12.9. GPIO 示例代码

(1). 功能需求: 设置 P1.0 为仅输入模式

汇编语言代码范例:

```
P1Mn0           EQU             01h

   ORL    P1M0, #P1Mn0
   ANL    P1M1, #(0FFh + P1Mn0)        ; 配置 P1.0 为仅输入模式
   SETB   P1.0                         ; 设置 P1.0 数据为“1”而使能输入模式
```

C 语言代码范例:

```
#define P1Mn0           0x01

   P1M0 |= P1Mn0;
   P1M1 &= ~P1Mn0;                     //配置 P1.0 为仅输入模式
   P10 = 1;                            //设置 P1.0 数据为“1”而使能输入模式
```

(2). 功能需求: 设置 P1.0 为推挽输出

汇编语言代码范例:

```
P1Mn0           EQU             01h

   ANL    P1M0, #(0FFh - P1Mn0)
   ORL    P1M1, #P1Mn0             ; /配置 P1.0 为推挽输出
   SETB   P1.0
```

C 语言代码范例:

```
#define P1Mn0           0x01

   P1M0 &= ~P1Mn0;
   P1M1 |= P1Mn0;                      //配置 P1.0 为推挽输出
   P10 = 1;                            // P10 输出高电平
```

(3). 功能需求: 设置 P1.0 为漏极开路输出模式

汇编语言代码范例:

```
P1Mn0           EQU             01h

   ORL    P1M0, #P1Mn0
   ORL    P1M1, #P1Mn0              ; 配置 P1.0 为漏极开路输出
   SETB   P1.0                      ; 设置 P1.0 数据为“1”而使能漏极开路输出模式
```

C 语言代码范例:

```
#define P1Mn0           0x01

   P1M0 |= P1Mn0;
   P1M1 |= P1Mn0;                      //配置 P1.0 为漏极开路输出
   P10 = 1;                            //设置 P1.0 数据为“1”而使能漏极开路输出模式
```

# 13. 中断

**MA801_MA802** 有 7 个中断源，支持 4 级中断结构。与这四级中断有关联的特殊功能寄存器有 IE、IP、IPH 和 AUXR。 IPH (中断优先级高位) 寄存器设置四级中断优先级。四级中断优先级结构为中断源的应用提供了很大的便利。

## 13.1. 中断结构

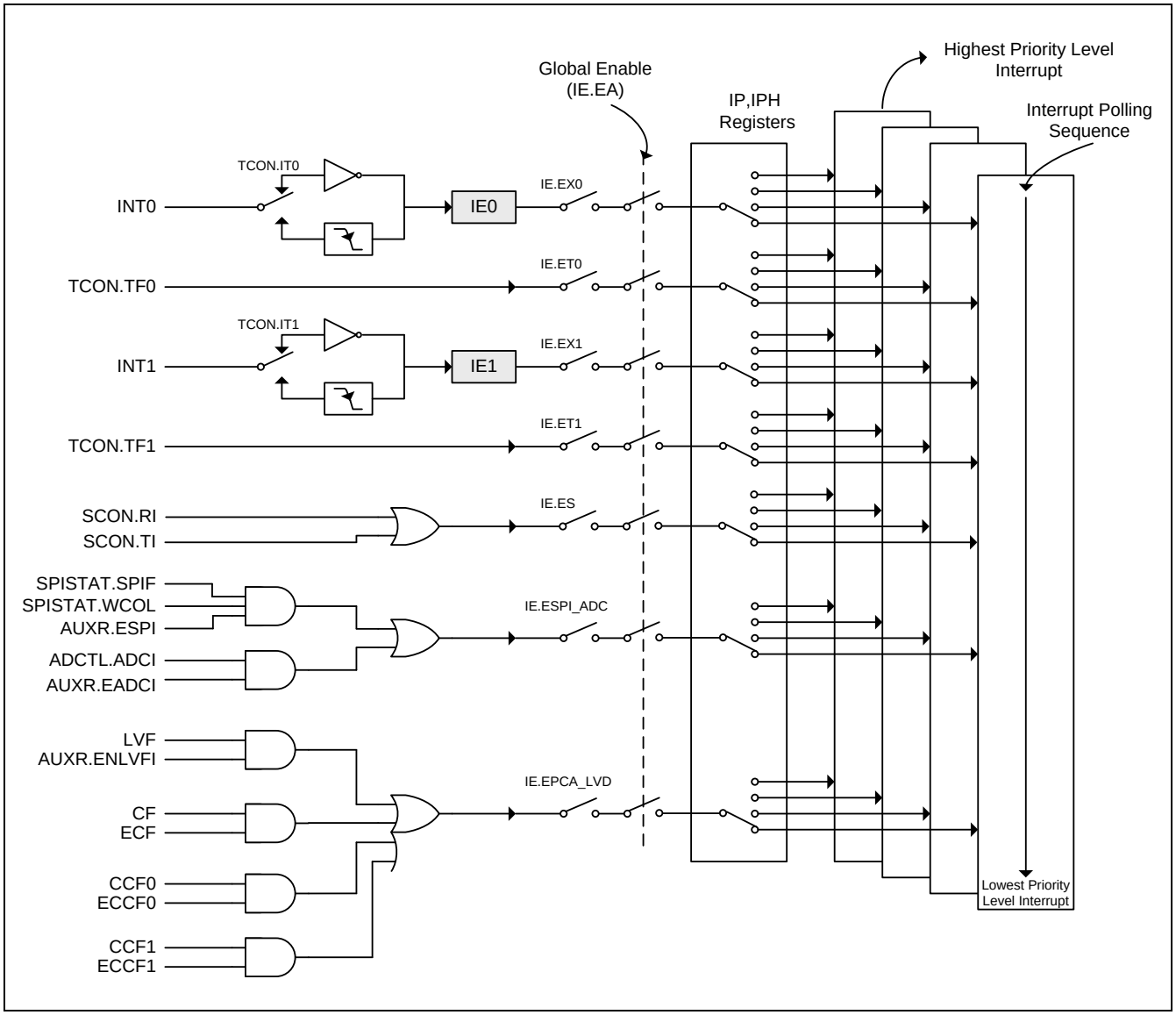
表 13–1 列出了所有的中断源。使能位被允许，中断请求时硬件会产生一个中断请求标志，当然，总中断使能位 EA ( IE 寄存器)必须使能。中断请求位能由软件置 1 或清零，这和硬件置 1 或清零结果相同。同理，中断可以由软件产生或取消，中断优先级位决定每个中断产生的优先级，多个中断同时产生时依照中断优先级顺序处理。中断向量地址表示中断服务程序的入口地址。

图 13–1 展示了整个中断系统。每一个中断将在下面部分做简单的描述。

表 13–1. 中断源

| 序号 | 中断源 | 使能位 | 请求位 | 优先级位 | 优先级顺序 | 向量地址 |
|---|---|---|---|---|---|---|
| #1 | 外部中断 0, INT0 | EX0 | IE0 | [ PX0H, PX0 ] | (最高) | 0003H |
| #2 | 定时器 0 | ET0 | TF0 | [ PT0H, PT0 ] | … | 000Bh |
| #3 | 外部中断 1, INT1 | EX1 | IE1 | [ PX1H, PX1 ] | ... | 0013H |
| #4 | 定时器 1 | ET1 | TF1 | [ PT1H, PT1 ] | … | 001BH |
| #5 | 串口 | ES | RI, TI | [ PSH, PS ] | … | 0023H |
| #6 | SPI/ADC | ESPI_ADC & (ESPI, EADCI) | SPIF, WCOL, ADCI | [ PSPIH_ADC, PSPI_ADC ] | … | 002BH |
| #7 | PCA/LVF | EPCA_LVD & (ECF, ECCFn, ENLVFI) | CF CCFn n=0~1) LVF | [ PPCAH_LVD, PPCA_LVD ] | (最低) | 0033H |

图 13–1. 中断系统


Global Enable
(IE.EA)
IP,IPH
Registers
Highest Priority Level
Interrupt
Interrupt Polling
Sequence
TCON.IT0
INT0
IE0
IE.EX0
IE.ET0
TCON.TF0
TCON.IT1
INT1
IE1
IE.EX1
IE.ET1
TCON.TF1
IE.ES
SCON.RI
SCON.TI
SPISTAT.SPIF
SPISTAT.WCOL
AUXR.ESPI
IE.ESPI_ADC
ADCTL.ADCI
AUXR.EADCI
LVF
AUXR.ENLVFI
IE.EPCA_LVD
CF
ECF
CCF0
ECCF0
CCF1
ECCF1
Lowest Priority
Level Interrupt

## 13.2. 中断源

表 13–2. 中断标志

| 序号 | 中断源名称 | 中断请求位 | 位位置 |
|---|---|---|---|
| #1 | 外部中断 0, INT0 | IE0 | TCON.1 |
| #2 | 定时器 0 | TF0 | TCON.5 |
| #3 | 外部中断 1, INT1 | IE1 | TCON.3 |
| #4 | 定时器 1 | TF1 | TCON.7 |
| #5 | Serial Port | RI<br>TI | SCON.0<br>SCON.1 |
| #6 | SPI/ADC | SPIF, WCOL,<br>ADCI | SPISTAT.7~6<br>ADCTL.4 |
| #7 | PCA/LVF | CF,<br>CCFn (n=0~1),<br>LVF | CCON.7<br>CCON.1~0<br>PCON.5 |

外部中断 nINT0 和 nINT1 分别通过 TCON 的 IT0 和 IT1 可以设置成电平触发或边沿触发。实际产生的中断标志位是 TCON 的 IE0 和 IE1。产生外部中断时，如果是边沿触发，进入中断服务程序后由硬件清除中断标志位，如果中断是电平触发，由外部请求源而不是由片内硬件控制请求标志。

定时 0 和定时器 1 中断由 TF0 和 TF1 产生，在大多数情况下，由各自的定时器/计数器翻转而置位。当产生定时器中断时，进入中断服务程序后由片内硬件清除标志位。

串口中断由 RI 和 TI 的逻辑产生。进入中断服务程序后，这些标志均不会被硬件清除。实际上，中断服务程序通常需要确定是由 R I 还是 TI 产生的中断，然后由软件清除中断标志。

SPI 中断（SPIF 和 WCOL）和 ADC 中断（ADCI）共享 SPI/ADC 中断。进入中断服务程序后，这些标志不会被硬件清除。服务程序应当轮询它们以决定是哪一个请求服务，并由软件清除这个标志。

PCA 中断（CF，CCF0 和 CCF1）和 LVD（低压检测器）中断（LVF 在 PCON.5）共享 PCA/LVF 中断。进入中断服务程序后，这些标志不会被硬件清除。服务程序应当轮询它们以决定是哪一个请求服务，并由软件清除这个标志。

## 13.3. 中断使能

表 13–3. 中断使能控制

| 序号 | 中断源名称 | 请求位 | 使能位 | 位位置 |
|---|---|---|---|---|
| #1 | 外部中断 0, INT0 | IE0 | EX0 | IE.0 |
| #2 | 定时器 0 | TF0 | ET0 | IE.1 |
| #3 | 外部中断 1, INT1 | IE1 | EX1 | IE.2 |
| #4 | 定时器 1 | TF1 | ET1 | IE.3 |
| #5 | 串口 | RI<br>TI | ES | IE.4 |
| #6 | SPI/ADC | SPIF,<br>WCOL<br>ADCI | ESPI_ADC &<br>(ESPI,<br>EADCI) | IE.5,<br>AUXR.3,<br>AUXR.4 |
| #7 | PCA/LVF | CF,<br>CCF0,<br>CCF1,<br>LVF | EPCA_LVD &<br>(ECF,<br>ECCF0,<br>ECCF1,<br>ENLVFI) | IE.6,<br>CMOD.0,<br>CCAPM0.0,<br>CCAPM1.0,<br>AUXR.2 |

**MA801_MA802** 有 7 个可用的中断源。通过设置寄存器 IE 能对每个中断进行使能和禁止操作。IE 也包括了一个全局使能位，EA，清它可以立刻禁止所有中断。如果 EA 置“1”， 中断单独的被相应的使能位使能或禁止。EA 清“0”，所有中断被禁止

## 13.4. 中断优先级

中断服务的优先级组合和 80C51 是一样的，除了有 4 个优先级比 80C51 多两个外。优先级设置位(见表 13–1 决定每个中断的优先级别。IP 和 IPH 组合成 4 级优先级中断。表 13–4 列出了设置位的值和每个组合对应的优先级

表 13–4. 中断优先级

| {IPH.x , IP.x} | 优先级 |
|---|---|
| 11 | 1 (最高) |
| 10 | 2 |
| 01 | 3 |
| 00 | 4 |

每一个中断源都有两个相对应的位表示它的优先级。一个在 IPH 寄存器 ，另一个在 IP 寄存器。高优先级中断不会被低优先级中断打断。如果两个如果同时收到两个不同优先级的中断请求，高优先级的请求将被处理。如果同时收到同样优先级的请求，内部轮询顺序将决定哪个请求被处理。表 13–2 列出了同优先级的内部轮询顺序和中断向量地址。

## 13.5. 中断处理

每个系统时钟周期都会采样中断标志位。在下一个系统时钟，这采样值被轮询。如果在第一个周期其中一个标志被置位，那么第二个周期（轮询周期）就会发现它，并且中断系统会产生一个硬件 LCALL 调用相应的的中断服务程序，只要没有被下列的条件给阻止：

***阻止条件:***

- 一个同等或高优先级的中断正在处理。
- 当前周期(轮询周期)不是正在执行的指令的最后一个周期。
- 正在执行指令 RETI 或正在写和中断相关的寄存器( IE, IP, IPH 寄存器)。

上述三种情况将锁定 LCALL 指令使之暂时不能访问中断服务程序；第二种情况在引导任何中断服务程序之前必须执行完；第三种情况保证 RETI 的执行或写中断相关的寄存器（如 IE 或 IP 等）能顺利完成，在中断进入引导程序之前至少运行一个以上的指令周期

## 13.6. 中断寄存器

TCON: 定时器/计数器控制寄存器
SFR 地址 = 0x88 复位值 = 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, 外部中断 1 触发标志。
0: 如果是边沿触发的中断则在进入中断向量后硬件清零。
1: 检测到外部中断 1 触发时由硬件置位 (边缘或电平触发).

Bit 2: IT1: 外部中断 1 类型控制位
0: 软件清零指定由低电平触发外部中断 1
1: 软件置位指定由下降沿触发外部中断 1

Bit 1: IE0, 外部中断 0 触发标志。
0: 如果是边沿触发的中断则在进入中断向量后硬件清零。
1: 检测到外部中断 0 触发时由硬件置位 (边缘或电平触发).

Bit 0: IT0: Interrupt 0 Type control bit.
0: 软件清零指定由低电平触发外部中断 0
1: 软件置位指定由下降沿触发外部中断 0

IE: 中断使能标志
SFR 地址 = 0xE8 复位值 = 0000-0000

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

Bit 7: EA, 总中断使能位
0: 全局禁止所有中断。
1: 全局使能所有中断

Bit 6: EPCA_LVD, PCA 和 LVD（低压检测器）组合中断使能位
0: 禁止 PCA 和 LVD 组合中断
1: 使能 PCA 和 LVD 组合中断

Bit 5: ESPI_ADC, SPI 和 ADC 组合中断使能位
0: 禁止 SPI 和 ADC 组合中断
1: 使能 SPI 和 ADC 组合中断

Bit 4: ES, 串口中断使能位
0: 禁止串口中断
1: 使能串口中断

Bit 3: ET1, 定时器 1 中断使能位
0: 禁止定时器 1 中断
1: 使能定时器 1 中断

Bit 2: EX1, 外部中断 1 中断使能位
0: 禁止外部中断 1
1: 使能外部中断 1

Bit 1: ET0, 定时器 0 中断使能位
0: 禁止定时器 0 中断
1: 使能定时器 0 中断

Bit 0: EX0, 外部中断 0 中断使能位
0: 禁止外部中断 0
1: 使能外部中断 0

AUXR: 辅助寄存器
SFR 地址 = 0x8E 复位值 = 0000-00xx

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

Bit 4: EADCI, ADC 中断使能位
0: 禁止 ADC 中断
1: 使能 ADC 中断

Bit 3: ESPI, SPI 中断使能位
0: 禁止 SPI 中断
1: 使能 SPI 中断

Bit 2: ENLVFI, LVD 中断使能位
0: 禁止 LVD（LVF）中断
1: 使能 LVD（LVF）中断

IP: 中断优先级低位寄存器
SFR 地址 = 0xB8 复位值 = xx00-0000

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

Bit 7: 保留. 当写 IP 时，软件必须在这位上写“0”
Bit 6: PPCA_LVD, PCA 和 LVD 中断优先级低位寄存器
Bit 5: PSPI_ADC, SPI 和 ADC 中断优先级低位寄存器
Bit 4: PS, 串口中断优先级低位寄存器.
Bit 3: PT1, 定时器 1 中断优先级低位寄存器.
Bit 2: PX1, 外部中断 1 优先级低位寄存器.
Bit 1: PT0, 定时器 0 中断优先级低位寄存器.
Bit 0: PX0, 外部中断 0 优先级低位寄存器.

IPH: 中断优先级高位寄存器
SFR 地址 = 0xB7 复位值 = xx00-0000

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

Bit 7: 保留. 当写 IPH 时，软件必须在这位上写“0”
Bit 6: PPCAH_LVD, PCA 和 LVD 中断优先级高位寄存器
Bit 5: PSPIH_ADC, SPI 和 ADC 中断优先级高位寄存器
Bit 4: PSH, 串口中断优先级高位寄存器.
Bit 3: PT1H, 定时器 1 中断优先级高位寄存器.
Bit 2: PX1H, 外部中断 1 优先级高位寄存器.
Bit 1: PT0H, 定时器 0 中断优先级高位寄存器.
Bit 0: PX0H, 外部中断 0 优先级高位寄存器..

## 13.7. 中断示例代码

(1). 功能需求: 在掉电模式下设置 INT0 高电平唤醒 MCU

汇编语言代码范例:

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

   ORG   0000h
   JMP   main

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

main:

   SETB   P3.2                  ;

   ORL    IP,#PX0               ; 选择 INT0 中断优先级
   ORL    IPH,#PX0H             ;
   JNB    P3.2, $               ; 确认 P3.2 输入高

   SETB   EX0                   ; 使能 INT0 中断
   CLR    IE0                   ; 清除 INT0 标志
   SETB   EA                    ; 使能全局中断

   ORL    PCON,#PD              ; 设置 MCU 进入掉电模式
   JMP    $
```

C 语言代码范例:

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

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

void main(void)
{
   P32 = 1;

   IP |= PX0;                   //选择 INT0 中断优先级
   IPH |= PX0H;

   while(!P32);                 //确认 P3.2 输入高

   EX0 = 1;                     //使能 INT0 中断
   IE0 = 0;                     //清除 INT0 标
   EA = 1;                      //使能全局中断

   PCON |= PD;                  //设置 MCU 进入掉电模式
   while(1);
```

# 14. 定时器/计数器

**MA801_MA802** 有两个定时器/计数器：定时器 0 和定时器 1。定时器 0/1 可以被配置成定时器或事件计数器。

在“定时器”功能中，定时器率是被预分频为每 12 个时钟周期寄存器值加 1。换句话说，它计数标准 C51 的机器周期。AUXR.T0X12 和 AUXR.T1X12 用来设置定时器 0/1 的定时器率为 1 个时钟周期。

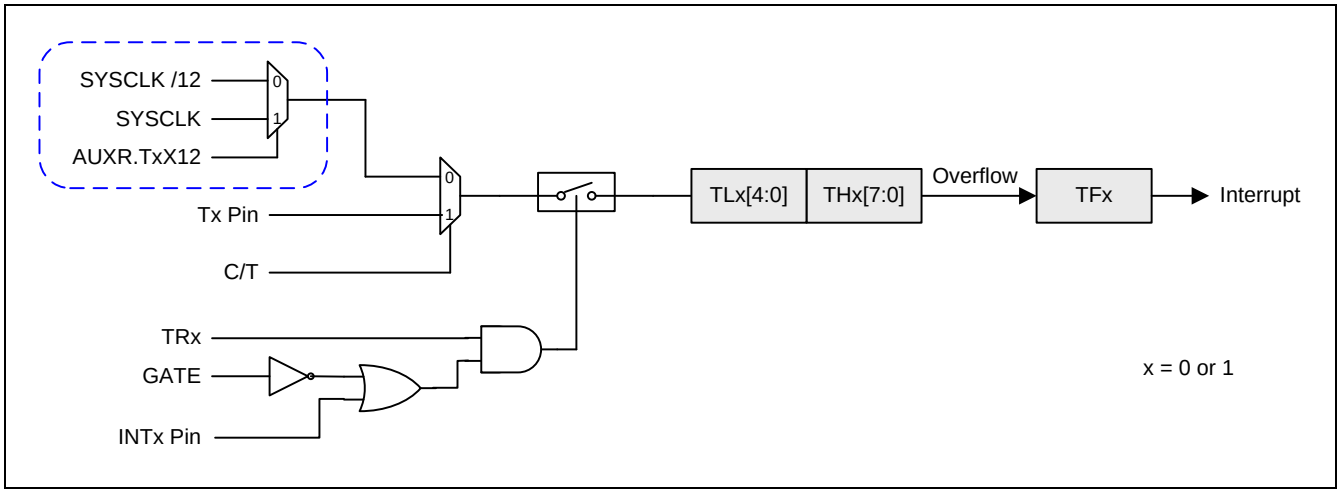
在“计数器”功能中，寄存器响应相对应的输入脚 T0 或 T1 从“1”到“0”的变化进行递增。在这个功能中，每个定时器率周期，外部输入脚被采样。当采样出现在这一周期是一高电平，在下一周期是低电平时，计数加 1。在检测到变化之后的这个周期结束后，新的计数值出现在寄存器上。

## 14.1. 定时器 0 和定时器 1

## 14.2. 模式 0 结构

定时器寄存器被配置成一个 13 位寄存器。一旦计数从全 1 变成全 0，它将置位定时器中断标志位 TFx.当 TRx=1 并且 GATE=0 或者 INTx=1 时，计数输入使能。定时器 0 和定时器 1 的模式 0 运作时相同的。定时器 0/1 的模式 0 功能见图 14–1.

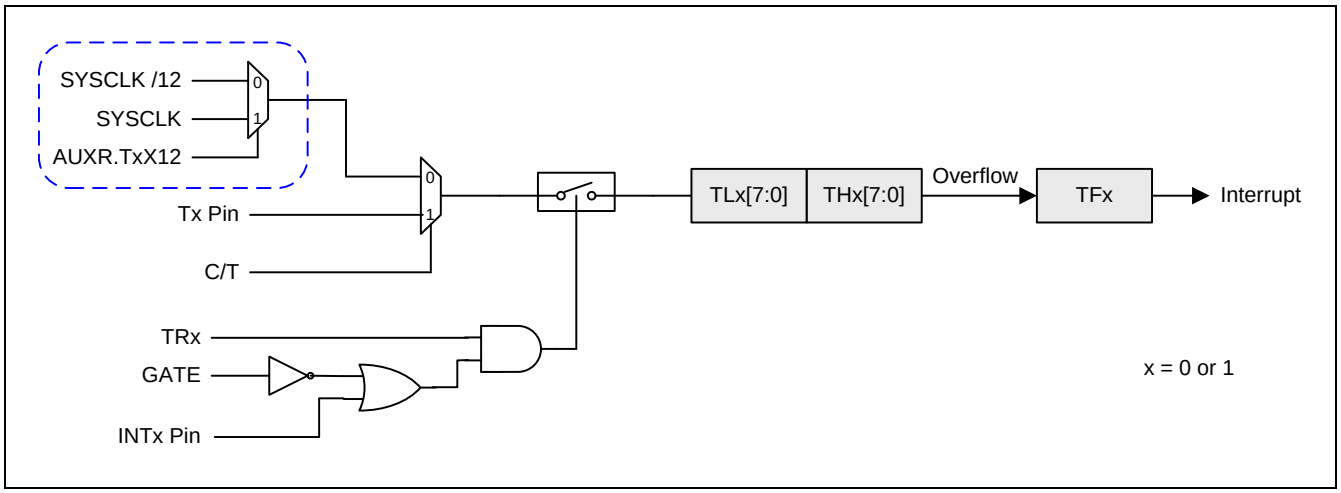
图 14–1. 定时器 0/1 模式 0 结构



### 14.2.1. 模式 1 结构

定时器 0/1 在模式 1 配置为 16 位的定时器或计数器。GATE, INTx 和 TRx 的功能与模式 0 一样。图 14–2 是定时器 0 和定时器 1 的结构图。

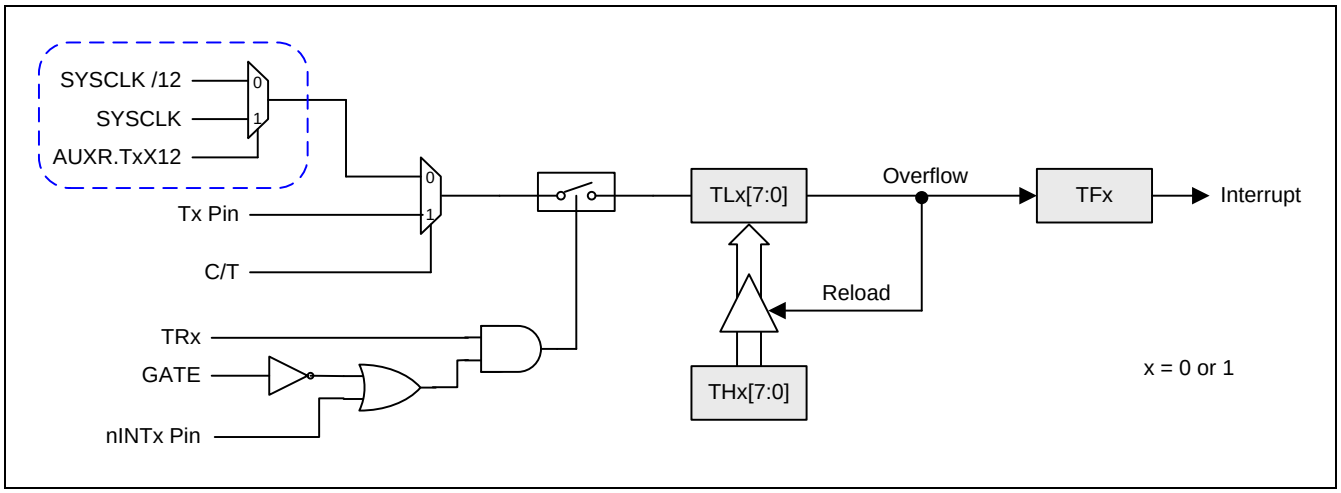
图 14–2. 定时器 0/1 模式 1 结构



### 14.2.2. 模式 2 结构

模式 2 配置定时器寄存器为一个自动加载的 8 位计数器(TLx)。TLx 溢出不仅置位 TFx，而且也将 THx 的内容加载到 TLx，THx 内容由软件预置，加载不会改变 THx 的值。定时器 0 和定时器 1 的模式 2 运作时相同的。

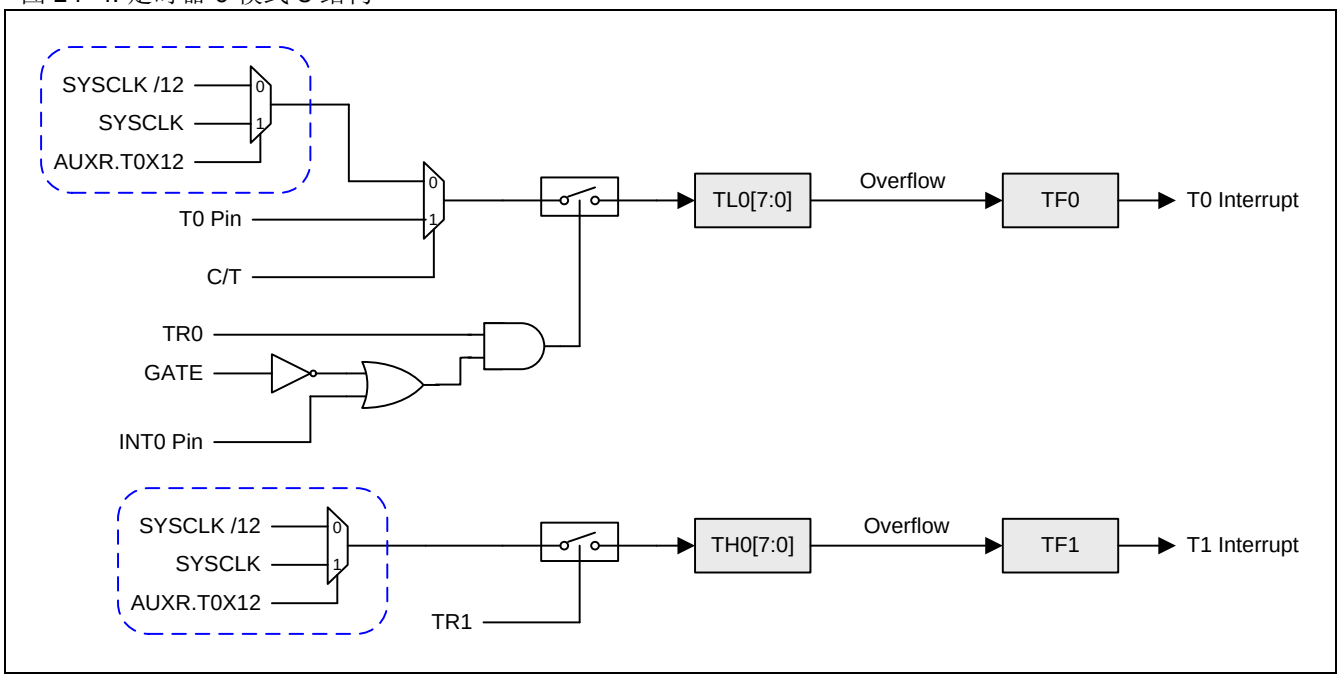
图 14–3. 定时器 0/1 模式 2 结构



### 14.2.3. 模式 3 结构

定时器 1 在模式 3 保持计数值。效果和设置 TR1=1 一样。定时器 0 在模式 3 建立 TL0 和 TH0 两个独立的计数器。TL0 使用定时器 0 控制位：C/T、GATE、TR0、INT0 和 TF0。TH0 锁定为定时器功能(不能成为外部事件计数器)且接替 Timer1 来使用 TR1 和 TF1，TH0 控制 Timer 1 中断。

图 14–4. 定时器 0 模式 3 结构



### 14.2.4. 定时器 0/1 寄存器

TCON: 定时器/计数器控制寄存器

SFR 地址 = 0x88 复位值 = 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, 定时器 1 溢出标志位。
0: 处理器进入中断向量程序由硬件清零，或由软件清零。
1: 定时器/计数器溢出时由硬件置位，或由软件置位。

Bit 6: TR1, 定时器 1 运行控制位.
0: 软件清零关闭定时器/计数器 1。
1: 软件置位开启定时器/计数器 1。

Bit 5: TF0, 定时器 0 溢出标志位.
0: 处理器进入中断向量程序由硬件清零，或由软件清零。
1: 定时器/计数器溢出时由硬件置位，或由软件置位。

Bit 4: TR0, 定时器 0 运行控制位.
0: 软件清零关闭定时器/计数器 0
1: 软件置位打开定时器/计数器 0.

TMOD: 定时器/计数器模式控制寄存器

SFR 地址 = 0x89 复位值 = 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 |
| \|←---------- 定时器 1 ----------→\| | | | | \|←---------- 定时器 0 ----------→\| | | |

Bit 7/3: Gate, 定时器 1/0 门控制位
0: 禁止定时器 1/0 门控位
1: 使能定时器 1/0 门控位。当它置 1 时，只有在/INT1 或/INT0 引脚为高电平且 TR1 或 TR0 控制位置位时，定时器 1/0 才会使能。

Bit 6/2: C/T, 定时器或计数器功能选择
0: 清零为定时器功能，从内部系统时钟输入
1: 置位为计数器功能，从 T1/T0 引脚输入

Bit 5~4/1~0: 工作模式选择

| **M1** | **M0** | **工作模式** |
|---|---|---|
| 0 | 0 | 13 位定时器/计数器 |
| 0 | 1 | 16 位定时器/计数器 |
| 1 | 0 | 8 位自动重载定时器/计数器 |
| 1 | 1 (定时器 0) | TL0 是 8 位定时器/计数器，TH0 仅是个 8 位定时器 |
| 1 | 1 (定时器 1) | 定时器/计数器 1 停止 |

TL0: 定时器 0 低 8 位寄存器

SFR 地址 = 0x8A 复位值 = 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: 定时器 0 高 8 位寄存器

SFR 地址 = 0x8C 复位值 = 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: 定时器 1 低 8 位寄存器

SFR 地址 = 0x8B 复位值 = 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: 定时器 0 高 8 位寄存器

SFR 地址 = 0x8D 复位值 = 0000-0000

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

AUXR: 辅助寄存器

SFR 地址 = 0x8E 复位值 = 0000-00xx

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

Bit 7: T0X12, 当 C/T=0，定时器 0 时钟源选项
0:清零选择 SYSCLK/12.
1: 置位选择 SYSCLK 作为时钟源

Bit 6: T1X12, 当 C/T=0，定时器 1 时钟源选项
0: 清零选择 SYSCLK/12.
1: 置位选择 SYSCLK 作为时钟源

### 14.2.5. 定时器 0/1 示例代码

(1). 功能需求: 定时器 T0 以 10KHz 的频率唤醒空闲模式，SYSCLK = 12MHz 晶振

汇编语言代码范例:

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

   ORG   0000h
   JMP   main
   ORG   0000Bh
time0_isr:
   to do…
   RETI
main:                          ;
   MOV    TH0,#(256-100)          ; 设置定时器 0 溢出率为 = SYSCLK x 100
   MOV    TL0,#(256-100)          ;
   ANL    TMOD,#0F0h              ; 设置定时器为模式 2
   ORL    TMOD,#T0M1              ;
   CLR    TF0                     ; 清定时器 0 标志位
   ORL    IP,#PT0                   ; 选择定时器 0 中断优先级
   ORL    IPH,#PT0H                 ;

   SETB   ET0                     ; 使能定时器 0 中断
   SETB   EA                      ; 使能全局中断
   SETB   TR0                     ; 启动定时器 0 运行
   ORL    PCON,#IDL                 ; 设置 MCU 进入空闲模式
   JMP    $
```

C 语言代码范例:

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

void time0_isr(void) interrupt 1
{     To do…
}
void main(void)
{
   TH0 = TL0 = (256-100);             //设置定时器 0 溢出率为 = SYSCLK x 100
   TMOD &= 0xF0;                   // S 设置定时器为模式 2
   TMOD |= T0M1;
   TF0 = 0;                       //清定时器 0 标志位

   IP |= PT0;                      //选择定时器 0 中断优先级
   IPH |= PT0H;
   ET0 = 1;                       //使能定时器 0 中断
   EA = 1;                        //使能全局中断

   TR0 = 1;                       //启动定时器 0 运行
   PCON =IDL;                      //设置 MCU 进入空闲模式
   while(1);
}
```

(2). 功能需求: 选择定时器 0 时钟源为 SYSCLK (使能 T0X12)

汇编语言代码范例:

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

   ORG   0000h
   JMP   main

   ORG   0000Bh
time0_isr:
   to do…
   RETI

main:
   ORL   AUXR, #T0X12          ; 选择定时器 0 时钟源为 SYSCLK
   CLR   TF0                   ; 清定时器 0 标志位
   ORL   IP,#PT0               ; 选择定时器 0 中断优先级
   ORL   IPH,#PT0H             ;

   SETB  ET0                   ; 使能定时器 0 中断
   SETB  EA                    ; 使能全局中断
   MOV   TH0, #(256 - 240)     ;中断间隔 20us
   MOV   TL0, #(256 - 240)     ;

   ANL   TMOD,#0F0h            ; 设置定时器 0 为模式 2
   ORL   TMOD,#T0M1            ;

   SETB  TR0                   ; 启动定时器 0
   JMP   $
```

C 语言代码范例:

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

   AUXR |= T0X12               //选择定时器 0 时钟源为 SYSCLK
   TF0 = 0;
   IP |= PT0;                  //选择定时器 0 中断优先级
   IPH |= PT0H;

   ET0 = 1;                    //使能定时器 0 中断
   EA = 1;                     //使能全局中断

   TH0 = TL0 = (256 - 240);
   TMOD &= 0xF0;               //设置定时器 0 为模式 2
   TMOD |= T0M1;

   TR0 = 1;                    //启动定时器 0
```

# 15. 串行口(UART)

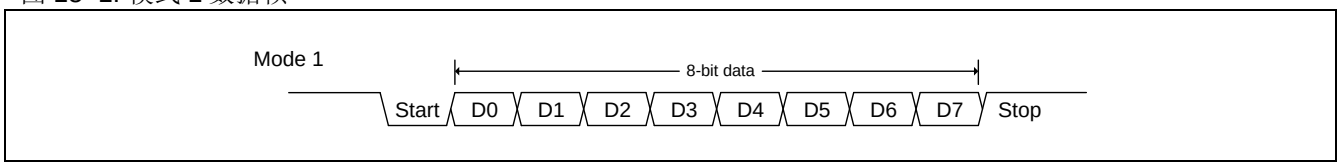
**MA801_MA802** 支持一个全双工的串行口，意思是可以同时发送和接收数据。它有一个接收缓冲，意味着在前一个接收到的字节没有从寄存器读出前，就可以开始接收第二个字节。但是，如果第一个字节在第二个字节接收完成前仍然没有被读出，则其中的一个字节将会丢失。串行口的接收和发送寄存器都通过特殊寄存器 SBUF 来访问。写到 SBUF0 加载到传送寄存器，当从 SBUF 读时是一个物理上独立分离的接收寄存器。

串行口可以工作在四种模式：模式 0 提供同步通讯同时模式 1、2 和模式 3 提供异步通讯。异步通讯作为一个全双工的通用异步收发器(UART)，可以同时发送和接收并使用不同的波特率。

**模式 0：** 8 位数据(低位先出)通过 RXD(P3.0)传送和接收。TXD(P3.1)总是作为输出移位时钟。波特率可通过 AUXR 寄存器的 URM0X6 位选择为系统时钟频率的 1/12 或 1/2。

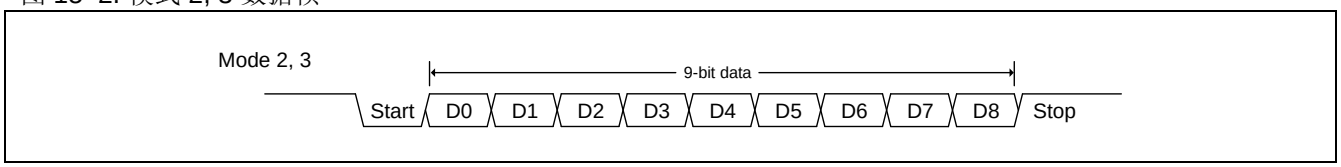
**模式 1**：10 位通过 TXD 传送或通过 RXD 接收，数据帧包括一个起始位(0)，8 个数据位(低位优先)，和一个停止位(1)见图 15–1.。在接收时，停止位进入到专用寄存器(SCON)的 RB8。波特率是可变的。

图 15–1. 模式 1 数据帧




**模式 2**：11 位通过 TXD 传送或通过 RXD 接收，数据帧包括一个起始位(0)，8 个数据位(低位优先)，一个可编程的第九个数据位和一个停止位(1) 见图 15–2.。在传送时，第 9 个数据位(TB8 在 SCON 寄存器)可以分配为 0 或者 1。在接收时，第九个数据位到 SCON 寄存器中的 RB8，同时忽略停止位。波特率可以配置为 1/32 或 1/64 的系统时钟频率。

图 15–2. 模式 2, 3 数据帧




**模式 3**：模式 3 除了波特率可变之外其它的都和模式 2 相同。

在四种模式中，使用 SBUF 作为一个目的寄存器，可以通过任何指令发起传输。在模式 0，当 RI=0 且 REN=1 时启动接收。在其它模式，在 REN=1 时，收到起始位时启动接收。

除了标准操作外，UART 还能具有侦察丢失停止位的帧错误和自动地址识别的功能。

## 15.1. 串行口模式 0

串行数据通过 RXD 读入和输出，TXD 输出移位时钟。接收和发送 8 位数据：8 个数据位(低位优先)。波特率可通过 AUXR 寄存器中的 URM0X6 选择为系统时钟的 1/12 或 1/2。图 15–3 显示了串口模式 0 的简化功能框图。

使用 SBUF 作为一个目的寄存器可通过任何指令来启动传输。“写到 SBUF”信号触发 UART 引擎开始发送。SBUF 里面的数据在 TXD（P3.1）脚的每一个上升沿移出到 RXD（P3.0）脚。八个上升沿移位时钟过后，硬件置 TI 为 1 标志发送完成。图 15–4 显示了模式 0 的传送时序图。

当 REN=1 和 RI=0 时接收启动。在下一个指令周期，RX 控制单元写 11111110 到接收移位寄存器，且在下一个时钟阶段激活接收。

接收使能将移位时钟（该时钟直接来自于 RX 时钟）输出到 P3.1 引脚。当接收激活时，在移位时钟的下降沿采样 RXD（P3.0）脚并移到寄存中。八个下降沿移位时钟过后，硬件置 RI 为 1 标志接收完成。图 15–5 显示了模式 0 的接收时序图。

图 15–3. 串口模式 0

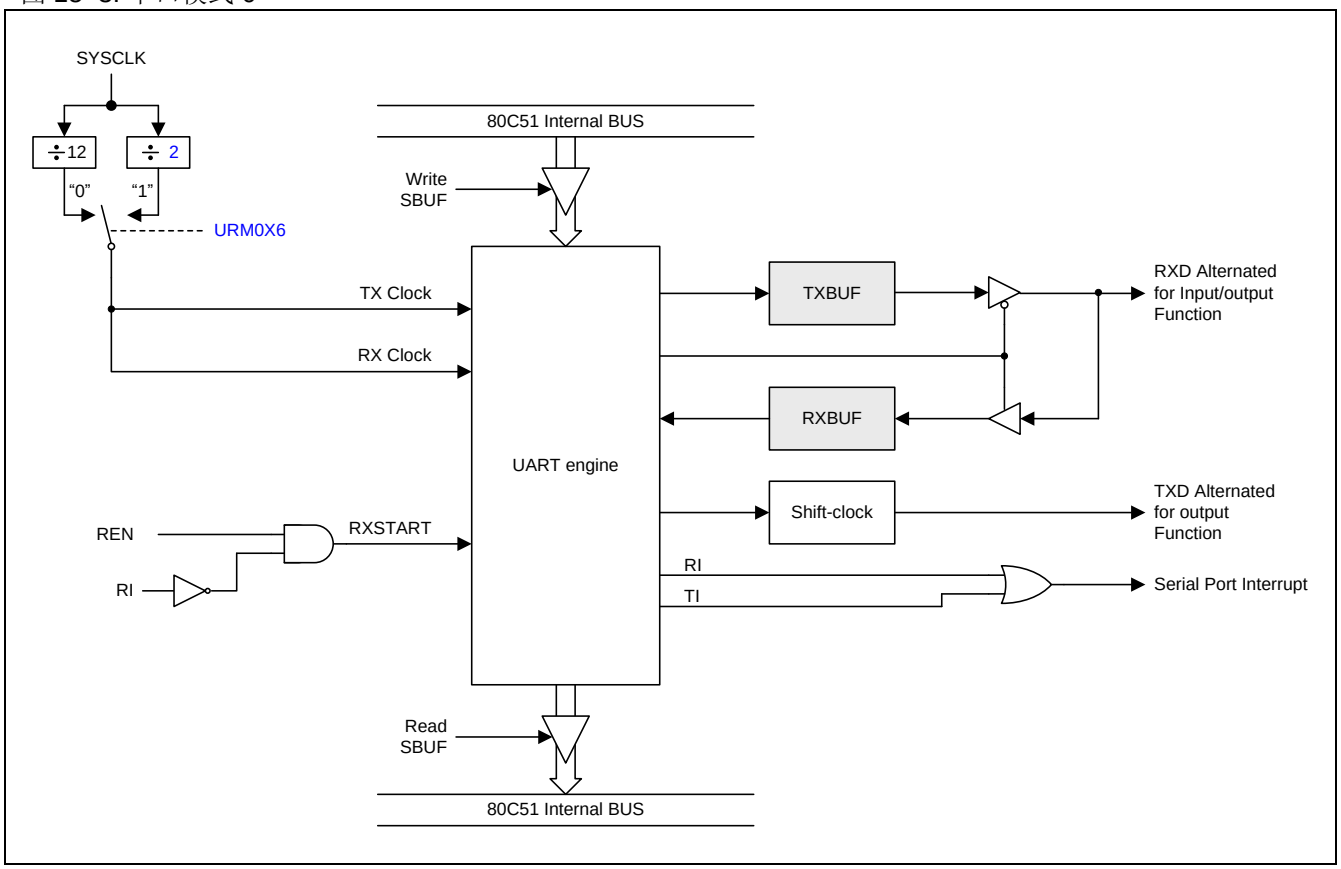


图 15–4. 模式 0 传送波形

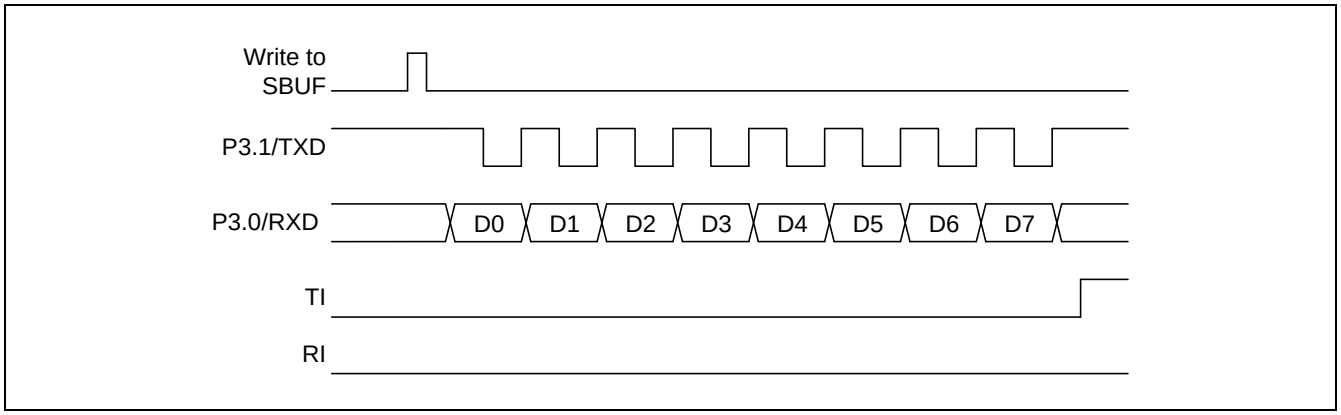
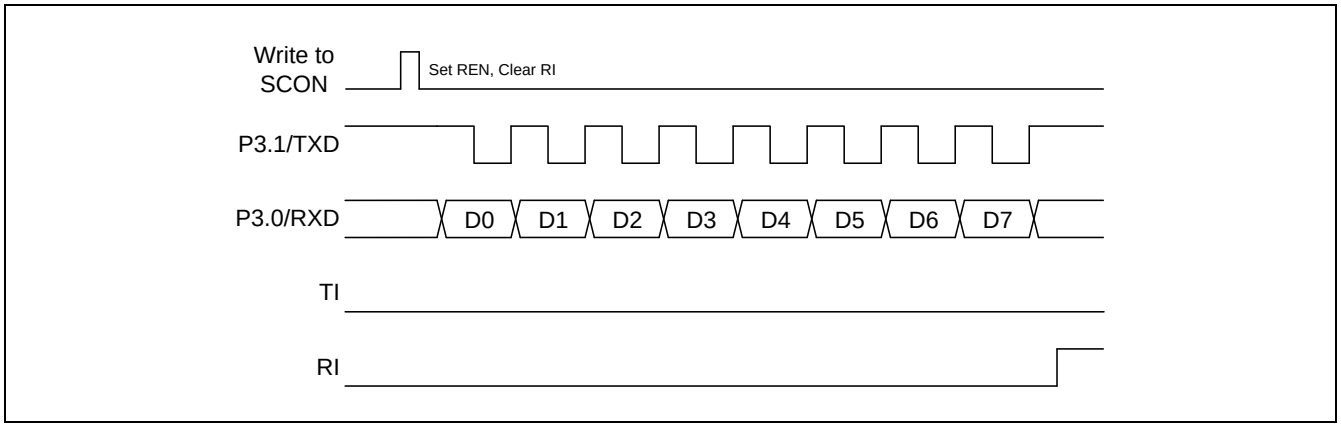



图 15–5. 模式 0 接收波形



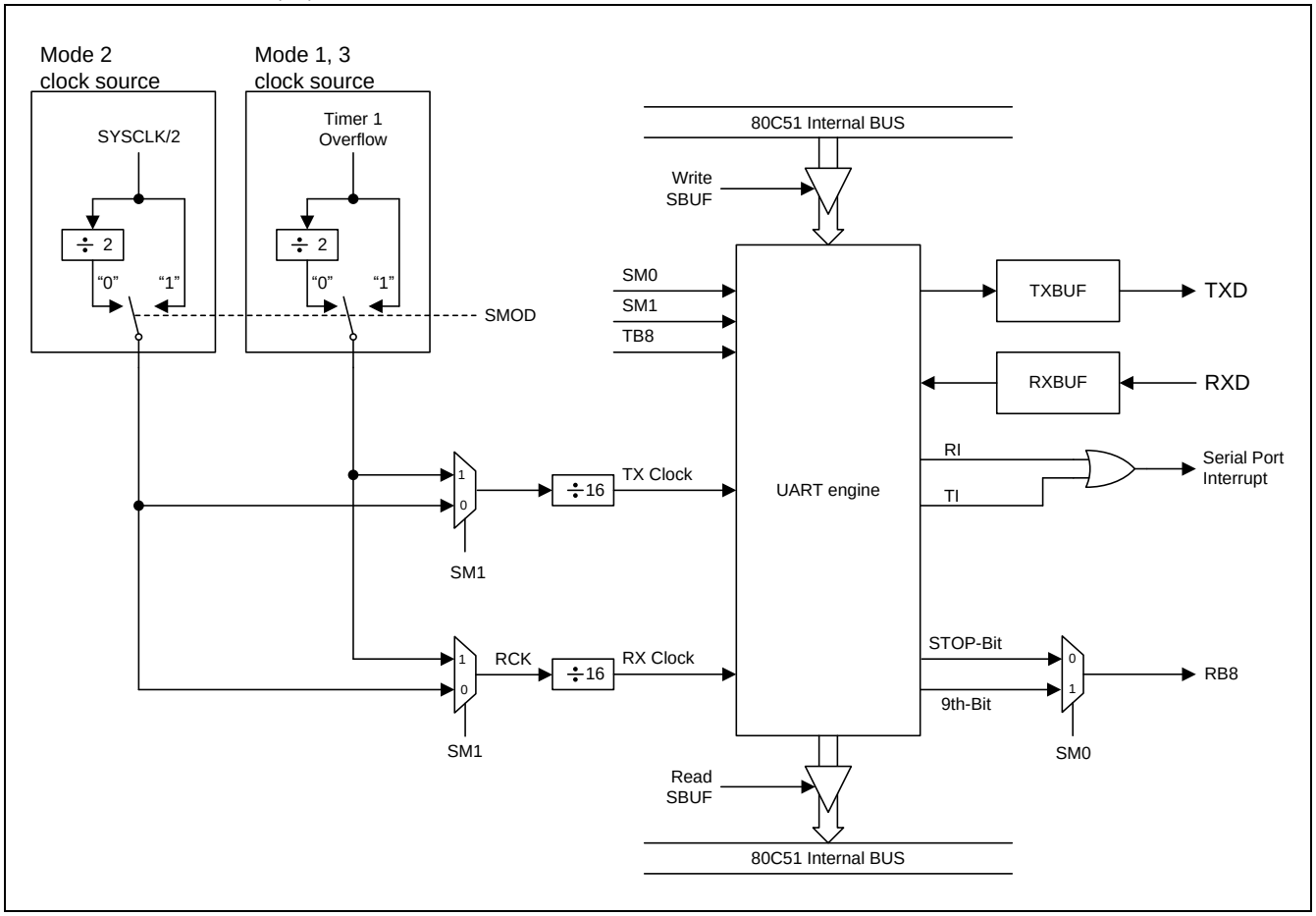
## 15.2. 串行口模式 1

通过 TXD 发送 10 位数据或通过 RXD 接收 10 位数据：一个起始位(0)，8 个数据位(低位先出)，和一个停止位(1)。在接收时，停止位进入 SCON 的 RB8，波特率由定时器 1 的溢出速率来决定。图 15–1 显示了模式 1 的数据帧和图 15–6 显示了模式 1 的串行口简化功能图。

使用 SBUF 作为目的寄存器的任何指令来启动传输。写到 SBUF 的信号请求 UART 引擎开始发送，当收到一个发送请求后，UART 将在 TX 时钟的上升沿开始发送。SBUF 中的数据从 TXD 引脚串行输出，数据帧如图 15–1 所示及数据宽度根据 TX 时钟不同而不同。当 8 位数据发送完后，硬件将置位 TI 表示发送结束。

当串行口控制器在 RCK 采样时钟下检测到在 RXD 有 1 到 0 跳变的起始位时接收开始。在 RXD 引脚上的数据将被串行口的位侦查器采样。当收到停止位后，硬件置位 RI 表示接收结束并把停止位加载到 SCON 寄存器的 RB8。

图 15–6. 串行口模式 1, 2, 3



## 15.3. 串行口模式 2 和模式 3

通过 TXD 传送 11 位或通过 RXD 接收 11 位：一个起始位(0)，8 个数据位(低位在先)，一个可编程的第 9 个数据位和一个停止位(1)。在传送时，数据的第 9 位(TB8)可分配为 0 或 1。在接收时，数据的第 9 位将进入到 SCON 的 RB8。在模式 2 波特率可编程为 1/16，1/32 或 1/64 的系统时钟频率。模式 3 可以产生可以从定时器 1 或定时器 2 产生可变的波特率。

图 15–2 列出了模式 2 和模式 3 的数据帧。图 15–6 列出了模式 2 和模式 3 的串行口功能图。接收部分和模式 1 相同。与模式 1 传送部分不同的仅仅是传送移位寄存器的第 9 位。

写到 SBUF 的信号请求 UART 引擎加载 TB8 到发送移位寄存器的第 9 位并开始发送，当收到一个发送请求后，UART 将在 TX 时钟的上升沿开始发送。SBUF 中的数据从 TXD 引脚串行输出，数据帧如图 15–2 所示及数据宽度根据 TX 时钟不同而不同。当 9 位数据发送完后，硬件将置位 TI 表示发送结束。

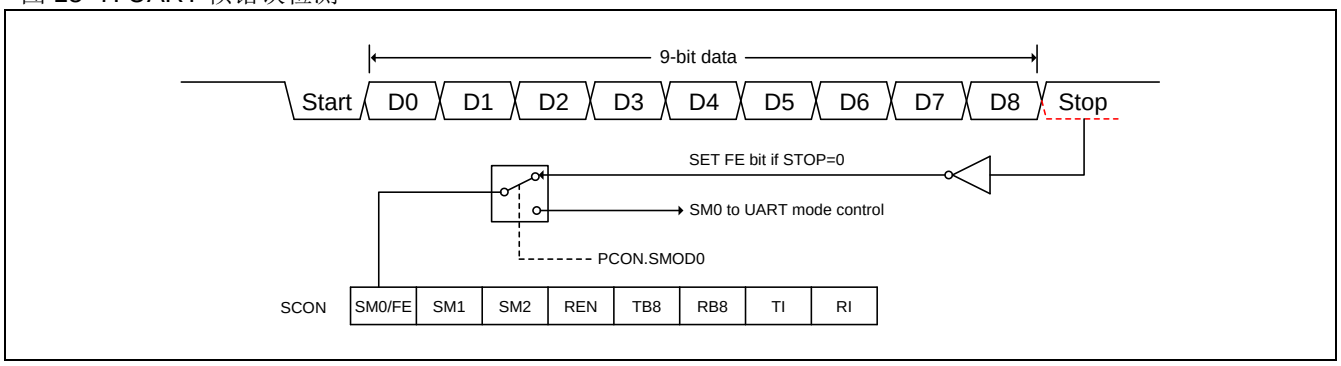
当串行口控制器在 RCK 采样时钟下检测到在 RXD 有 1 到 0 跳变的起始位时接收开始。在 RXD 引脚上的数据将被串行口的位侦查器采样。当收数据接收完后，硬件置位 RI 表示接收结束并把第 9 位加载到 SCON 寄存器的 RB8。

在四种模式中，使用 SBUF 作为一个目的寄存器，可以通过任何指令发起传输。在模式 0，当 RI=0 且 REN=1 时启动接收。在其它模式，在 REN=1 时，收到有 1 到 0 跳变的起始位时启动接收。

## 15.4. 帧错误检测

开启帧错误检测功能后，UART 会在通讯中检测是否丢失停止位，如果丢失一个停止位，就设置 SCON 寄存器的 FE 标志位。FE 标志位和 SM0 标志位共享 SCON.7，SMOD0 标志位(PCON.6)决定 SCON.7 究竟代表哪个标志，如果 SMOD0 位（PCON.6）置位则 SCON.7 就是 FE 标志，SMOD0 位清零则 SCON.7 就是 SM0 标志。当 SCON.7 代表 FE 时，只能软件清零。参考图 15–7.。

图 15–7. UART 帧错误检测



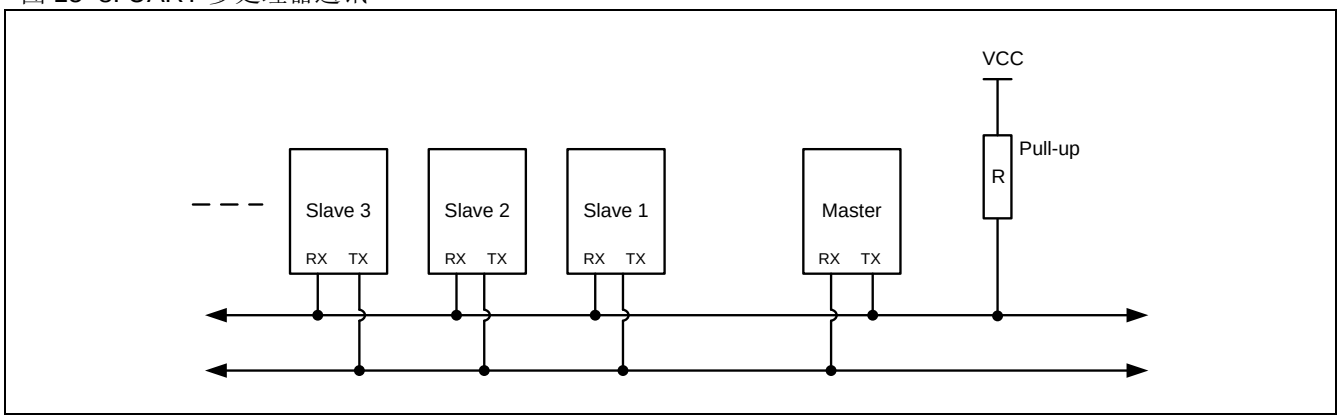
## 15.5. 多处理器通讯

模式 2 和 3 在用作多处理器通讯时有特殊的规定见图 15–8。在这两种模式，接收 9 个数据位。第 9 个数据位存入 RB8，接着进来一个停止位。端口可以编程为：在 RB8=1 时，当收到停止位后，串口中断将激活。这种特征通过设置 SM2 位(在 SCON 寄存器中)来使能。这种方式用于多处理器系统如下：

当主处理器想传送一个数据块到多个从机中的某一个时，首先传送想要传送的目标地址标识符的地址。地址字节与数据字节的区别在于，在地址字节中第 9 位为 1，数据字节中为 0。当 SM2=1 时，收到一个数据字节将不会产生中断。然而一个地址字节将引发所有从机中断。因而所有的从机可以检测收到的字节是否是自己的地址。从机地址将清除 SM2 位并准备好接收即将进来的所有数据。从机地址不匹配的将保持 SM2 置位，并继续他们的工作，忽略进来的数据字节。

SM2 在模式 0 和模式 1 没有影响，但是可以用来检测停止位的有效性。在接收模式 1 中，如果 SM2=1，除非收到一个有效的停止位否则接收中断不会被激活

图 15–8. UART 多处理器通讯




## 15.6. 自动地址识别

自动地址识别通过硬件比较可以让 UART 识别串行码流中的地址部分，该功能免去了使用软件识别时需要大量代码的麻烦。该功能通过置位 SCON 的 SM2 位来开启。

在 9 位数据 UART 模式下，即模式 2 和模式 3，收到特定地址或广播地址时自动置位接收中断(RI)标志，9 位模式的第 9 位信息为 1 表明接收的是一个地址而不是数据。自动地址识别功能请参考图 15–9。在 8 位模式，即模式 1 下，如果 SM2 置位并且在 8 位地址与给定地址或广播地址核对一致后收到有效停止位则 RI 置位。 模式 0 是移位寄存器模式，SM20 被忽略。

使用自动地址识别功能可以让一个主机选择性的同一个或多个从机进行通讯，所有从机可以使用广播地址接收信息。增加了 SADDR 从机地址寄存器和 SADEN 地址掩码寄存器。

SADEN 用来定义 SADDR 中的那些位是“无关紧要”的，SADEN 掩码和 SADDR 寄存器进行逻辑与来定义供主机寻址从机的“给定” 地址，该地址让多个从机进行排他性的识别。

下面的实例帮助理解这个方案的通用性:

| 从机 0 | 从机 1 |
|---|---|
| SADDR = 1100 0000 | SADDR = 1100 0000 |
| SADEN = 1111 1101 | SADEN = 1111 1110 |
| 特定地址= 1100 00X0 | 特定地址= 1100 000X |

上例中 SADDR 相同，而 SADEN 不同以区分两个分机。从机 0 要求位 0 等于 0，并忽略位 1。 从机 1 要求位 1 等于 0，并忽略位 0。从机 0 的唯一地址是 1100 0010，因为从机 1 要求位 1 等于 0。从机 1 的唯一地址是 1100 0001，因为位 0 等于 1 可以屏蔽从机 0。一个地址的位 0=0 并且位 1=0，可以同时选择两个从机。因此，地址 1100 0000 可以同时寻址从机 0 和从机 1

下面是一个更为复杂的系统，可选择从机 0 和从机 1，而屏蔽从机 2：

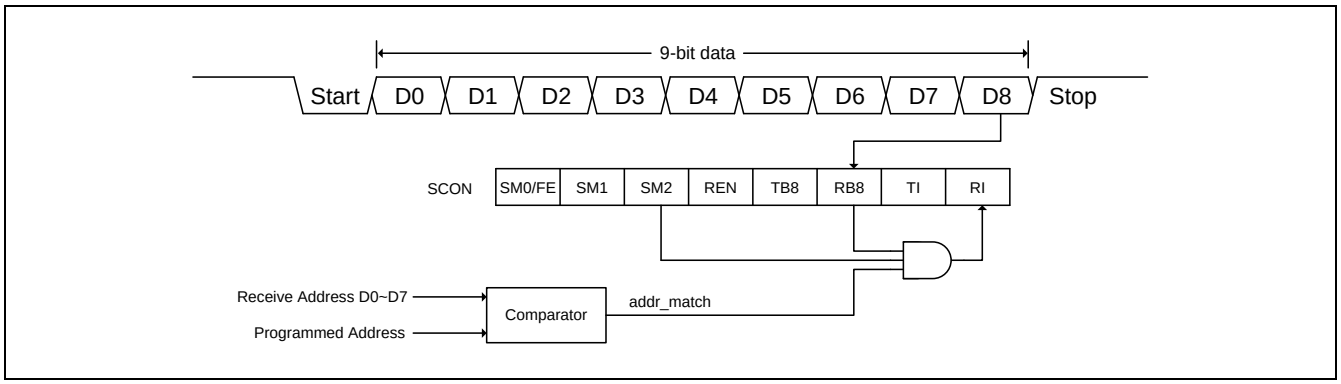
| 从机 0 | 从机 1 | 从机 2 |
|---|---|---|
| SADDR = 1100 0000 | SADDR = 1110 0000 | SADDR = 1110 0000 |
| SADEN = 1111 1001 | SADEN = 1111 1010 | SADEN = 1111 1100 |
| 特定地址= 1100 0XX0 | 特定地址= 1110 0X0X | 特定地址= 1110 00XX |

上述例子中，三个从机地址只有低 3 位不同。从机 0 要求位 0 等于 0，可通过 1110 0110 单独寻址。从机 1 要求位 1 等于 0，可通过 1110 0101 单独寻址。从机 2 要求位 2 等于 0，可通过 1110 0011 单独寻址。为了选择从机 0 和从机 1，同时屏蔽从机 2，可以使用地址 1110 0100，因为要屏蔽从机 2，这个地址的位 2 必须为 1。

每个从机的广播地址是由 SADDR 和 SADEN 逻辑或所得，结果为 0 的位视为无关位。大多数情况下，无关位被认为是 1，这样广播地址就是 0xFF。.

复位后，SADDR(SFR 地址 0xA9)和 SADEN(SFR 地址 0xB9)的值均为 0，这样就产生了一个所有位都是无关位的特定地址，和所有位都是无关位的广播地址一样。这实际上禁止了自动地址识别模式，从而可让微处理器使用没有这个功能的标准 80C51 的 UART。

图 15–9. 自动地址识别




注意:
(1) 地址匹配后(addr_match=1), 清除 SM2 以接收数据字节
(2) 收完全部数据字节后,置 SM2 为 1 以等待下一个地址

## 15.7. 波特率设置

AUXR 寄存器的位 T1X12 和 URM0X6 为波特率设置提供了一个新的选项。如下所示：

### 15.7.1. 模式 0 的波特率

图 15–10. 模式 0 波特率计算公式

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

注意:
如果 URM0X6=0,波特率公式和标准 8051 一样。

表 15–1. 串行口模式 0 波特率范例

| SYSCLK | URM0X6 | 模式 0 波特率 |
|---|---|---|
| 12MHz | 0 | 1M bps |
| 12MHz | 1 | 6M bps |
| 24MHz | 0 | 2M bps |
| 24MHz | 1 | 12M bps |

### 15.7.2. 模式 2 的波特率

图 15–11. 模式 2 波特率计算公式

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

表 15–2. 串行口模式 2 波特率范例

| SYSCLK | SMOD | 模式 2 波特率 |
|---|---|---|
| 22.1184MHz | 0 | 345.6K bps |
| 22.1184MHz | 1 | 172.8K bps |
| 24MHz | 0 | 750K bps |
| 24MHz | 1 | 375K bps |

### 15.7.3. 模式 1 和 3 的波特率

**使用定时器 1 作为波特率发生器**

图 15–12. 模式 1/3 波特率计算公式

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

“表 15–3 ~ 表 15–6“例举了各种常用的波特率及它们如何从定时器 1 的 8 位自动装载模式获取得到.

表 15–3. 定时器 1 在 $F_{SYSCLK}$=11.0592MHz 时产生常用波特率

| 波特率 | **TH1**, 自动装载值 | | | | | |
|---|---|---|---|---|---|---|
| | **T1X12=0** | | | **T1X12=1** | | |
| | SMOD=0 | SMOD=1 | 误差 | SMOD=0 | SMOD=1 | 误差 |
| 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% |

表 15–4. 定时器 1 在 $F_{SYSCLK}$=22.1184MHz 时产生常用波特率

| 波特率 | **TH1**, 自动装载值 | | | | | |
|---|---|---|---|---|---|---|
| | **T1X12=0** | | | **T1X12=1** | | |
| | SMOD=0 | SMOD=1 | 误差 | SMOD=0 | SMOD=1 | 误差 |
| 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% |

表 15–5. 定时器 1 在 $F_{SYSCLK}$=12.0MHz 时产生常用波特率

| 波特率 | **TH1**, 自动装载值 | | | | | |
|---|---|---|---|---|---|---|
| | **T1X12=0** | | | **T1X12=1** | | |
| | SMOD=0 | SMOD=1 | 误差 | SMOD=0 | SMOD=1 | 误差 |
| 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 | -- | -- | -- | -- | -- | -- |

表 15–6. 定时器 1 在 $F_{SYSCLK}$=24.0MHz 时产生常用波特率

| 波特率 | **TH1**, 自动装载值 | | | | | |
|---|---|---|---|---|---|---|
| | **T1X12=0** | | | **T1X12=1** | | |
| | SMOD=0 | SMOD=1 | 误差 | SMOD=0 | SMOD=1 | 误差 |
| 1200 | 204 | 152 | 0.16% | -- | -- | -- |
| 2400 | 230 | 204 | 0.16% | -- | -- | -- |
| 4800 | 243 | 230 | 0.16% | 100 | -- | 0.16% |
| 9600 | -- | 243 | 0.16% | 178 | 100 | 0.16% |
| 14400 | -- | -- | -- | 204 | 152 | 0.16% |
| 19200 | -- | -- | -- | 217 | 178 | 0.16% |
| 28800 | -- | -- | -- | 230 | 204 | 0.16% |
| 38400 | -- | -- | -- | -- | 217 | 0.16% |
| 57600 | -- | -- | -- | 243 | 230 | 0.16% |
| 115200 | -- | -- | -- | -- | 243 | 0.16% |

## 15.8. 串行口寄存器

串行口的四种操作模式除波特率的设定之外都与标准的 8051 相同。此两个寄存器 PCON, AUXR 是与波特率的设定有关

SCON: 串行口控制寄存器
SFR 地址 = 0x98 复位值 = 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, 帧错误位。SMOD0 必须置位才能访问 FE 位。
0: FE 位不会被有效的帧清零，它应当被软件清零。
1: 当检测到一个无效的停止位时，该位被接收器置位。

Bit 7: SM0, 串行口模式位 0， (SMOD0 必须 = 0 才能访问位 SM0)
Bit 6: SM1, 串行口模式位 1

| SM0 | SM1 | 模式 | 描述 | 波特率 |
|---|---|---|---|---|
| 0 | 0 | 0 | 移位寄存器 | SYSCLK/12 or SYSCLK/2 |
| 0 | 1 | 1 | 8-bit UART | 可变 |
| 1 | 0 | 2 | 9-bit UART | SYSCLK/64, /32 |
| 1 | 1 | 3 | 9-bit UART | 可变 |

Bit 5: SM2, 串行口模式位 2
0: 禁止 SM2 功能
1: 在模式 2 和 3 时使能地址自动识别，如果 SM2=1 那么 RI 将不被置位，除非接收到的第 9 位数据(RB8)为 1，表示是一个地址，并且接收到的字节是特定地址或者是一个广播地址；在模式 1，如果 SM2=1 那么 RI0 将不能被激活除非收到一个有效的停止位，并且接收到的字节是特定地址或者是一个广播地址；在模式 0，SM2 应当为 0。

Bit 4: REN, 使能串行接收
0: 软件清零是禁止串行接收.
1: 软件置位是使能串行接收

Bit 3: TB8, 在模式 2 和 3 下此位是被传送数据的第 9 位。由软件清零或置位。
Bit 2: RB8, 在模式 2 和 3 下此位是接收到数据的第 9 位。在模式 1 下，如果 SM2 = 0, RB8 则是接收到的停止位。在模式 0，RB8 没有用到。

Bit 1: TI. 传送中断标志位
0: 必须由软件清零。
1: 在模式 0 下第 8 位传送时间结束硬件设置，在其它模式下传送停止位开始硬件设置。

Bit 0: RI. 接收中断标志位
0: 必须由软件清零。
1: 在模式 0 下第 8 位接收时间结束硬件置位，在其它模式下接收停止位一半时开始硬件置位（除了关注 SM2 之外）。

SBUF: 串口缓冲寄存器
SFR 地址 = 0x99 复位值 = 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: 作为传送和接收的缓冲寄存器。

SADDR: 从地址寄存器

SFR 地址 = 0xA9 复位值 = 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: 从地址屏蔽寄存器

SFR 地址 = 0xB9 复位值 = 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 寄存器 和 SADEN 寄存器结合形成用于自动地址识别的特定/广播地址。实际上，SADEN 作为用于 SADDR 寄存器的“屏蔽”寄存器功能。如下所示：

SADDR = 1100 0000
SADEN = 1111 1101
特定地址 = 1100 00x0 ⟶ 这特定的从机地址将被选中，
除了位 1 作不关心处理外

每个从设备的广播地址是 SADDR 和 SADEN 进行逻辑“或”的结果，结果中为“0”的位将被忽略。在系统复位后，SADDR 和 SADEN 都被初始化为 0，从而“不在乎”所有的特定地址和“不在乎”所有的广播地址，而导致自动地址识别功能无效。

PCON: 电源控制寄存器

SFR 地址 = 0x87 POR = 0011-0000, 复位值 = 0000-0000

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

Bit 7: SMOD, 双倍波特率控制位
0: 禁止 UART 的双倍波特率
1: 使能 UART 在模式 1，2，3 的双倍波特率.

Bit 6: SMOD0, 帧错误选择.
0: SCON.7 是 SM0 功能.
1: SCON.7 是 FE 功能。注意，在一个帧错误后，FE 将被置位，不管 SMOD0 的状态。

AUXR: 辅助寄存器

SFR 地址 = 0x8E 复位值 = 0000-00xx

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

Bit 6: T1X12, 当 C/T=0 时定时器 1 时钟源选择
0: 清零选择 SYSCLK/12.
1: 置位选择 SYSCLK 作为时钟源

Bit 5: URM0X6, UART 模式 0 的波特率时钟源选择.
0: 清零选择 SYSCLK/12 作为 UART 模式 0 的波特率。
1: 置位选择 SYSCLK/2 作为 UART 模式 0 的波特率。

## 15.9. 串行口示例代码

(1). 功能需求: 串行口输入 RI 唤醒空闲模式

汇编语言代码范例:

```
PS                  EQU             10h
PSH                 EQU             10h


   ORG   00023h
uart_ri_idle_isr:
   JB    RI,RI_ISR                 ; 判断是否串行输入中断
   JB    TI,TI_ISR                 ; 判断是否串行发送中断   RETI                    ; 中断返回

RI_ISR:
;  Process
   CLR   RI                        ; 清除 RI 标志
   RETI                            ; 中断返回

TI_ISR:
;  Process
   CLR   TI                        ; 清除 TI 标志
   RETI                            ; 中断返回

main:
   CLR   TI                        ; 清除 TI 标志
   CLR   RI                        ; 清除 RI 标志
   SETB  SM1                       ;
   SETB  REN                       ; 8 位的模式 2，接收使能

   CALL  UART_Baud_Rate_Setting    ;参考 "表 15–3 ~ 表 15–6" 获得更多信息

   MOV   IP,#PSL                   ; 选择串行口中断优先级
   MOV   IPH,#PSH                  ;

   SETB  ES                        ; 使能串行口中断
   SETB  EA                        ; 使能全局中断

   ORL   PCON,#IDL;                ; 设置 MCU 进入空闲模式
```

C 语言代码范例:

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

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

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

```
void main(void)
{
   TI = RI = 0;
   SM1 = REN = 1;                    // 8 位的模式 2，接收使能

   UART_Baud_Rate_Setting()          //参考 “表 15–3 ~ 表 15–6“ 获得更多信息

   IP = PSL;                        //选择串行口中断优先级
   IPH = PSH;                        //

   ES = 1;                          // 使能串行口中断
   EA = 1;                          //使能全局中断

   PCON |= IDL;                     //设置 MCU 进入空闲模式
}
```

# 16. 可编程计数器阵列(PCA)

**MA801_MA802** 预备了一个可编程计数器阵列 (PCA), 它与标准的定时器/计数器相比用较少的 CPU 干预提供了更多的时间功能。它的优点包括降低软件开销和提供精确度。

## 16.1. PCA 概述

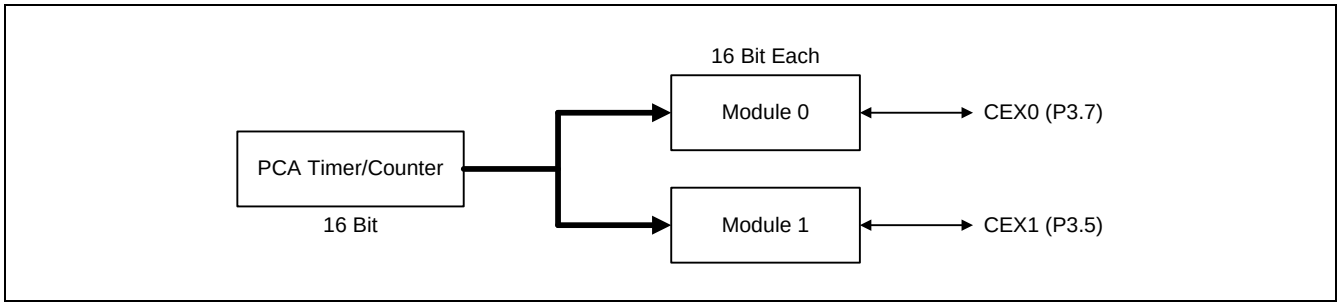
PCA 包含一个专用的定时器/计数器，它给一个系列的 2 个比较/捕获模块提供时基服务。图 16–1 列出了 PCA 的方框图。注意 PCA 定时器和模块都是 16 位的。如果一个外部事件被关联到一个模块，那它功能与相对应的引脚共享。如果模块没有用到引脚，则这引脚仍用于标准 I/O 口。

模块 0 和模块 1 可以被编程为下列模式的任一种：

- -上升和/或下将沿捕获
- -软件定时器
- -高速输出
- -脉宽调制 (PWM) 输出

所有模式将在之后详细说明。然而，让我们首先看看怎样设置 PCA 定时器和模块。

图 16–1. PCA 方框图



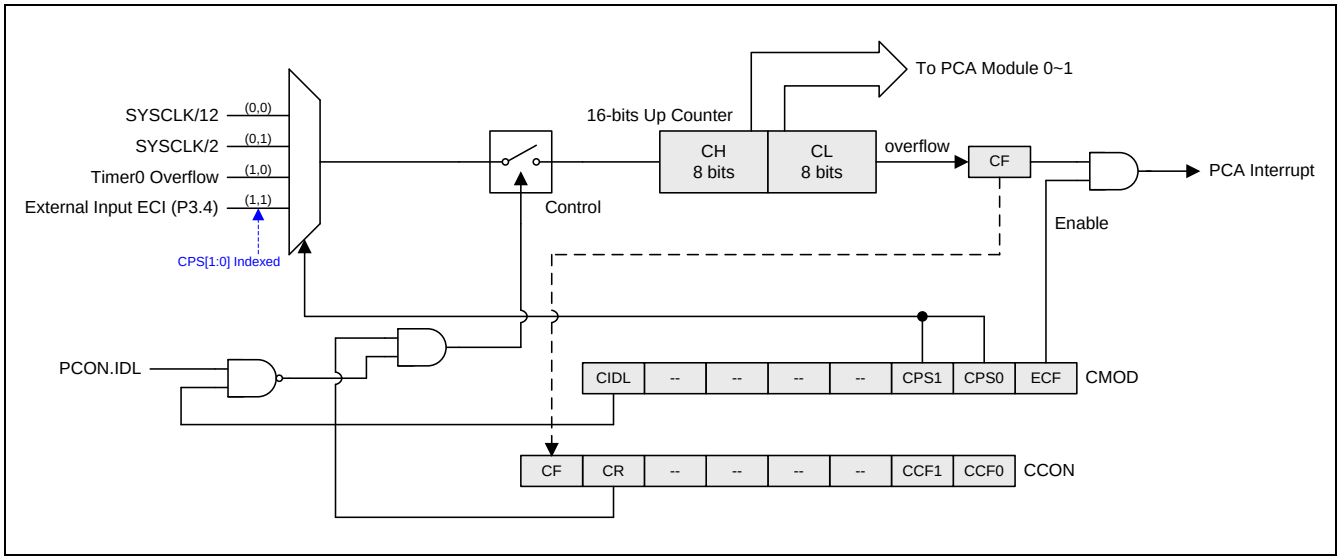
## 16.2. PCA 定时器/计数器

PCA 定时器/计数器是一个自动装载的 16 位定时器包含 CH 和 CL 寄存器(计数值的高和低字节)，如图 16–2 所示。它是所有模块的通用时基，它的时钟输入可以选择下列源：

- 1/12 系统时钟频率
- 1/2 系统时钟频率
- -定时器 0 溢出,它允许一个慢些时钟输入范围给定时器.
- -外部时钟输入, 1 到 0 变化, 在 ECI 引脚 (P3.4).

特殊功能寄存器 CMOD 包含用于 PCA 定时器输入的计数脉冲选择位(CPS1 和 CPS0) 。这个寄存器也包括 ECF 位，当计数器溢出，它就使能一个中断。另外，用户有在 Idle 模式是否运行 PCA 计数器的选项，通过设置计数器 Idle 位(CIDL)来选择。这能进一步的在 Idle 模式下降低功耗。

图 16–2. PCA 定时器/计数器




CMOD: PCA 计数器模式寄存器

SFR 地址 = 0xD9 复位值 = 0xxx-x000

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

Bit 7: CIDL, PCA 计数器空闲模式控制
0: 在空闲模式下 PCA 计数器继续运行
1: 在空闲模式下关闭 PCA 计数器

Bit 6~3: 保留。当写 CMOD 时，软件必须在这些位上写”0”。

Bit 3~1: CPS2-CPS0, PCA 计数器时钟源选择位

| CPS1 | CPS0 | PCA 时钟源 |
|---|---|---|
| 0 | 0 | 内部时钟, SYSCLK/12 |
| 0 | 1 | 内部时钟, SYSCLK/2 |
| 1 | 0 | 定时器 0 溢出 |
| 1 | 1 | ECI 引脚外部时钟 |

Bit 0: ECF, 使能 PCA 计数器溢出中断
0: 当 CF 位（在 CCON 寄存器里）置 1 时不触发中断

1: 当 CF 位（在 CCON 寄存器里）置 1 时触发中断.

下面列出的 CCON 寄存器包含了 PCA 的控制位和 PCA 定时器及每个模块的标志位。要运行 PCA，必须软件置位 CR(CCON.6)，清零关闭 PCA。 PCA 计数器溢出，置位 CF(CCON.7)，如果 CMOD 寄存器的 ECF 位为 1，则会产生一个中断。CF 位仅由软件清零。 CCF0 和 CCF1 是模块 0 和模块 1 的中断标志位，各自的，当一个匹配或者捕获发生时，它们由硬件置位。这些标志位也仅能由软件清零。 PCA 中断系统如图 16–3. 所示。

CCON: PCA 计数器控制寄存器

SFR 地址 = 0xD8 复位值 = 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 计数器溢出标志
0: 仅由软件清零.
1: 计数器溢出时由硬件置位。如果 CMOD 里的 ECF 为 1，则 CF 标志位能产生一个中断。CF 可以由硬件或软件置位。

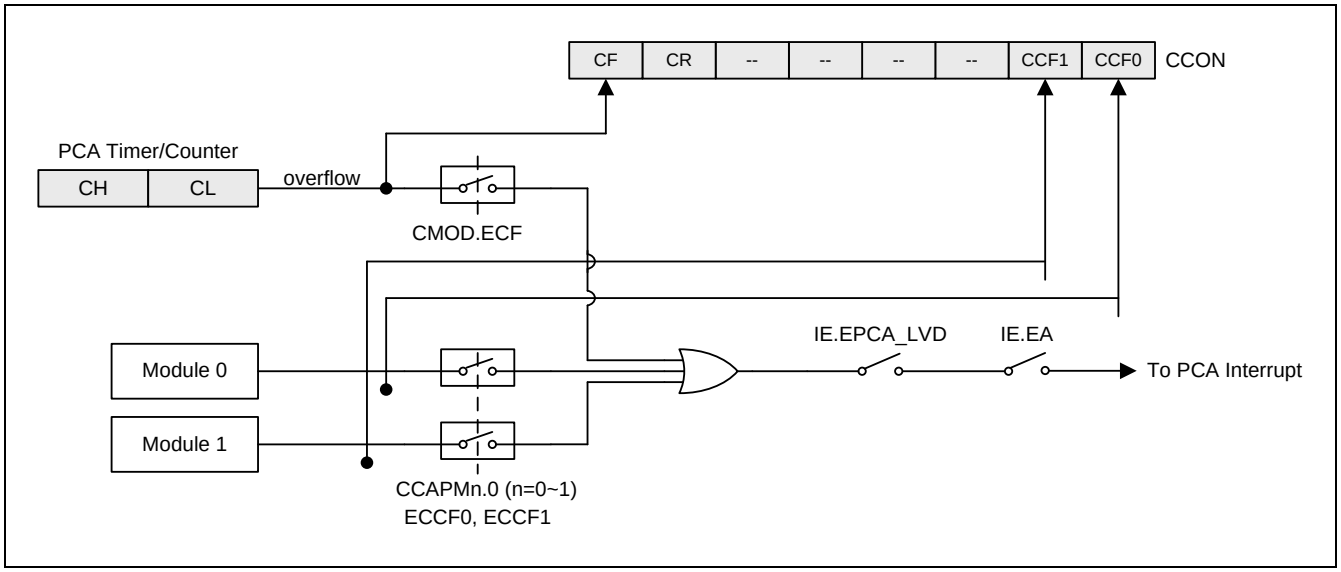
Bit 6: CR, PCA 计数器运行控制位.
0: 必须软件清零关闭 PCA 计数器。
1: 软件置位打开 PCA 计数器。

Bit 5~2: 保留。当写 CCON 时，软件必须在这些位上写”0”。

Bit 1: CCF1, PCA 模块 1 中断标志
0: 必须由软件清零
1: 当一个匹配或捕获发生时由硬件置位。

Bit 0: CCF0, PCA 模块 0 中断标志
0: 必须由软件清零
1: 当一个匹配或捕获发生时由硬件置位。

图 16–3. PCA 中断系统




## 16.3. 比较/捕获模块

每个比较/捕获模块都有一个叫做 CCAPMn(n=0 或 1)的模式寄存器，用来选择要执行的哪个功能。注意位 ECCFn 用来在模块的中断标志位置 1 时使能产生一个中断。

CCAPMn: PCA 模块比较/铺货寄存器, n=0 ~ 1
SFR 地址 = 0xDA~0xDB 复位值 = 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: 保留。当写 CCAPMn 时，软件必须在这个位上写”0”。

Bit 6: ECOMn, 使能比较器
0: 禁止数字比较器功能。
1: 使能数字比较器功能。

Bit 5: CAPPn, 捕获上升沿使能
0: 禁止 PCA 捕获功能在 CEXn 上升沿上检测。
1: 使能 PCA 捕获功能在 CEXn 上升沿上检测。

Bit 4: CAPNn, 捕获下降沿使能
0: 禁止 PCA 捕获功能在 CEXn 下降沿上检测。
1: 使能 PCA 捕获功能在 CEXn 下降沿上检测。

Bit 3: MATn, 匹配控制
0: 禁止数字比较器匹配事件置位 CCFn。
1: PCA 计数器和模块的比较/捕获寄存器匹配导致 CCON 上的 CCFn 位置 1。

Bit 2: TOGn, 反转控制
0: 禁止数字比较器匹配事件反转 CEXn。
1: PCA 计数器和模块的比较/捕获寄存器匹配导致 CEXn 引脚的反转。

Bit 1: PWMn, PWM 控制.
0: 禁止在 PCA 模块的 PWM 模式。
1: 使能 PWM 功能，CEXn 引脚被用来作为脉宽调制输出。

Bit 0: ECCFn, 使能 CCFn 中断。
0: 禁止 CCON 寄存器里的比较/捕获标志 CCFn 产生中断。
1: 使能 CCON 寄存器里的比较/捕获标志 CCFn 产生中断。

注意：位 CAPNn(CCAPMn.4)和 CAPPn(CCAPMn.5)决定捕获输入脚触发的边缘。如果两个位都置 1，则两个边沿都使能，不管什么变化都会捕获。

每个模块都有一对 8 位比较/捕获寄存器（CCAPnH,CCAPnL）与之对应。这些寄存器用来存储捕获事件或比较事件发生时的时间。当一个模块用于 PWM 模式时，在上述两个寄存器之外，一个额外的寄存器 PCAPWMn 用来提升输出占空比的范围。提升后的占空比的范围是从 0%到 100%，步长是 1/256。

CCAPnH: PCA 模块 n 捕获高寄存器, n=0~1
SFR 地址 = 0xFA~0xFB 复位值 = 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 模块 n 捕获低寄存器, n=0~1
SFR 地址 = 0xEA~0xEB 复位值 = 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 模式辅助寄存器，n=0, 1
SFR 地址 = 0xF2~0xF3 复位值 = xxxx-xx00

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

Bit 7~2: 保留。当写 PCAPWMn 时，软件必须在这些位上写”0”。

Bit 1: ECAPnH, 扩展第 9 位（最高位），与 CCAPnH 关联组成一个用于 PWM 模式的 9 位寄存器。

Bit 0: ECAPnL, 扩展第 9 位 (最高位)，与 CCAPnL 关联组成一个用于 PWM 模式的 9 位寄存器。

## 16.4. PCA 工作模式

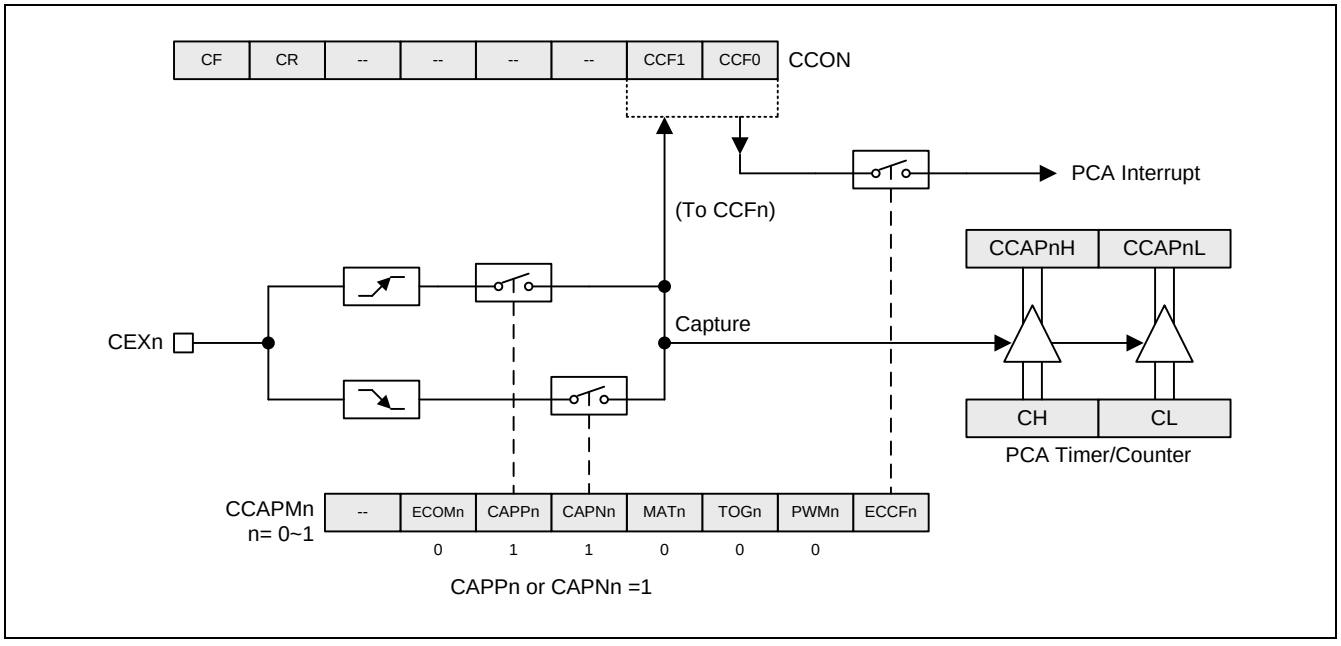
表 16–1 列出了各种 PCA 功能的 CCAPMn 寄存器的设置。
表 16–1. PCA 模块模式

| ECOMn | CAPPn | CAPNn | MATn | TOGn | PWMn | ECCFn | 模块功能 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 不工作 |
| X | 1 | 0 | 0 | 0 | 0 | X | 16 位捕获 CEXn 上的上升沿 |
| X | 0 | 1 | 0 | 0 | 0 | X | 16 位捕获 CEXn 上的下降沿 |
| X | 1 | 1 | 0 | 0 | 0 | X | 16 位捕获 CEXn 上的变化 |
| 1 | 0 | 0 | 1 | 0 | 0 | X | 16 位软件定时器 |
| 1 | 0 | 0 | 1 | 1 | 0 | X | 16 位高速输出 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 8 位脉宽调制器 (PWM) |

### 16.4.1. 捕获模式

为了将某一 PCA 模块用作捕获模式，那个模块的位 CAPN 和 CAPP，任何一个或两个必须置 1。模块的外部 CEX 输入脚变化时采样。当一个有效的变化发生时，PCA 硬件将把 PCA 计数器寄存器（CH 和 CL）的值载入到模块的捕获寄存器（CCAPnL 和 CCAPnH）。如果模块对应的 CCFn 和 ECCFn 位都置 1，将会产生一个中断。

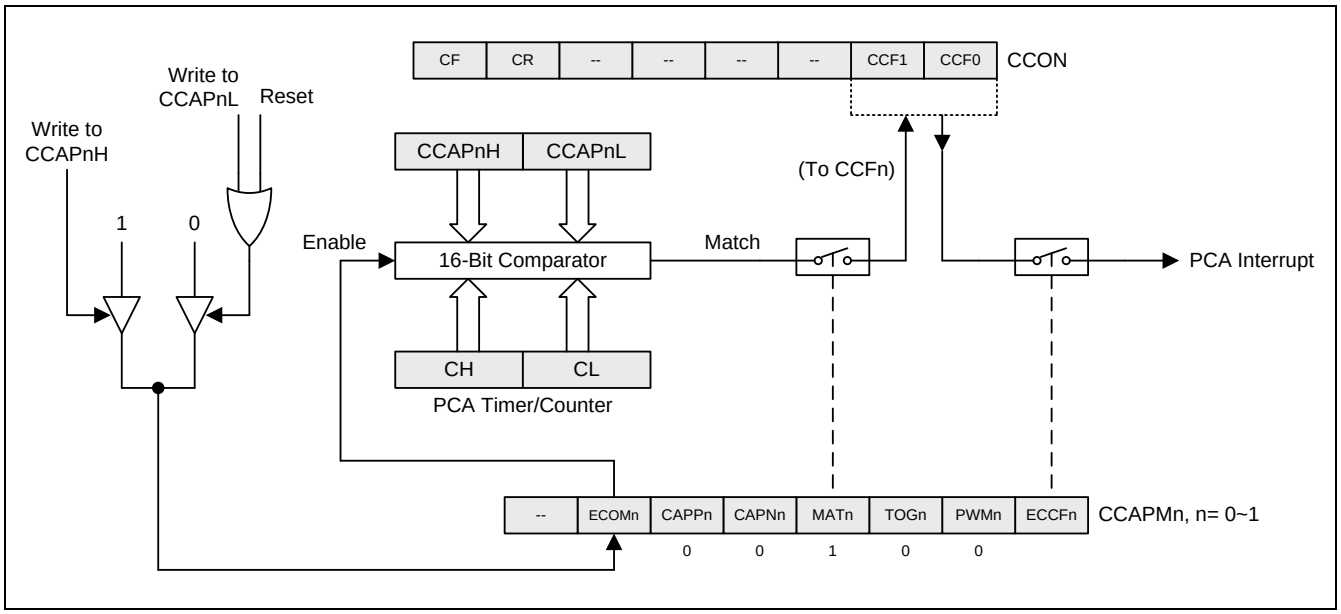
图 16–4. PCA 捕获模式



### 16.4.2. 16 位软件定时器模式

模块的 CCAPMn 寄存器的 ECOM 和 MAT 位同时置 1，可以将 PCA 模块用作软件定时器。 PCA 定时器将与模块捕获寄存器作比较，当相匹配时，如果模块的 CCFn 和 ECCFn 位都置 1，则会产生一个中断。

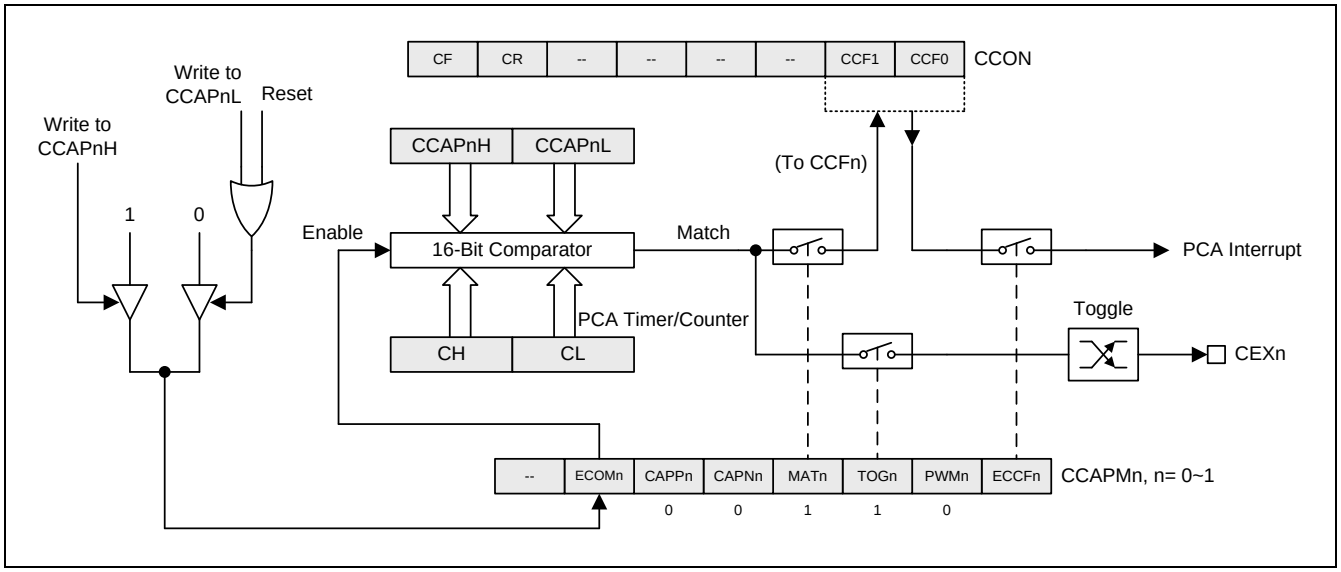
图 16–5. PCA 软件定时器模式



### 16.4.3. 高速输出模式

在这个模式，每次 PCA 计数器和模块捕获寄存器相匹配时都会触发 PCA 模块相对应的 CEX 输出。为了激活这个模块，模块的 CCAPMn 寄存器的 TOG, MAT 和 ECOM 位必须置 1.

图 16–6. PCA 高速输出模式



### 16.4.4. PWM 模式

所有的 PCA 模块都能用于 PWM 输出。输出频率取决于 PCA 时钟的时钟源。所有的模块有相同的输出频率，因为它们共用 PCA 时钟。

每个模块的占空比由模块的捕获寄存器 CCAPnL 和扩展的第 9 位 ECAPnL 所决定。当这 9 位｛0，[CL]｝的值小于{ECAPnl,[CCAPnL]}的 9 位的值时输出低电平，否则大于或等于则输出高电平。

当 CL 由 0xFF 溢出到 0x00， { ECAPnL, [CCAPnL] } 将从{ ECAPnH, [CCAPnH] }的值重载。这允许无毛刺的更新 PWM。模块的 CCAPMn 寄存器里的 PWMn 和 ECOMn 位必须置 1 来使能 PWM 模块。

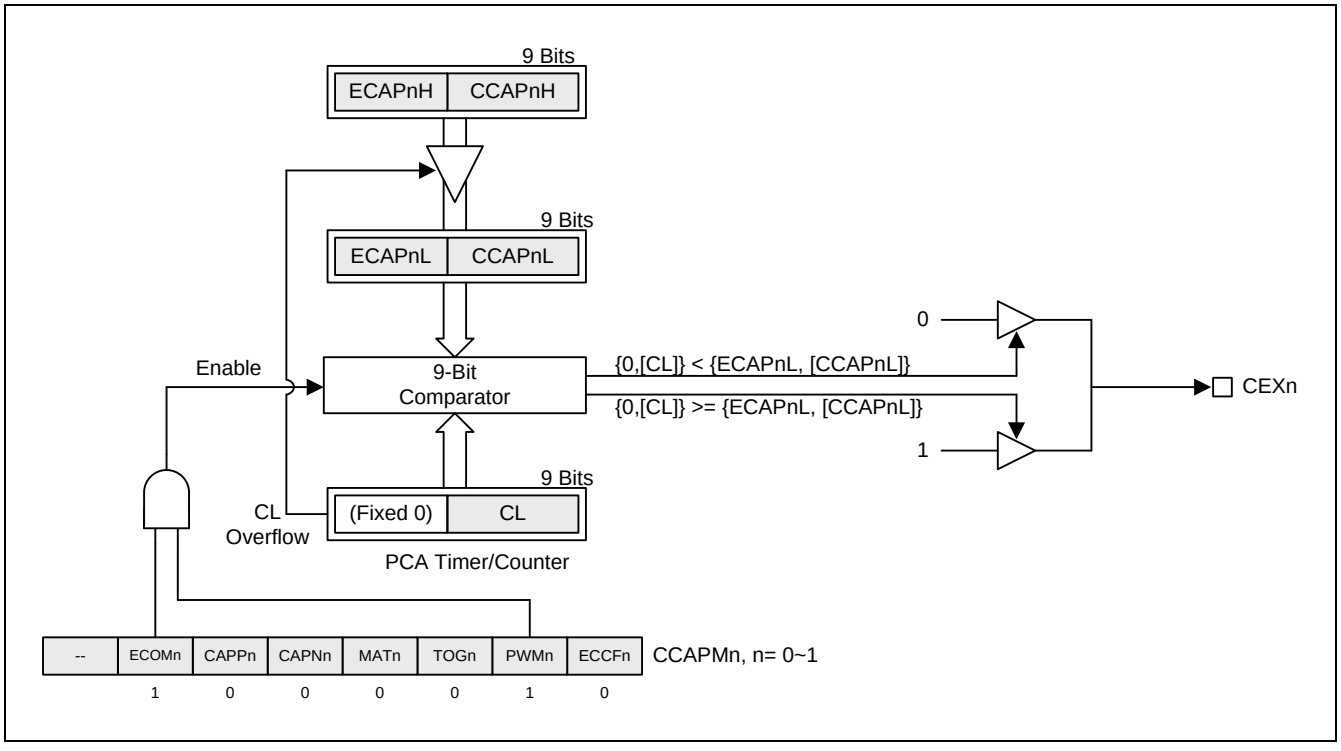
使用 9 位比较，输出的占空比可以提升到实际上的 0%到 100%。占空比的计算公式是：

占空比= 1 – { ECAPnH, [CCAPnH] } / 256.

这里， *[CCAPnH]* 是 CCAPnH 寄存器的 8 位值， *ECAPnH* (PCAPWMn 寄存器的位 1)是一个位的值。因此 *{ ECAPnH, [CCAPnH] }* 组成一个用于 9 位比较器的 9 位数值。

a. 如果 ECAPnH=0 & CCAPnH=0x00 (也就是 0x000), 占空比是 100%.
b. 如果 ECAPnH=0 & CCAPnH=0x40 (也就是 0x040) 占空比是 75%.
c. 如果 ECAPnH=0 & CCAPnH=0xC0 (也就是 0x0C0), 占空比是 25%.
d. 如果 ECAPnH=1 & CCAPnH=0x00 (也就是 0x100), 占空比是 0%.

图 16–7. PCA PWM 模式



## 16.5. PCA 示例代码

(1). 功能需求: 设置模块 0 为 CEX0 上升沿捕捉模式，模块 1 为 PWM 输出占空比为 25%的波形

汇编语言代码范例:

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

PWM2_PWM3:
        MOV         CCON,#00H                   ; 停止 CR
        MOV         CMOD,#02H                    ; PCA 时钟源 = 系统时钟 / 2

        ORL         CCAPM0, #CAPP0              ; 模块 0 为捕捉 CEX0 上升沿

        ORL         CCAPM1, #(ECOM1 + PWM1)     ; 模块 1 为 PWM 输出
        MOV         CCAP3H, #0C0h                       ; 占空比 25 %
        SETB        CR                          ; 启动 PCA
```

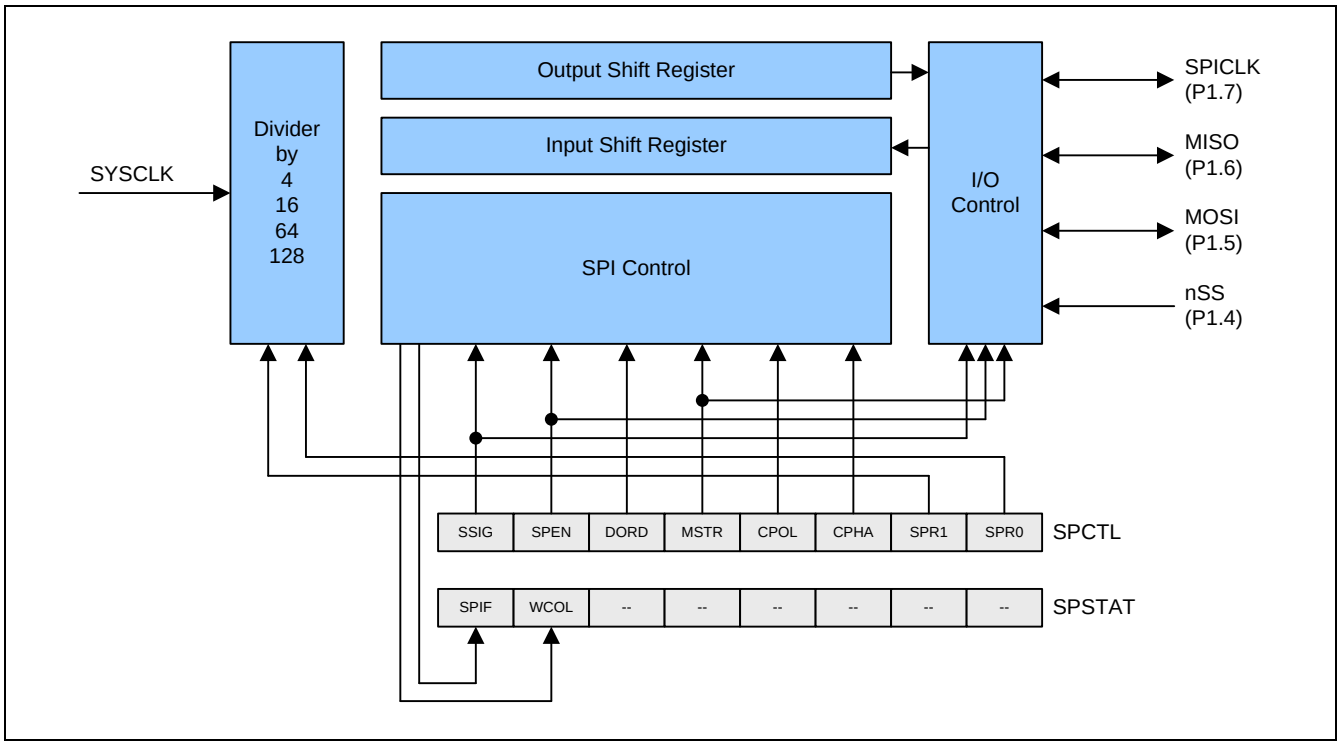
C 语言代码范例:

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


void main(void)
{
        // set PCA
        CCON = 0x00;                                 // 禁止 PCA & 清 CCF0, CCF1, CF 标志
        CMOD = 0x02;                                 // PCA 时钟源 = 系统时钟 / 2

    CCAPM0 |= CAPP0;                                //模块 0 为捕捉 CEX0 上升沿

        CCAPM1 |= (ECOM1 | PWM1);                   //模块 1 为 PWM 输出
        CCAP3H = 0xC0;                              //占空比 25 %

        CR = 1;                                      //启动 PCA

        while (1);
}
```

# 17. 串行外设接口 (SPI)

**MA801_MA802** 提供了一个高速串行通讯接口（SPI）。SPI 接口是一种全双工、高速同步通讯总线，有两种操作模式：主机模式和从机模式。无论哪种模式，12MHz 系统时钟时支持高达 3Mbps 的通讯速度。**MA801_MA802** 的 SPI 状态寄存器（SPSTAT）有一个传送完成标志（SPIF）和写冲突标志（WCOL）。

图 17–1. SPI 框图




SPI 接口有 4 个引脚： MISO (P1.6), MOSI (P1.5), SPICLK (P1.7) 和 nSS (P1.4):

• SPICLK, MOSI 和 MISO 通常将两个或多个 SPI 设备连接在一起。数据从主机到从机使用 MOSI 引脚 (Master Out / Slave In 主出从入) ，从从机到主机使用 MISO 引脚 (Master In / Slave Out 主入从出)。 SPICLK 信号在主机模式时输出，从机模式时输入。若 SPI 接口禁用，即 SPEN (SPCTL.6) = 0，这些引脚可以作为普通 I/O 口使用。

• /SS 是从机选择端。典型配置中，SPI 主机可以使用其某个端口选择某一个 SPI 设备作为当前从机，一个 SPI 从机设备使用它的/SS 引脚确定自己是否被选中。下面条件下/SS 被忽略:

- 若 SPI 系统被禁用，即 SPEN (SPCTL.6) = 0 (复位值).
- 若 SPI 作为主机运行，即 MSTR (SPCTL.4) = 1, 且 P1.4 (/SS) 被配置成输出。
- 若/SS 被设置成忽略，即 SSIG (SPCTL.7) = 1, 这个端口作为普通 I/O 使用。

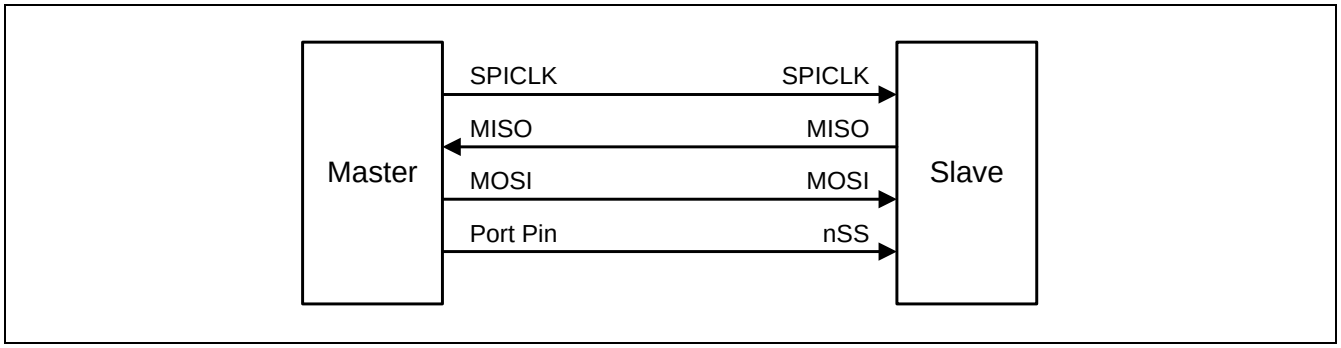
注意，即使 SPI 被配置成主机运行(MSTR=1), 它仍然可以被 nSS 引脚的低电平拉成从机(若 SSIG=0)，一旦发生这种情况，SPIF 位(SPSTAT.7)置位。 (参见 17.2.3 节：nSS 引脚的模式改变)

## 17.1. 典型 SPI 配置

### 17.1.1. 单主机和单从机

对于主机: 任何端口，包括 P1.4 (nSS)，都可以用来控制从机的 nSS 片选引脚。
对于从机: SSIG 为 ‘0’，nSS 引脚决定该设备是否被选中。

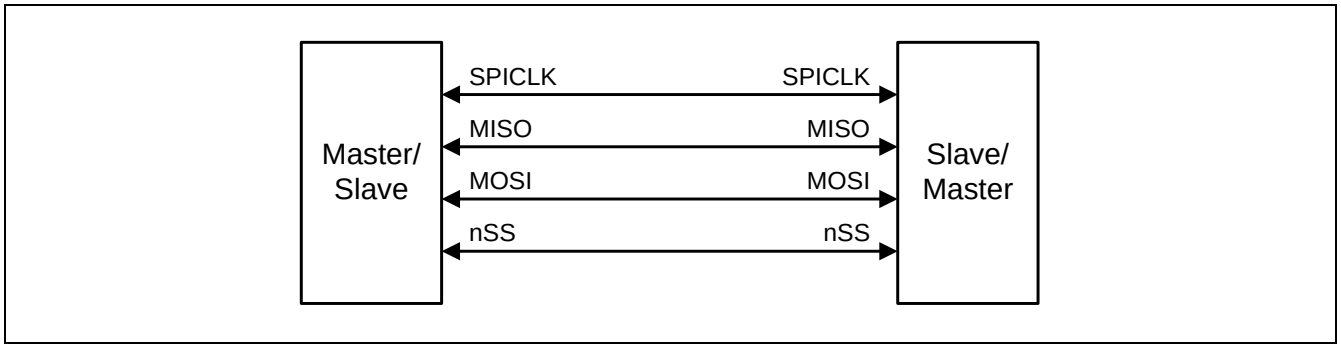
图 17–2. SPI 单主从机配置




### 17.1.2. 双驱动器,可以是主机或从机

两个彼此连接的设备，均可成为主机或从机，没有 SPI 操作时，都可以被通过设置 MSTR=1，SSIG=0, P1.4 (nSS) 双向口配置成主机。任何一方要发起传输，它可以配置 P1.4 位输出并强行拉低，使另一个设备发生“被改成从机模式”事件。 (参见 17.2.3 节：nSS 引脚模式改变)

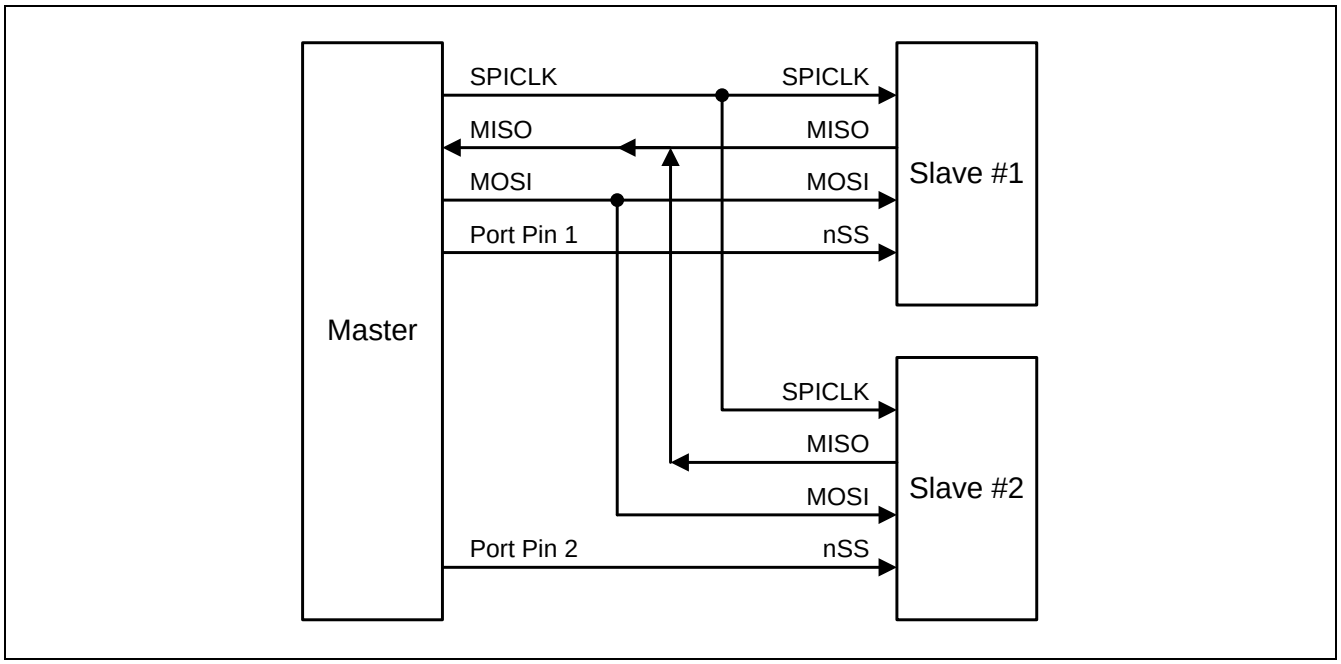
图 17–3. SPI 双驱动器，可以是主机或从机配置



### 17.1.3. 单主机和多从机

对于主机: 任何端口，包括 P1.4 (nSS)，都可以用来控制从机的 nSS 片选引脚。
对于所有从机: SSIG 为 ‘0’，nSS 引脚决定该设备是否被选中。

图 17–4. SPI 单主机和多从机配置



## 17.2. SPI 配置

表 17–1 显示了主/从机模式配置及使用方法和传输方向。.

表 17–1. SPI 主从机选择

| SPEN (SPCTL.6) | SSIG (SPCTL.7) | nSS -引脚 | MSTR (SPCTL.4) | 模式 | MISO -引脚 | MOSI -引脚 | SPICLK -引脚 | 备注 |
|---|---|---|---|---|---|---|---|---|
| 0 | X | X | X | SPI 禁用 | 输入 | 输入 | 输入 | P1.4~P1.7 用作通用 I/O |
| 1 | 0 | 0 | 0 | 从机 (被选中) | 输出 | 输入 | 输入 | 被选择为从机 |
| 1 | 0 | 1 | 0 | 从机 (未被选中) | 高阻 | 输入 | 输入 | 未被选中 |
| 1 | 0 | 0 | 1 ➔ 0 | 从机 (通过模式改变) | 输出 | 输入 | 输入 | 若/SS 被拉低，MSTR 被硬件自动清‘0’，模式被改为从机 |
| 1 | 0 | 1 | 1 | 主机 (待机) | 输入 | 高阻 | 高阻 | MOSI 和 SPICLK 在主机待机时被置为高阻，以防止总线冲突。 |
| | | | | 主机 (活动) | | 输出 | 输出 | MOSI 和 SPICLK 在主机活动时被上拉。 |
| 1 | 1 | X | 0 | 从机 | 输出 | 输入 | 输入 | |
| 1 | 1 | X | 1 | 主机 | 输入 | 输出 | 输出 | |

“X” 表示”无需关心”.

### 17.2.1. 从机注意事项

当 CPHA = 0 时, SSIG 必须为 0 且 nSS 引脚必须在每次串行字节传输前负跳变，传输结束恢复正常高电平。注意 SPDAT 寄存器不能在 nSS 引脚低电平时写入；CPHA = 0, SSIG=1 的操作是未定义的。

当 CPHA =1 时, SSIG 可以为 0 或 1。若 SSIG=0, nSS 引脚可以在每次成功传输之间保持低电平（可以一直拉低）， 这种格式有时非常适合单固定主从机配置应用。

### 17.2.2. 主机注意事项

SPI 通讯中，传输总是有主机发起。若SPI 使能(SPEN=1)并作为主机运行，写入 SPI 数据寄存器(SPDAT) 数据即可开始 SPI 时钟生成器和数据传输器，大约半个到 1 个 SPI 位时间后写入 SPDAT 的数据开始出现在 MOSI 线上。

在开始传输之前，主机通过拉低相应/SS 引脚选择一个从机作为当前从机。写入 SPDAT 寄存器数据从主机 MOSI 引脚移出，同时从从机 MISO 移入主机 MISO 的数据也写入到主机的 SPDAT 寄存器中。

移出 1 字节后，SPI 时钟发生器停止，置传输完成标志 SPIF，若 SPI 中断使能则生成一个中断。主机 CPU 和从机 CPU 中的两个移位寄存器可以看成一个分开的 16 位环形移位寄存器，数据从主机移到从机同时数据也从从机移到主机。这意味着，在一次传输过程中，主从机数据进行了交换。

### 17.2.3. nSS 引脚的模式改变

若 SPEN=1, SSIG=0, MSTR=1 且 /SS pin=1, SPI 使能在主机模式，这种情况下，其他主机可以将/SS 引脚拉低来选择该设备为从机并开始发送数据过来。为避免总线冲突，该 SPI 设备成为一个从机，MOSI 和 SPICLK 引脚被强制为输入端口，MISO 成为输出端口，SPSTAT 中 SPIF 标志置位，若此时 SPI 中断使能，则还会产生一个 SPI 中断。用户软件必须经常去检查 MSTR 位，若该位被从机选择清零而用户又想要继续保持该 SPI 主机模式，用户必须再次设置 MSTR 位，否则，将处于从机模式。

### 17.2.4. 数据冲突

SPI 在发送方向是单缓冲的，而在接收方向是双缓冲的。发送数据直到上一次数据发送完成后才能写入移位寄存器，数据发送过程中写入数据寄存器就会使 WCOL(SPSTAT.6) 置位来表明数据冲突。这种情况下，正在发送的数据继续发送，而刚写入数据寄存器造成冲突的数据就会丢失。

写冲突对于主从机都有可能发生，对于主机，这种现象并不多见，因为主机控制着数据的传送；然而对于从机，由于没有控制权，因此很可能会发生。

对于数据接收，接收的数据被传输到一个并行读数据缓冲器中，以便于移位寄存器再能接收新的字节。然而，接收的数据必须在下个字节完全移入前从数据寄存器 SPDAT 读出，否则前一个数据就会丢失。

WCOL 使用软件向其位写入’1’来清零。

### 17.2.5. SPI 时钟频率选择

SPI 时钟频率选择（主机模式）使用 SPCTL 寄存器的 SPR1 和 SPR0 位来设置，表 17–2.

表 17–2. SPI 串行时钟速率

| SPR1 | SPR0 | SPI 时钟速率 @ SYSCLK=12MHz | SYSCLK 分频 |
|---|---|---|---|
| 0 | 0 | 3 MHz | 4 |
| 0 | 1 | 750 KHz | 16 |
| 1 | 0 | 187.5 KHz | 64 |
| 1 | 1 | 93.75 KHz | 128 |

这里, Fosc 是系统时钟.

## 17.3. 数据模式

时钟相位(CPHA) 位可以让用户设定数据采样和改变时的时钟沿。时钟极性位,CPOL, 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。

图 17–5. SPI 从机传送,CPHA=0

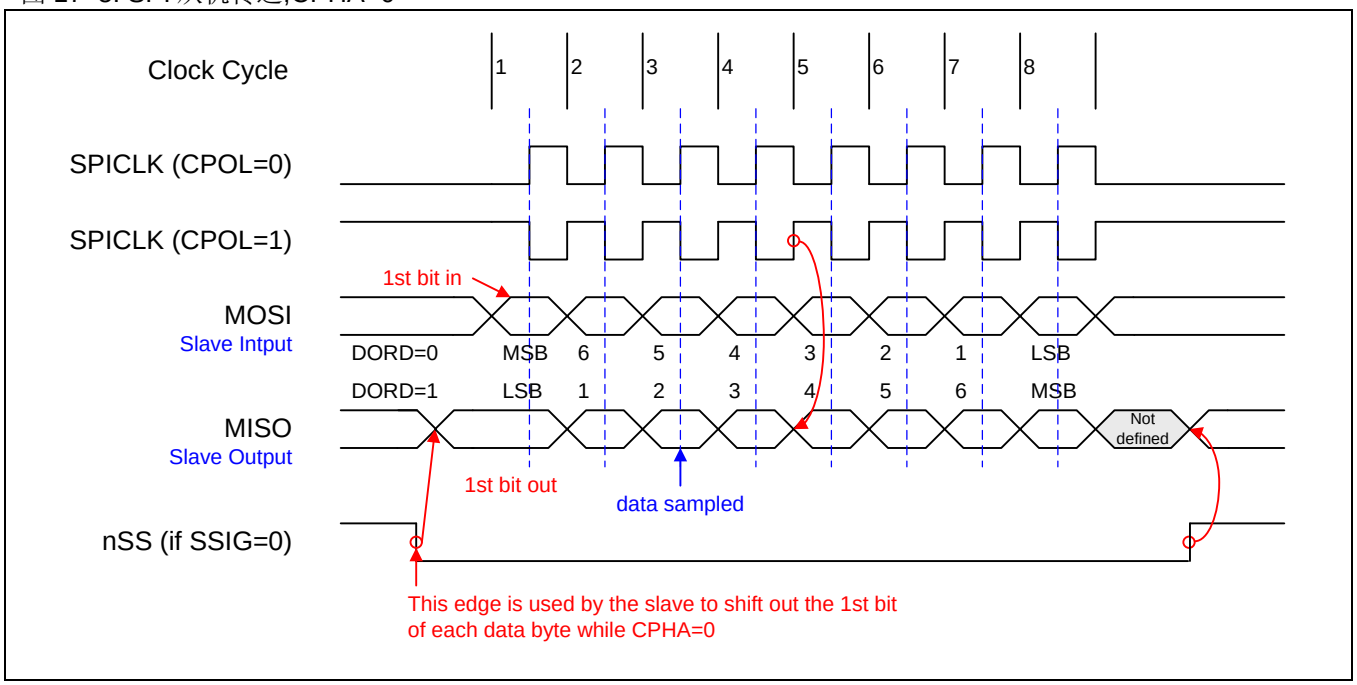



图 17–6. SPI 从机传送,CPHA=1

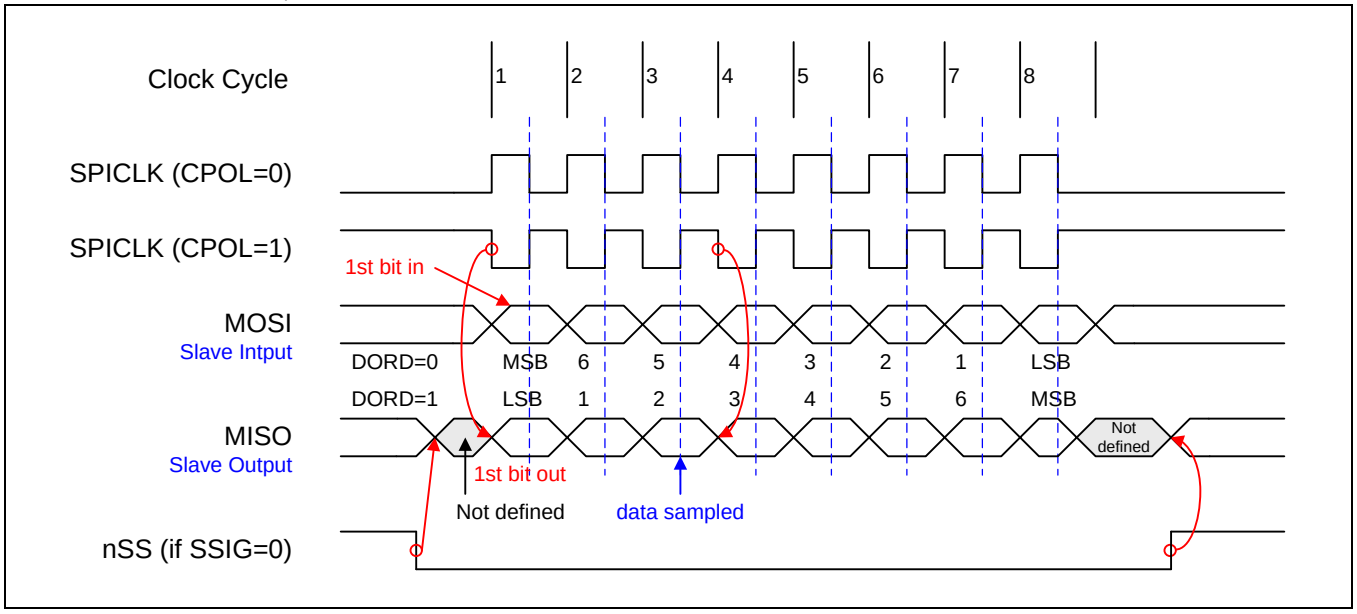


图 17–7. SPI SPI 主机传送, CPHA=0

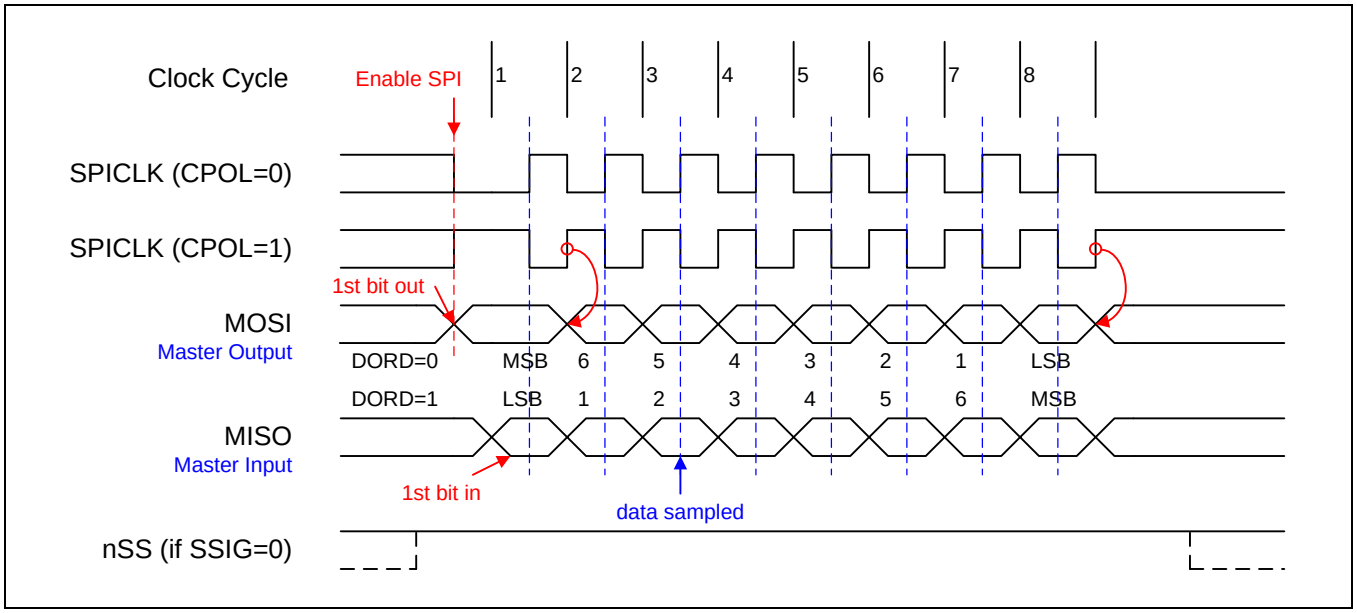
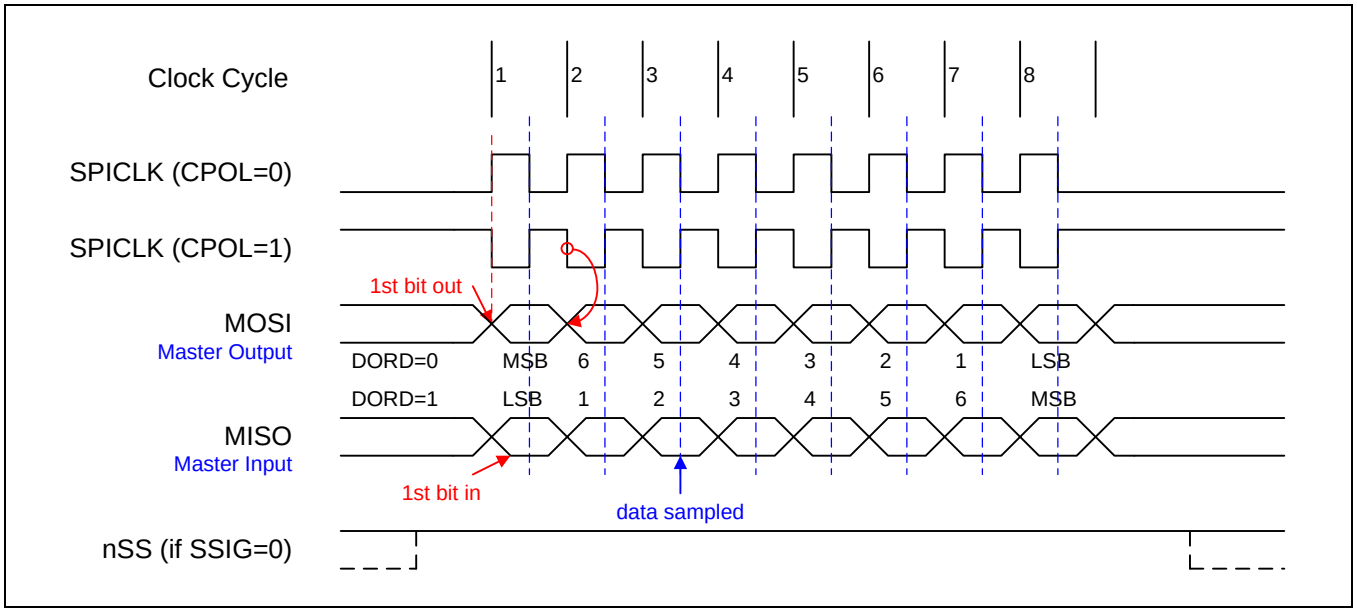



图 17–8. SPI 主机传送, CPHA=1



## 17.4. SPI 寄存器

下面是 SPI 操作相关寄存器:

SPCON: SPI 控制寄存器

SFR 地址 = 0x85 复位值= 0000-0100

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

Bit 7: SSIG, 忽略 nSS
0: 若 SSIG=0, nSS 位决定该设备是主机还是从机。
1: 若 SSIG=1, MSTR 位决定该设备是主机还是从机。

Bit 6: SPEN, SPI 使能。
0: 若 SPEN=0, SPI 接口禁用，所有 SPI 引脚可作为通用 I/O 口使用。
1: 若 SPEN=1, SPI 功能打开。

Bit 5: DORD, SPI 数据顺序。
0: 传送数据时先传数据字节最高位。
1: 传送数据时先传数据字节最低位。

Bit 4: MSTR, 主机/从机模式选择
0: 选择 SPI 从机模式。
1: 选择 SPI 主机模式。

Bit 3: CPOL: SPI 时钟极性选择

0: SPICLK 待机是为低电平，SPICLK 时钟脉冲前沿是上升沿，而后沿是下降沿。
1: SPICLK 待机是为高电平，SPICLK 时钟脉冲前沿是下降沿，而后沿是上升沿。

Bit 2: CPHA: SPI 时钟相位选择
0: nSS 引脚低电平 (SSIG=0)开始放数据并在 SPICLK 后沿改变数据. 数据在 SPICLK 的前沿采样。
1: SPICLK 脉冲前沿放数据，后沿采样。
(注 : 若 SSIG=1, CPHA 必须不为 1，否则就是为非定义操作。)

Bit 1~0: SPR1-SPR0, SPI 时钟速率选择 (主机模式)
00: SYSCLK/4
01: SYSCLK/16
10: SYSCLK/64
11: SYSCLK/128 (*Fosc 是系统时钟。*)

SPSTAT: SPI 状态寄存器

SFR 地址 = 0x84 复位值= 00XX-XXXX

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

Bit 7: SPIF, SPI 传输完成标志。
0: 当软件写‘1’到此位会自动清除 SPIF 标志， 软件写 “0” 无效。
1: 当一次串行传输完成时，SPIF 位置位，同时若 SPI 中断允许，会产生一个中断。若 nSS 引脚在主机模式下被拉低且 SSIG=0，SPIF 位也会置位以表明“模式改变”。SPIF 标志通过软件在该位写入“1”来清零。

Bit 6: WCOL, SPI 写冲突标志。
0: WCOL 标志通过软件在该位写入“1”来清零。 Software 软件写 “0” 无效。
1: SPI 数据寄存器 SPDAT 在数据传输过程中被写入时，WCOL 置位。 (参考章节 17.2.4: 写冲突)。

Bit 5~0: 保留。 当写 SPSTAT 寄存器时此几位必须写 “0”。

SPDAT: SPI 数据寄存器
SFR 地址 = 0x86 复位值= 0000-0000

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

SPDAT 有两个物理缓冲器供传输过程中写入和读取，一个写入缓冲器，一个读取缓冲器。

## 17.5. SPI 示例代码

(1). 功能需求: SPI 主机读和写，采样数据在上升沿并且时钟前沿是上升沿。

汇编语言范例:

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

Initial_SPI:                                        ;初始化 SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR)             ;使能 SPI 主模式
   RET

SPI_Write:
   MOV    SPIDAT, R7                                ;写自变量 R7
wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                         ;等待传送完成
   ANL    SPISTAT, #(0FFh - SPIF)                   ;清除 SPI 中断标志
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                             ;触发 SPI 读
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                          ;等待读完成
   ANL    SPISTAT, #(0FFh - SPIF)                   ;清楚 SPI 中断标志
   MOV    A, SPIDAT                                 ;移动读的数据到 A
   RET
```

C 语言范例：

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

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR);                  // 使能 SPI 主机模式
}
void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                    //写自变量
   while(!(SPISTAT & SPIF));                        //等待传送完成
   SPISTAT &= ~SPIF;                                //清楚 SPI 中断标志
}
unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                   //触发 SPI 读
   while(!SPISTAT & SPIF);                          //等待传送完成
   SPISTAT &= ~SPIF;                                //清除 SPI 中断标志
   return SPIDAT;
}
```

(2). 功能需求： SPI 主机读和写，采样数据在上升沿和时钟前沿为下降沿。

汇编语言范例:

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

Initial_SPI:                                            ;初始化 SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR + CPOL)          ;使能 SPI 主机模式
   RET

SPI_Write:
   MOV    SPIDAT, R7                                    ;写自变量 R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                             ;等待传送完成
   ANL    SPISTAT, #(0FFh - SPIF)                       ;清除 SPI 中断标志
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                                 ;触发 SPI 读
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                              ;等待读完成
   ANL    SPISTAT, #(0FFh - SPIF)                       ;清除 SPI 中断标志
   MOV    A, SPIDAT                                     ;转移读的数据到 A
   RET
```

C 语言范例:

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

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR | CPOL);               // 使能 SPI 主机模式
}
void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                        //写自变量
   while(!(SPISTAT & SPIF));                            //等待传送完成
   SPISTAT &= ~SPIF;                                    //清除 SPI 中断标志
}
unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                       //触发 SPI 读
   while(!SPISTAT & SPIF);                              //等待读完成
   SPISTAT &= ~SPIF;                                    //清除 SPI 中断标志
   return SPIDAT;
}
```

(3). 功能需求: SPI 主机读和写,采样数据在下降沿并且时钟前言是上升沿。

汇编语言代码范例:

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

Initial_SPI:                                        ; 初始化 SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR + CPHA)      ; 使能 SPI 主机模式
   RET

SPI_Write:
   MOV    SPIDAT, R7                                ; 写自变量 R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                         ; 等待传送完成
   ANL    SPISTAT, #(0FFh - SPIF)                   ; 清除 SPI 中断标志
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                             ; 触发 SPI 读
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                          ; 等待读完成
   ANL    SPISTAT, #(0FFh - SPIF)                   ; 清除 SPI 中断标志
   MOV    A, SPIDAT                                 ;转移读的数据到 A
   RET
```

C 语言代码范例：

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

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR | CPHA);           //使能 SPI 主机模式
}

void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                    //写自变量
   while(!(SPISTAT & SPIF));                        //等待传送完成
   SPISTAT &= ~SPIF;                                //清除 SPI 中断标志
}
unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                   //触发 SPI 读
   while(!SPISTAT & SPIF);                          //等待读完成
   SPISTAT &= ~SPIF;                                //清除 SPI 中断标志
   return SPIDAT;
}
```

(4). 功能需求: SPI 主机读和写,采样数据在下降沿并且时钟前言是下降沿。

汇编语言代码范例:

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

Initial_SPI:                                                    ;初始化 SPI
   ORL    SPICTL, #(SSIG + SPEN + MSTR + CPOL + CPHA)    ;使能 SPI 主机模式
   RET

SPI_Write:
   MOV    SPIDAT, R7                                       ;写自变量 R7
 wait_write:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_write                                ; 等待传送完成
   ANL    SPISTAT, #(0FFh - SPIF)                          ; 清除 SPI 中断标志
   RET

SPI_Read:
   MOV    SPIDAT, #0FFh                                    ;触发 SPI 读
 wait_read:
   MOV    A, SPISTAT
   JNB    ACC.7, wait_read                                 ; 等待读完成
   ANL    SPISTAT, #(0FFh - SPIF)                          ; 清除 SPI 中断标志
   MOV    A, SPIDAT                                        ; 转移读的数据到 A
   RET
```

C 语言代码范例:

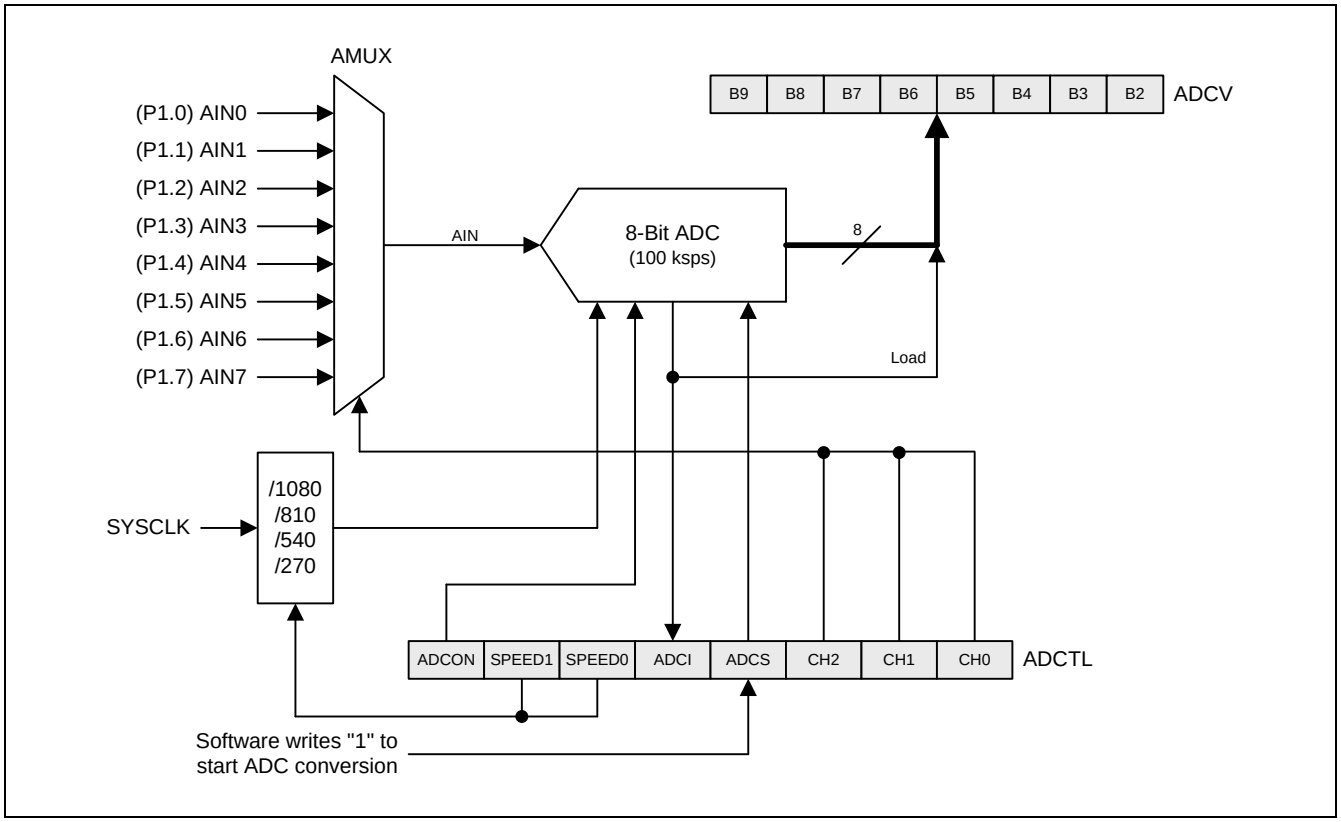
```
#define CPHA                    0x04
#define CPOL                    0x08
#define MSTR                    0x10
#define SPEN                    0x40
#define SSIG                    0x80
#define SPIF                    0x80

void Initial_SPI(void)
{
   SPICTL |= (SSIG | SPEN | MSTR | CPOL | CPHA);          //使能 SPI 主机模式
}

void SPI_Write(unsigned char arg)
{
   SPIDAT = arg;                                                //写自变量
   while(!(SPISTAT & SPIF));                                    //等待传送完成
   SPISTAT &= ~SPIF;                                            //清除 SPI 中断标志
}
unsigned char SPI_Read(void)
{
   SPIDAT = 0xFF;                                               //触发 SPI 读
   while(!SPISTAT & SPIF);                                      //等待读完成
   SPISTAT &= ~SPIF;                                            //清除 SPI 中断标志
   return SPIDAT;
}
```

# 18. 8-位 ADC

**MA801_MA802** 的 ADC 子系统是由一个模拟多路复用器（AMUX）和一个 100Ksps,8 位逐次逼近型 ADC 组成。如图 18–1 所示，AMUX 可以经由特殊功能寄存器去配置。ADC 操作在单端模式，可以配置成去测量 P1 口的任何引脚。ADC 子系统仅当 ADC 控制寄存器的 ADEN 位设置成逻辑 1 时被使能，当被设置成逻辑 0 时，ADC 子系统进入低功耗关闭状态。

## 18.1. ADC 结构

图 18–1. ADC 方框图




## 18.2. ADC 工作

ADC 的最大转换速度是 100 ksps. ADC 转换时钟由系统时钟分频而来，由寄存器 ADCTL 的 SPEED1~0 位决定。转换完成后(ADCI 是高),在 ADC 结果寄存器（ADCV）里可以找到转换的结果。对于单端转换，结果是：

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

### 18.2.1. ADC 输入通道

模拟多路复用器(AMUX)选择用于 ADC 的输入，允许 P1 口的任一引脚作为单端模式加以测量。如图 18–1 所述，通过寄存器 ADCTL 的 CHS2-0 位配置和选择 ADC 输入通道。 选择的脚相对于 GND 进行测量。

### 18.2.2. 启动一个转换

在使用 ADC 之前，用户应当:

1) 设置 ADCON 这个位来打开 ADC 硬件。
2) 通过位 SPEED1 和 SPEED0 来配置 ADC 输入时钟。
3) 通过位 CHS2,CHS1 和 CHS0 来选择模拟输入通道。
4) 通过 P1,P1M0 和 P1AIO 配置已选择的输入脚（和 P1 共享）通过设置 P1M0 和 P1M1 为仅输入模式。

现在，用户可以置 ADCS 位来开始 AD 转换。转换时间由 SPEED1 和 SPEED0 控制。一旦转换完成，硬件将自动清 ADCS 位，置位中断标志为 ADCI，同时将 8 位的转换结果加载到 ADCV。

如上所述，中断标志位 ADCI，当硬件置位后，表示了一个完整的转换。因此可以通过两个方式来检测转换是否完成：（1）总是用软件轮询中断标志位 ADCI。（2）通过置位 EADCI ( AUXR 寄存器), ESPI_ADC (IE 寄存器)和 EA(在 IE 寄存器)使能 ADC 中断，当转换完成后，CPU 会跳到中断服务程序。无论是(1)还是(2),ADCI 标志位都应当在下一个转换之前由软件清零。

### 18.2.3. ADC 转换时间

用户可以依照模拟输入信号的频率选择适当的转换速度。通过配置 ADCTL 中的 SPEED1~0 来指定转换率。例如，如果 SYSCLK =25MHz 并且选择了 SPEED[1:0] = SYSCLK/270，那么模拟输入信号的最大转换频率为 92.6KHz
(转换率 = 25MHz/270 = 92.6KHz.)

### 18.2.4. I/O 引脚用于 ADC 功能

用于 A/D 转换的模拟输入引脚也有 I/O 口的数字输入和输出功能。为了得到更好的模拟性能，用于 ADC 的引脚应当禁止它的数字输出功能。它应当设置成仅输入模式，参考章节“12 I/O 端口配置”。

### 18.2.5. 空闲和掉电模式

在掉电模式，ADC 不能工作。如果 A/D 打开了，它将浪费一些功耗。因此，在进入空闲模式和掉电模式之前关掉 ADC 硬件（ADEN=0）将会节省一些功耗。

## 18.3. ADC 寄存器

ADCTL: ADC 控制寄存器

SFR 地址 = 0xC5 复位值 = 0000-0000

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

Bit 7: ADCON, ADC 使能.

0: 清零关闭 ADC 模块

1: 置位打开 ADC 模块，置位 ADCS 之前必须至少有 5US 的 ADC 使能时间。

Bit 6~5: SPEED1 和 SPEED0, ADC 转换速度控制。

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

推荐 ADC 时钟不要超过 12MHz。

Bit 4: ADCI, ADC 中断标志位.

0: 这位必须软件清零。

1: A/D 转换完成后置位这个位，如果 ADC 中断使能，则会进入 ADC 中断。

Bit 3: ADCS. ADC 转换启始

0: ADCS 不能被软件清零

1:软件置这个位来开始一个 A/D 转换。转换完成后，ADC 硬件将清除这个位，并且置位 ADCI，当 ADCS 或 ADCI 为高时，不能启动一个新的转换。

Bit 2~0: CHS2 ~ CHS1, ADC 模拟转换器输入通道选项。

| CHS[2:0] | 通道选择 |
|---|---|
| 0 0 0 | AIN0 (P1.0) |
| 0 0 1 | AIN1 (P1.1) |
| 0 1 0 | AIN2 (P1.2) |
| 0 1 1 | AIN3 (P1.3) |
| 1 0 0 | AIN4 (P1.4) |
| 1 0 1 | AIN5 (P1.5) |
| 1 1 0 | AIN6 (P1.6) |
| 1 1 1 | AIN7 (P1.7) |

ADCV: ADC 结果寄存器

SFR 地址 = 0xC6 复位值 = xxxx-xxxx

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ADCV.7 | ADCV.6 | ADCV.5 | ADCV.4 | ADCV.3 | ADCV.2 | ADCV.1 | ADCV.0 |
| R | R | R | R | R | R | R | R |

**MA801_MA802** 转换结果是一个 8 位无符号整型数据。输入测量值由'0'到 VDD x 255/256 计算，结果示例见下表：

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

## 18.4. ADC 示例代码

(1). 功能需求: ADC 示例代码为系统时钟 SYSCLK=24MHz， 模拟输入是 P1.0/P1.1/P1.2 且 SPEED[1:0]=SYSCLK/270 为 88.9KHz 转换率。

汇编语言代码范例:

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

INITIAL_ADC_PIN:
        ORL     P1M0, #00000111B            ; P1.0, P1.1, P1.2 = 仅输入模式
        ANL     P1M1,#11111000B
        MOV     ADCTL,#ADCON                ; 使能 ADC 模块
        ; delay 5us
        ; call ....

Get_P10:
        MOV     ADCTL, #(ADCON + SPEED1 + SPEED0)
                                            ; 使能 ADC 模块和启动转换
                                          ; 转换速度 114.3k @ 24MHz, 选择 P1.0 为 ADC 输出引脚
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     A, ADCTL                    ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCTL,#(0FFh - ADCI - ADCS) ; clear ADCI & ADCS
        MOV     AIN0_data_V,ADCV            ; 保存 P1.0 ADC 数据
        ; to do ...
Get_P11:
        MOV     ADCTL,#(ADCON + SPEED1 + SPEED0 + CHS0) ; 选择 P1.1
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     A, ADCTL                    ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCTL,# (0FFh - ADCI - ADCS) ; 清除 ADCI & ADCS
        MOV     AIN1_data_V,ADCV
    ; to do ...
Get_P12:
        MOV     ADCTL,#(ADCON + SPEED1 + SPEED0 + CHS1) ; 选择 P1.2
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     ACC,ADCTL                   ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCTL,# (0FFh - ADCI - ADCS) ; 清除 ADCI & ADCS
        MOV     AIN2_data_V,ADCV

    ; to do ...
        RET
```

C 语言代码范例:

```
#define CHS0                    0x01
#define CHS1                    0x02
#define ADCS                    0x08
#define ADCI                    0x10
#define SPEED0                  0x20
#define SPEED1                  0x40
#define ADCON                   0x80

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

        P1M0 |= 0x07;                                   // P1.0, P1.1, P1.2 = 仅输入模式
        P1M1 &= ~0x07;

        ADCTL = ADCON;                                  //使能 ADC 模块
        // delay 5us
        // ...

        // select P1.0
        ADCTL = (ADCON | SPEED1 | SPEED0);
                                                        //使能 ADC 模块和启动转换
                                                // 转换速度为 114.3k @ 24MHz, 选择 P1.0 为 ADC 输入脚
    Delay_5us();
    ADCTL |= ADCS;

        while ((ADCTL & ADCI) == 0x00);                 //等待完成
        ADCTL &= ~(ADCI | ADCS);
        AIN0_data_V = ADCV;

        // to do ...

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

        while ((ADCTL & ADCI) == 0x00);                 //等待完成
        ADCTL &= ~(ADCI | ADCS);
        AIN1_data_V = ADCV;

        // to do ...

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

        while ((ADCTL & ADCI) == 0x00);                 //等待完成
        ADCTL &= ~(ADCI | ADCS);
        AIN2_data_V = ADCV;

        // to do ...

        while (1);
}
```

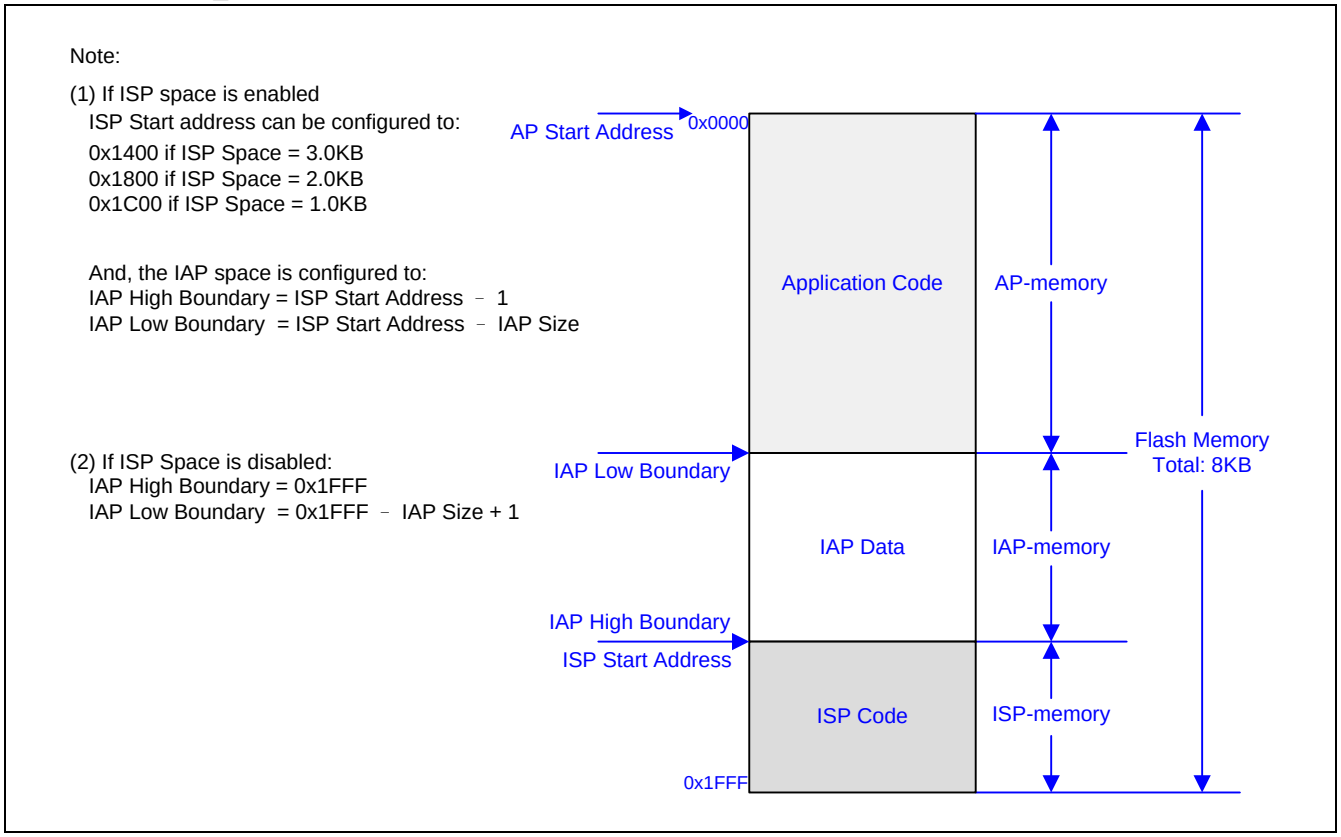
# 19. ISP 和 IAP

**MA801_MA802** 的 Flash 存储空间划分为 AP 存储空间，IAP 存储空间和 ISP 存储空间。AP 存储空间用来存储用户应用程序；IAP 存储空间用来存储非易失性数据；ISP 存储空间用来存储在线编程的引导码。系统在 ISP 空间运行时，MCU 可以修改 AP 和 IAP 来更新软件。如果 MCU 运行在 AP 空间，那么 MCU 就仅能修改 IAP 存储的数据。

## 19.1. MA801_MA802 Flash 存储空间配置

**MA801_MA802** 总共有 8K 字节的 Flash，图 19–1 显示了 Flash 的配置。ISP 存储空间可以被禁止或由硬件选项配置最大 **3K** 字节。IAP 存储空间大小由 IAP 低边界和高边界决定。IAP 低边界由 IAPLB 寄存器的值决定。IAP 高边界与 ISP 的起始地址相关，ISP 存储空间由硬件选项决定。IAPLB 寄存器值由硬件选项配置或 AP 软件编程设定。所有 AP, IAP 和 ISP 存储空间共享总 8K 字节的存储空间。

图 19–1. **MA801_MA802** Flash 存储空间配置




注意:

笙泉公司 **MA801_MA802** 样品的默认设置是：**1K ISP, 1K IAP** 和加密。**1K** ISP 文件是有笙泉专利的通过一条线就能在线下载的 1-线 ISP 协议。1K IAP 大小可以通过修改 IAPLB 来重新配置。

## 19.2. MA801_MA802 Flash 在 ISP/IAP 的访问

**MA801_MA802** 给 ISP 和 IAP 应用提供三种 Flash 访问模式：页擦除模式，编程模式及读取模式。MCU 软件使用这三种模式去更新 Flash 的数据和获取 Flash 的数据。本章展示了不同 Flash 模式的流程图和范例代码。

### 19.2.1. ISP/IAP Flash 页擦除模式

**MA801_MA802** Flash 数据中的任何位只能写入“0”。如果使用者想写“1”到 Flash 数据位中，必须先擦除。但是 **MA801_MA802** ISP/IAP 操作仅仅支持“页擦除”模式，页擦除会使整页中所有的数据位变为“1”。
**MA801_MA802** 的每页有 512 字节，页开始地址应该对齐到 A8~A0 = 0x000。目标地址定义在 IFADRH 和 IFADRL 中，所以，在 Flash 页擦除模式，IFADRH.0(A8) 和 IFADRL.7~0(A7~A0) 必须写“0” 为正确的页地址选择。图 19–2 显示 ISPIAP 页擦除操作的流程。

图 19–2. ISP/IAP 页擦除流程

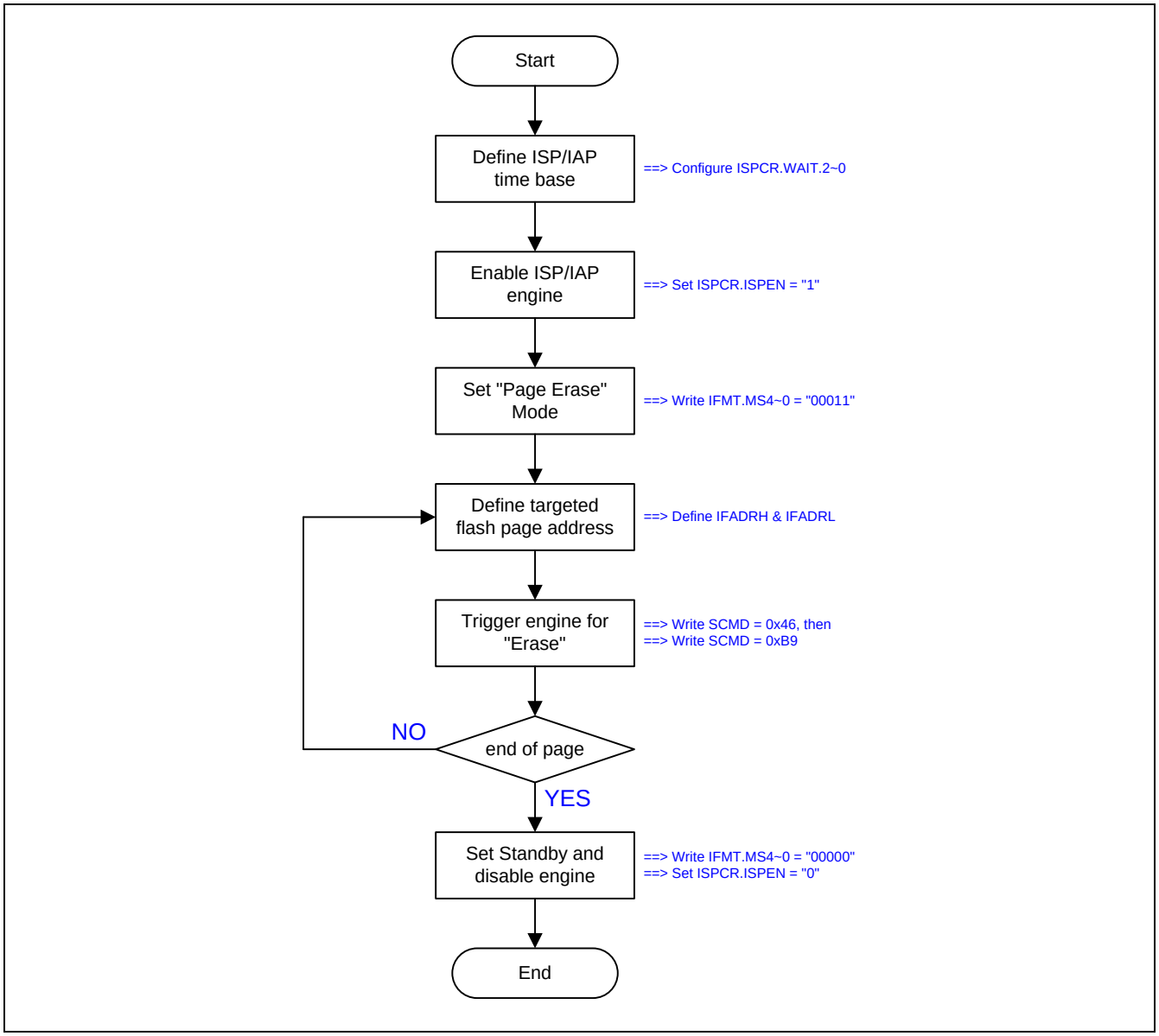


图 19–3 显示 ISP/IAP 页檫除操作示例代码。

图 19–3. ISP/IAP 页檫除汇编语言代码范例

```
MOV    ISPCR,#00010111b ; XCKS4~0 = 十进制 23 相当于 OSCin = 24MHz

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

MOV    IFMT,#03h        ; 选择页檫除模式

MOV    IFADRH,??        ; 填写 [IFADRH,IFADRL] 页地址
MOV    IFADRL,??        ;

MOV    SCMD,#46h        ; 触发 ISP/IAP 处理
MOV    SCMD,#0B9h       ;

; 此时，MCU 会停止在这里直到事件完成。

MOV    IFMT,#00h        ; 选择待机模式
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

### 19.2.2. ISP/IAP Flash 写模式

**MA801_MA802** 写模式提供 Flash 存储空间的字节写操作来更新数据。IFADRH 和 IFADRL 指向 Flash 的物理字节地址。IFD 存储编程到 Flash 的内容。 图 19–4 展示了 ISP/IAP 操作的 Flash 字节编程流程。

图 19–4. ISP/IAP 字节写流程

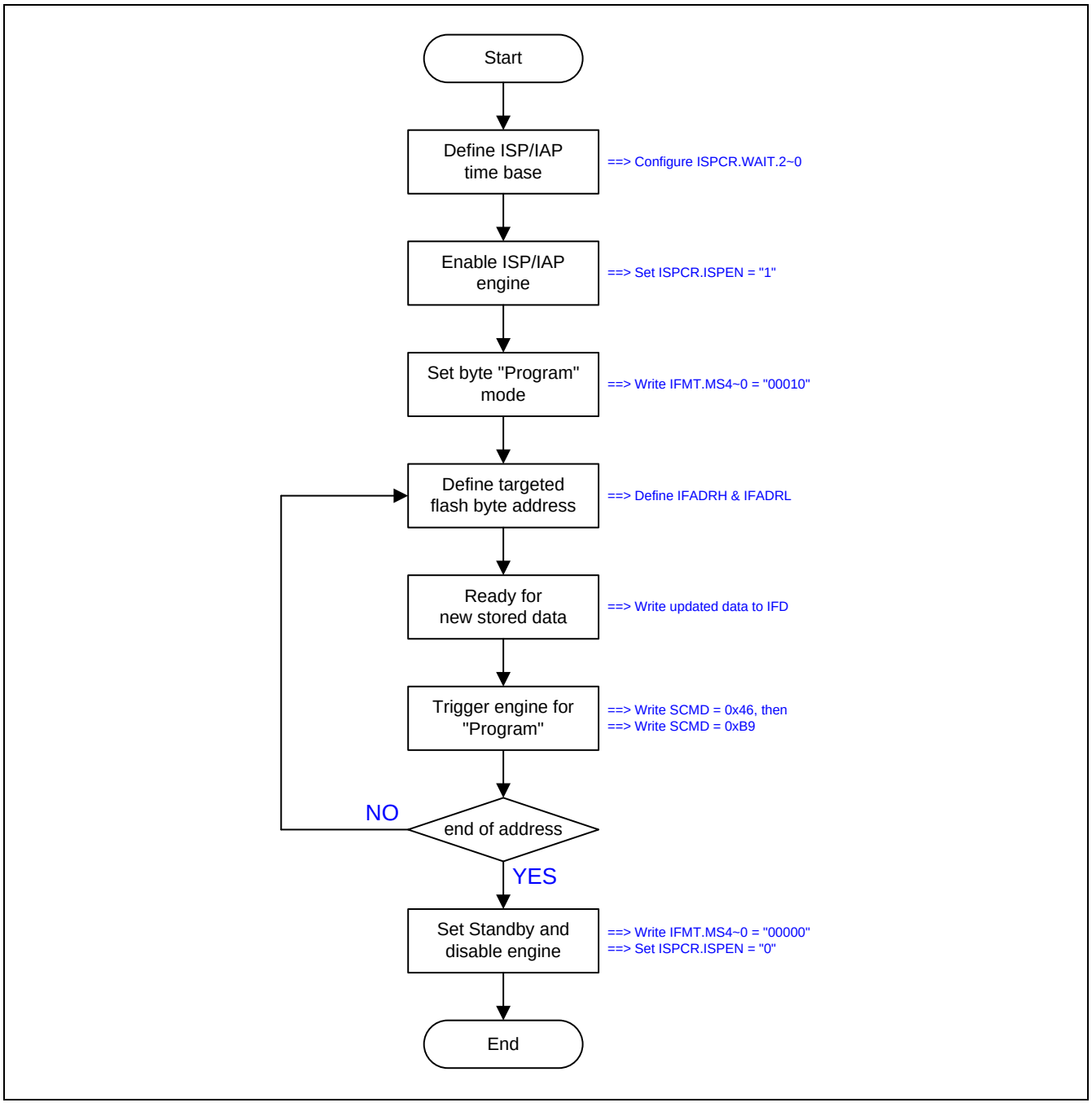


图 19–5 显示 ISP/IAP 字节写操作的示例代码。

图 19–5. ISP/IAP 字节写操作的汇编示例代码

```
MOV   ISPCR,#00010111b ; XCKS4~0 = 十进制 23 相当于 OSCin = 24MHz

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

MOV   IFMT,#02h        ; 选择写模式

MOV   IFADRH,??        ; 填写 [IFADRH,IFADRL] 字节地址
MOV   IFADRL,??        ;

MOV   IFD,??           ; 填写 IFD 要写入的数据

MOV   SCMD,#46h        ;触发 ISP/IAP 处理
MOV   SCMD,#0B9h       ;

; 此时，MCU 会停止在这里直到事件完成。

MOV   IFMT,#00h        ; 选择待机模式
MOV   ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

### 19.2.3. ISP/IAP Flash 读取模式

**MA801_MA802** 读取模式提供从 Flash 存储空间获取已存储数据的字节读取操作。IFADRH 和 IFADRL 指向 Flash 的物理字节地址。IFD 存储从 Flash 读取到的内容。建议在数据编程或页擦除之后通过读取模式核对 Flash 数据。图 19–6 展示了 ISP/IAP 操作下的 Flash 字节读取流程。

图 19–6. ISP/IAP 字节读流程

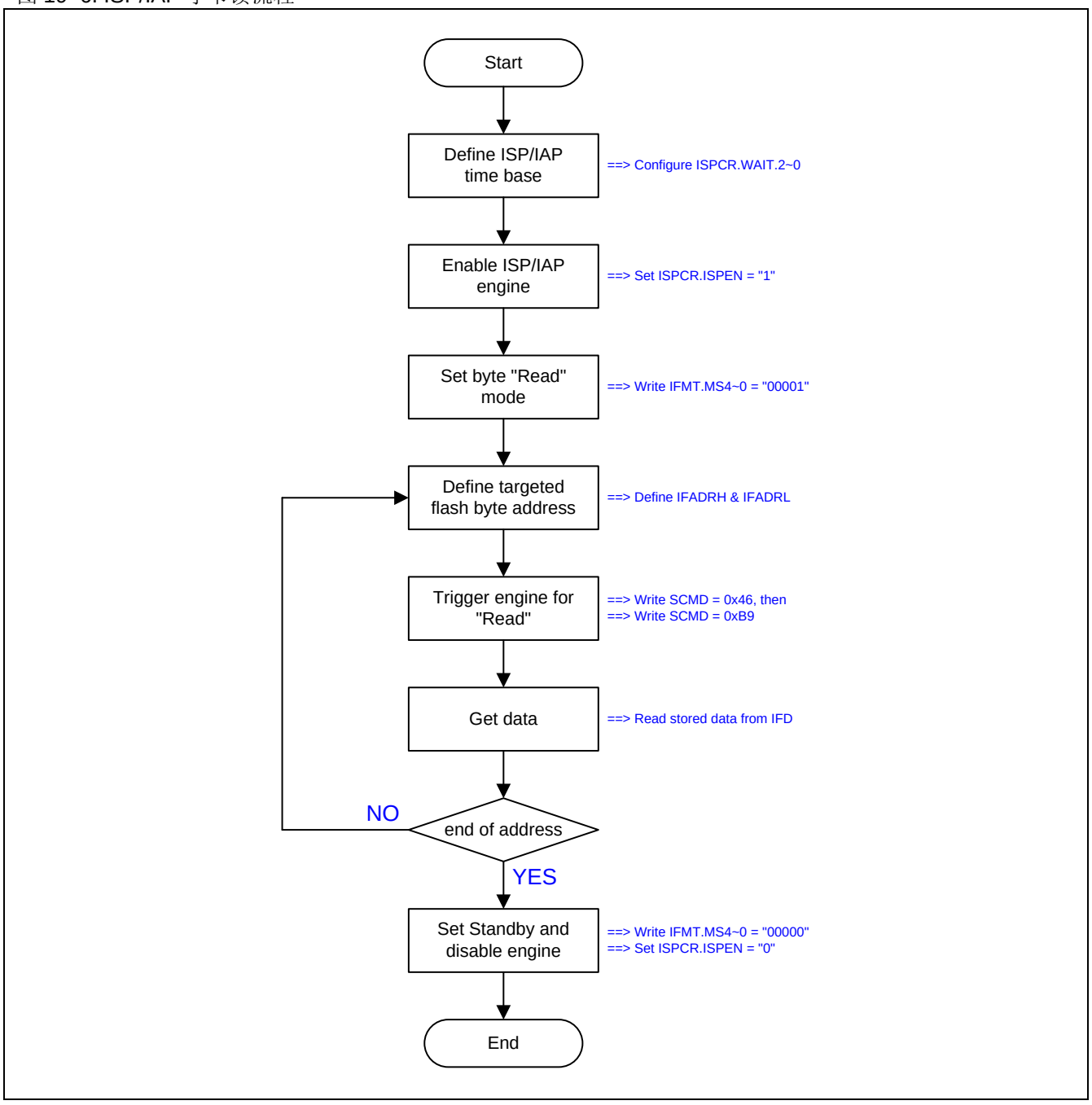


图 19–7 显示 ISP/IAP 字节读的示例代码

图 19–7. ISP/IAP 字节读的范例汇编程序

```
MOV    ISPCR,#00010111b ; XCKS4~0 = 十进制 23 相当于 OSCin = 24MHz

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

MOV    IFMT,#01h        ; 选择读取模式

MOV    IFADRH,??        ; 填写 [IFADRH,IFADRL] 的字节地址
MOV    IFADRL,??        ;

MOV    SCMD,#46h        ; 触发 ISP/IAP 处理
MOV    SCMD,#0B9h       ;

; 此时，MCU 会停止在这里直到事件完成。

MOV    A,IFD            ; 此时,读取的数据存在 IFD

MOV    IFMT,#00h        ; 选择备用模式
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

## 19.3. ISP 操作

ISP 意指在系统可编程，不需要在实际的终端产品上移除 MCU 芯片就可以更新用户的应用程序(AP 存储空间)和非易失性应用数据(IAP 存储空间)。这个可使用性就有一个宽的现场应用范围。ISP 模式使用引导程序来编程 AP 存储空间和 IAP 存储空间。

注意:
(1) 在用 ISP 功能之前，使用者必须先配置 ISP-存储器空间并用通用烧写器或笙泉的烧写器插入 ISP 文件到 ISP-存储器中。
(2) ISP-存储器中的 ISP 文件 code 只能下载 AP-存储器和非易失的 IAP-存储器。

在 ISP 操作完成之后,软件写 “001”到 ISPCR.7 ~ ISPCR.5 这样会触发一个软件**复位**并且使 CPU 再启动到应用程序存储空间(AP)的 0x0000 地址。

如我们所知,ISP 代码的作用就是编程 AP 存储空间和 IAP 存储空间。因此，***MCU 为了执行 ISP 代码必须从 ISP 存储空间启动***。根据 MCU 如何从 ISP 存储空间启动，有两种方法执行在系统可编程。

### 19.3.1. 硬件访问 ISP

在上电复位时为了使 MCU 直接从 ISP 存储空间启动，MCU 的硬件选项 *HWBS* 和 *ISP 存储空间*必须使能。硬件选项的 ISP 进入方法叫做硬件访问。一旦 *HWBS* 和 *ISP 存储空间*使能,当上电复位时 MCU 总是从 ISP 存储空间启动去执行 ISP 代码(引导程序)。ISP 代码做的第一件事是核对是否有 ISP 请求。如果没有 ISP 请求,ISP 代码触发软件复位(设置 ISPCR.7~5 为“101”)使 MCU 在启动到 AP 存储空间去运行用户应用程序。

下列范例描述怎样从 ISP 存储区到 AP 存储区：

从 ISP 复位到 AP 或从 AP 复位到 AP 的范例

汇编程序范例：

```
SWRST   EQU     20h
SWBS    EQU     40h

        ANL     ISPCR, #(0FFh - SWBS)   ;清除 SWBS
        ORL     ISPCR, #SWRST           ;触发软件复位
```

C 程序范例：

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

        ISPCR &= ~SWBS;                 // 清除 SWBS
        ISPCR |= SWRST;                 //触发软件复位
```

### 19.3.2. 软件访问 ISP

当 MCU 运行在 AP 存储空间时，软件访问 ISP 通过触发软件复位使 MCU 从 ISP 存储空间启动。这种情况，HWBS 不用使能。仅有的方法是当 MCU 运行在 AP 存储空间时同时设置 ISPCR.7~5 为“111”触发软件复位 MCU 从 ISP 存储空间启动。注意： ISP 存储空间必须通过硬件选项配置一个有效空间来保留 ISP 模式给软件访问 ISP 应用。

下面的范例描述怎样离开 AP 到 ISP 程序：

从 AP 复位 ISP 示例代码

汇编语言代码范例:

```
SWRST       EQU             20h
SWBS        EQU             40h

        ORL         ISPCR, #( SWBS + SWRST)                 ;设置 SWBS 并触发软件复位
```

C 语言代码范例:

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

        ISPCR |=(SWBS + SWRST);                             //设置 SWBS 并触发软件复位
```

### 19.3.3. ISP 注意事项

开发 ISP 代码

尽管 ISP 存储空间的 ISP 代码是可编程的，ISP 存储空间在 MCU 的 Flash 中有一个 *ISP 起始地址*(见 图 19–1), 但是并不意味着你需要在你的源代码中加入这个偏移量 ( *ISP 起始地址*)。代码偏移量硬件自动处理。用户只需像在 AP 存储空间开发应用程序一样开发。

ISP 期间中断

在触发 ISP/IAP flash 处理之后，内部 ISP 处理时 MCU 将停止一会儿直到处理完成。此时，如果中断已使能则中断事件将排队等待服务。一旦 ISP/IAP flash 处理完成，MCU 继续运行并且如果中断标志仍然有效则排队中的中断将立即服务。不过用户需要意识到下列事项：

(1) 当 MCU 停止在 ISP 处理时，中断不能实时服务。
(2) 低/高电平触发外部中断 nINTx,必须保持到 ISP 处理完成，否则将被忽略。

ISP 和空闲模式

**MA801_MA802** 不使用空闲模式执行 ISP 功能。反而 ISP/IAP 引擎操作 Flash 存储空间将冻结 CPU 的运行。一旦 ISP/IAP 运行结束，CPU 将继续并且推进紧跟着 ISP/AP 激活的指令。

ISP 的访问目标

如前所述, ISP 用来编程 AP 存储空间和 IAP 存储空间。一旦访问目标地址超出 IAP 存储空间的最后一个字节之外, 硬件将自动忽略 ISP 处理的触发。这样 ISP 触发是无效的并且硬件不做任何事情。

ISP 的 Flash 持久期

内置 Flash 的持久期是 20,000 写周期，换句话说写周期不能超过 20,000 次。这样用户必须注意应用中需要频繁更新 AP 存储空间和 IAP 存储空间这一点。

### 19.3.4. MA801_MA802 默认 ISP 代码

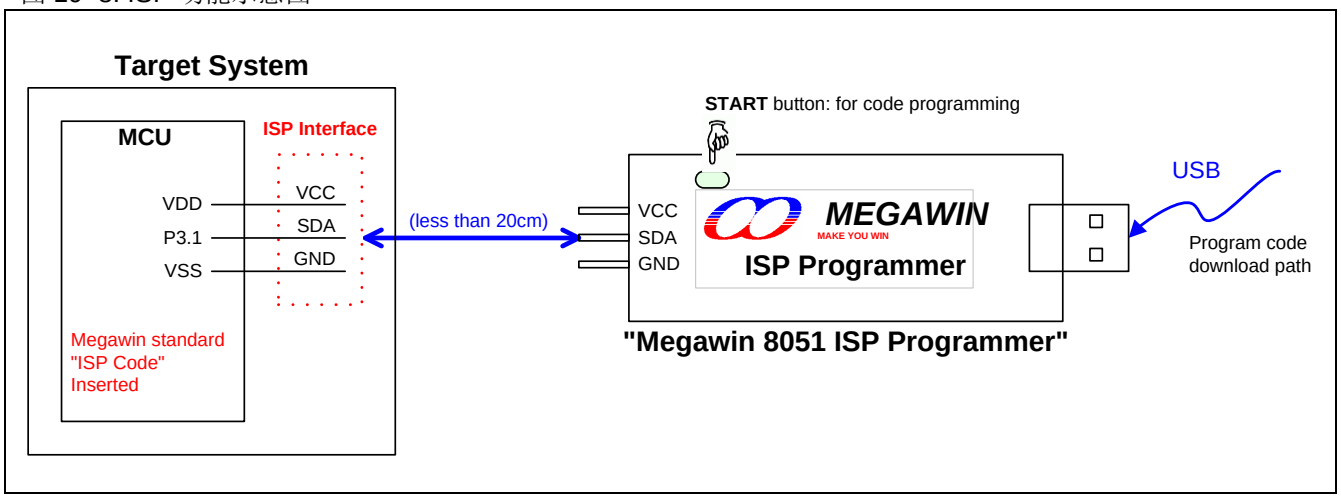
虽然使用者可以使用自己的 ISP 代码， MA801_MA802 在出厂前已经植入了有笙泉知识产权的 ISP 代码。笙泉提供工具， 笙泉的 8051 ISP 烧录器， 去执行在系统升级 AP 应用程序。这个章节简短的描述笙泉的 ISP 工具。

特征

标准的 'ISP code' 在出厂之前已经植入。
仅仅只用的 Only one port pin (P3.1) 作为 ISP 的接口。
工作不受震荡频率影响。
可以脱机烧录。

ISP 工具以上特征对使用者来说是非常有益的，特别，程序下载以后能脱机工作，这是特别有用在没有电脑的领域，
"笙泉 8051 ISP 工具"示意图如图 19–8。仅仅 3 条线用在 ISP 接口：SDA 传输数据从 ISP 工具到目标芯片；***VCC*** & ***GND*** 作为 ISP 工具的电源供应器。章节"22.4 ISP 接口电路"描述 ISP 接口电路如何应用在使用者的系统中。ISP 的 USB 连接器可以直接插入 PC 的 USB 口来从 PC 下载应用程序到 ISP 工具。

图 19–8. ISP 功能示意图



## 19.4. 在应用可编程(IAP)

**MA801_MA802** 内建一个在应用可编程(IAP)功能, 当应用程序运行时在 Flash 存储空间里允许一些区域被应用成非易失性数据存储区。这个有用特点能使用在断电后还需要保存数据的应用中。这样不需要使用外部的串行 EEPROM (比如 93C46, 24C01, .., 等等)来保存非易失性的数据。

事实上，IAP 的操作除了 Flash 存储空间被划分在不同的区域之外与 ISP 一样。ISP 操作的可编程 Flash 范围在 AP 存储空间和 IAP 存储空间，而 IAP 操作的范围只在 IAP 存储空间。

注意:
(1) **MA801_MA802** 的 IAP 特点，软件通过写 SFR P 页的 IAPLB 寄存器声明 IAP 存储空间。IAP 存储空间也可以通过通用的烧入器/编程器或笙泉专利的烧入器/编程器来配置 IAPLB 的初始值。
(2) 执行 IAP 的程序代码是在 AP 存储空间并且仅能编程 IAP 存储空间而不能编程 ISP 存储空间。

### 19.4.1. IAP-存储空间边界/范围

如果 ISP 存储空间被声明，IAP 存储空间范围由 IAP 和 ISP 起始地址决定如下列表：
IAP 高边界= ISP 起始地址 – 1.
IAP 低边界= ISP 起始地址 – IAP.

如果 ISP 存储空间没有被声明，IAP 存储空间范围由下列公式决定：
IAP 高边界= 0x1FFF.
IAP 低边界= 0x1FFF – IAP + 1.

例如，如果 ISP 存储空间是 1K 字节，这样 ISP 的起始地址是 **0x1C00**，并且 IAP 存储空间是 1K 字节，此时 IAP 存储空间的范围就在 0x1800 ~ 0x1BFF 。**MA801_MA802** 的 IAP 低边界由 IAPLB 寄存器决定，IAPLB 寄存器可以在用工具修改来调整 IAP 大小。

### 19.4.2. IAP-存储空间更新数据

ISP/IAP 相关的特殊功能寄存器见章节“19.5 ISP/IAP 寄存器“。

因为 IAP-存储器是 Flash 的一部分，只能整页檫除，不能字节檫除。为了刷新“一个字节”到 IAP 存储器，使用者不能直接写新的数据到那个字节。下面显示正确的步骤：

步骤 1: 保存整页的 Flash 数据(512 字节)到 XRAM 缓存器中包含需要修改的数据。
步骤 2: 檫除这个页 (使用 ISP/IAP  Flash 页檫除模式)。
步骤 3: 在 XRAM 缓存器中修改新数据。
步骤 4: 将刷新过的 XRAM 缓存器中的数据写到这页 ***(使用 ISP/IAP  Flash 页写模式)***.

使用者可以***使用 ISP/IAP  Flash 页读取模式*** 读取 IAP-存储器中的目标数据。

### 19.4.3. IAP 注意事项

IAP 期间中断

在触发 ISP/IAP flash 处理之后，内部 IAP 处理时 MCU 将停止一会儿直到处理完成。此时，如果中断已使能则中断事件将排队等待服务。一旦 ISP/IAP Flash 处理完成，MCU 继续运行并且如果中断标志仍然有效则排队中的中断将立即服务。不过用户需要意识到下列事项：

(1) 当 MCU 停止在 IAP 处理时，中断不能实时服务。
(2) 低/高电平触发外部中断 nINTx,必须保持到 IAP 处理完成，否则将被忽略。

IAP 和空闲模式

**MA801_MA802** 不使用空闲模式执行 IAP 功能。反而 ISP/IAP 引擎操作 Flash 存储空间将冻结 CPU 的运行。一旦 ISP/IAP 运行结束，CPU 将继续并且推进紧跟着 ISP/AP 激活的指令。

IAP 的访问目标

如前所述, IAP 用来编程 IAP 存储空间。一旦访问目标地址不在 IAP 存储空间之内, 硬件将自动忽略 ISP 处理的触发。这样 IAP 触发是无效的并且硬件不做任何事情。

读取 IAP 数据的另一种方法

IAP 存储空间读取 Flash 数据，除了使用 Flash 的读取模式之外，另一个方法是使用“MOVC A,@A+DPTR”指令。这里，DPTR 和 ACC 各自填入想要的地址和偏移量。并且访问目标必须在 IAP 存储空间内，否则读取的数据将不确定。注意使用‘MOVC’指令比使用 Flash 的读取模式更快。

IAP 的 Flash 持久期

内置 Flash 的持久期是 20,000 写周期，换句话说写周期不能超过 20,000 次。这样用户必须注意应用中需要频繁更新 IAP 存储空间这一点。

## 19.5. ISP/IAP 寄存器

下面的特殊功能寄存器有关于 ISP 和 IAP 的操作.:

IFD: ISP/IAP Flash 数据寄存器
SFR 地址 = 0xE2 复位值= 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 为 ISP/IAP 操作的数据寄存器，ISP/IAP 进行读写操作时，IFD 作为数据缓冲区.

IFADRH: ISP/IAP 地址高
SFR 地址 = 0xE3 复位值= 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 存放 ISP/IAP 操作的目标地址的高位.

IFADRL: ISP/IAP 地址低
SFR 地址 = 0xE4 复位值= 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 存放 ISP/IAP 操作的目标地址的低位.

IFMT: ISP/IAP Flash 模式表
SFR Page = 普通
SFR 地址 = 0xE5 复位值= xxxx-xx00

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

Bit 7~5: 保留. 写 IFMT 时这些位必须写”0”.

Bit 3~0: ISP/IAP/Page-P 操作模式选择

| MS[4:0] | Mode |
|---|---|
| 0 0 0 0 0 | 空闲状态 |
| 0 0 0 0 1 | 读 AP/IAP-存储器中的 Flash |
| 0 0 0 1 0 | 写 AP/IAP-存储器中的 Flash |
| 0 0 0 1 1 | 檫 AP/IAP-存储器中的 Flash |
| Others | Reserved |

IFMT 用于选则 ISP/IAP 操作功能

SCMD: 顺序命令寄存器
SFR 地址 = 0xE6 复位值= 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 |

ISP/IAP/IAPLB 的操作都需要用 SCMD 寄存器来触发，当 ISPCR.7 为“1”且 SCMD 顺序写入命令“0x46 0xB9”时，ISP 操作被触发。

ISPCR: ISP 控制寄存器

SFR 地址 = 0xE7 复位值= 0000-0xxx

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

Bit 7: ISPEN, ISP/IAP 使能位
0: 全局禁用 ISP/IAP 编程/擦除/读功能
1: 使能 ISP/IAP 编程/擦除/读功能

Bit 6: SWBS, 软件引导选择位
0: 复位后从 AP 区域启动
1: 复位后从 ISP 区域启动

Bit 5: SWRST, 软件复位触发控制位
0: 无操作
1: 产生软件复位，启动后硬件自动清除它

Bit 4: CFAIL, ISP/IAP 命令是否执行失败
0: ISP/IAP 操作成功。
1: ISP/IAP 操作失败。是由于禁止写 Flash 存储造成。

Bit 3: 保留。当写 ISPCR 时软件必须写 “0”在这位。

Bit 2~0: WAIT.2~0, ISP 忙等待时间表等待周期设置。

| WAIT[2:0] | 震荡频率 (MHz) |
|---|---|
| 0 0 0 | > 24 |
| 0 0 1 | 20 ~ 24 |
| 0 1 0 | 12 ~ 20 |
| 0 1 1 | 6 ~ 12 |
| 1 0 0 | 3 ~ 6 |
| 1 0 1 | 2 ~ 3 |
| 1 1 0 | 1 ~ 2 |
| 1 1 1 | < 1 |

## 19.6. ISP/IAP 示例代码

(1). 功能需求: ISP/IAP Flash 读的子程序

汇编语言代码范例:

```
IxP_Flash_Read      EQU             01h
ISPEN               EQU             80h

_ixp_read:
ixp_read:

   MOV    ISPCR,#ISPEN             ; 功能使能
   MOV    IFMT,# IxP_Flash_Read    ; ixp_read=0x01

   MOV    IFADRH,??                ; 填写 [IFADRH,IFADRL] 字节地址
   MOV    IFADRL,??

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

MOV    A,IFD                       ; 现在读出来的数据存在于 IFD 中

   MOV    IFMT,#000h               ; Flash_Standby=0x00
   ANL    ISPCR,#(0FFh – ISPEN)    ; 禁止功能

   RET
```

C 语言代码范例:

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


unsigned char ixp_read (void)
{
   unsigned char arg;
ISPCR = ISPEN;                  //功能使能
   IFMT = IxP_Flash_Read;       // IxP_Read=0x01

   IFADRH = ??
   IFADRL = ??

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

   arg = IFD;

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

   return arg;
}
```

(2). 功能需求: ISP/IAP Flash 檫除的子程序

汇编语言代码范例:

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

_ixp_erase:
ixp_erase:

   MOV    ISPCR,#ISPEN              ; 功能使能
   MOV    IFMT,# IxP_Flash_Erase        ; ixp_erase=0x03

   MOV    IFADRH,??                 ; 填写 [IFADRH,IFADRL] 字节地址
   MOV    IFADRL,??

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

   MOV    IFMT,#000h                ; Flash_Standby=0x00
   ANL    ISPCR,#(0FFh – ISPEN)         ; 禁止功能

   RET
```

C 语言代码范例:

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


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

   IFADRH = Addr_H;
   IFADRL = Addr_L;

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

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

(3). 功能需求: ISP/IAP Flash 写的子程序

汇编语言代码范例:

```
IxP_Flash_Program       EQU                 02h
ISPEN                   EQU                 80h

_ixp_program:
ixp_program:

   MOV     ISPCR,#ISPEN              ; 功能使能
   MOV     IFMT,# IxP_Flash_Program      ; ixp_program=0x03

   MOV     IFADRH,??                 ; 填写[IFADRH,IFADRL] 字节地址
   MOV     IFADRL,??
MOV     IFD, A                    ; 现在,要写的数据存在于 A 累加器中

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

   MOV     IFMT,#000h                ; Flash_Standby=0x00
   ANL     ISPCR,#(0FFh – ISPEN)         ; 禁止功能

   RET
```

C 语言代码范例:

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

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

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

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

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

}
```

# 20. 辅助特殊功能寄存器

AUXR: 辅助寄存器

SFR 地址 = 0x8E 复位值= 0000-00xx

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

Bit 7: T0X12, 当 C/T=0 时，定时器 0 的时钟源选择。
0: 选择 SYSCLK/12 作时钟源
1: 选择 SYSCLK 作时钟源

Bit 6: T1X12, 当 C/T=0 时，定时器 1 的时钟源选择。
0: 选择 SYSCLK/12 作时钟源
1: 选择 SYSCLK 作时钟源

Bit 5: URM0X6, 串口模式 0 波特率选择
0: 选择 SYSCLK/12 作 UART 模式 0 波特率
1: 选择 SYSCLK/2 作 UART 模式 0 波特率

Bit 4: EADCI, ADC 中断使能
0: 禁止 ADC 中断使能
1: 使能 ADC 中断使能

Bit 3: ESPI, SPI 中断使能
0: 禁止 SPI 中断使能
1: 使能 SPI 中断使能

Bit 2: ENLVFI, 低电平检测中断使能
0: 禁止低电平检测中断
1: 使能低电平检测中断

Bit 1~0:保留.写 AUXR 时,此位必须写“0”.

# 21. 硬件选项

MCU 的硬件选项定义这个设备的行为,它不能被软件改写或控制.硬件选项只能被普通编程器改写或者"笙泉的 8051 烧写器 U1"改写,在整片檫除之后,所有的硬件选项在失效状态,并且没有 ISP 存储器和 IAP 存储器配置,
MA801_MA802 有以下硬件选项:

**LOCK:**
☑:使能：当用通用编程器读取内部数据时，读出的数据将全部是FF .
☐:禁用：不锁数据

**SB:**
☑:使能：当用通用编程器读取内部数据时，读出的数据将被打乱.
☐:禁用：数据不被打乱

**ISP**-存储器空间:

ISP 存储器空间用说明它的开始地址.而且,它的高地址被限制在 Flash 空间的末尾地址,例如: 0x1FFF. 下面的表列出 ISP 的空间选项. 缺省设置 I, **MA801_MA802** ISP 空间设置为 **1K** 并且已经植入了有笙泉知识产权的 I-线的 ISP 引导码

| ISP-存储容量 | ISP 开始地址 |
|---|---|
| 3K bytes | 0x1400 |
| 2K bytes | 0x1800 |
| 1K bytes | 0x1C00 |
| 没有 ISP 空间 | -- |

**HWBS:**
☑: 使能:上电复位时，如果ISP空间被配置，则MCU从ISP入口启动.
☐: 禁用: MCU 总是从 AP 入口启动.

**IAP**-存储器空间:
IAP-存储器空间指定用户定义 IAP 空间。 IAP-存储器空间被硬件选项配置。缺省值为 1K 字节。

**ENLVR:**
☑: 使能:在应用程序开始地址低压复位有效, (对于3.3V芯片是2.3V, 对于5.0V芯片是3.7V).
☐: 禁用:禁止低压复位

**LVFWP:**
☑:使能:低压FLASH写保护. 低压时IAP或ISP对FLASH写无效. (对于3.3V芯片是2.3V, 对于5.0V芯片是3.7V)
☐: 禁用:禁止低压写保护.

**OSCDN:**
☑: 使能: 当主震荡小于12MHz时，减少振荡器增益以降低功耗和EMI
☐: 普通增益（当主震荡大于25MHz时）

**ENROSC:**
☑: 使能： 单片机使用内部6MHz RC振荡
☐: 禁用： 单片机使用外部振荡

**HWENW:** 硬件加载 WDTCR 中的"ENW".
☑: 使能:上电后使能 WDT 并加载 HWWIDL 和 HWPS2~0 的内容到 WDTCR.
☐: 禁止: 上电时对看门狗无动作.

**HWWIDL, HWPS2, HWPS1, HWPS0:**

当 HWENW 使能, 上电后相应的这 4 个熔丝位会加载 WDTCR SFR 的值.

**WDSFWP:**

☑: 使能. WDT SFRs, WDTCR 寄存器中的 WIDL, PS2, PS1 和 PS0 被写保护.

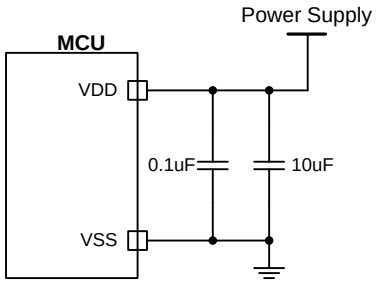
☐: 禁止. WDT SFRs, WDTCR 寄存器中的 WIDL, PS2, PS1 和 PS0, 可以由软件写.

# 22. 应用说明

## 22.1. 电源电路

**MA801_MA802** 工作电压范围, MA801 工作在 4.5V 到 5.5V, MA802 工作在 2.4V 到 3.6V. 必须加外部的去藕和旁路电容.如 表 22–1.

表 22–1. 电源电路



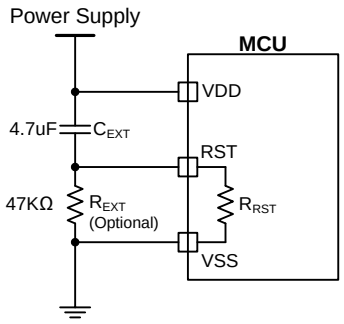

## 22.2. 复位电路

正常情况, 复位信号在上电复位时都能安全的产生. 但是,为了进一步确保上电时 MCU 更可靠的复位, 外部的复位电路是必须的. 图 22–2 显示外部的复位电路, 连接一个电容 $C_{EXT}$ 到 VDD (供电电源)和一个电阻到 $R_{EXT}$ VSS (地).

通常, $R_{EXT}$ 是可以选择的,因为 RST 口有一个内部的下拉电阻($R_{RST}$). 这内部的集成电阻到 VSS 允许上电复位仅仅用一个外部的电容到 $C_{EXT}$ 到 VDD.

参考章节“23.2 直流电气特性” $R_{RST}$ 值.

图 22–2. 复位电路



## 22.3. XTAL 振荡电路

为了能成功起振 (最大到 25MHz), 电容 C1 和 C2 是必须的，如图 22–3 所示。 通常， C1 和 C2 使用相同的值。 . 表 22–1 列举了 C1 & C2 在不同晶振下的值。

图 22–3. XTAL 振荡电路

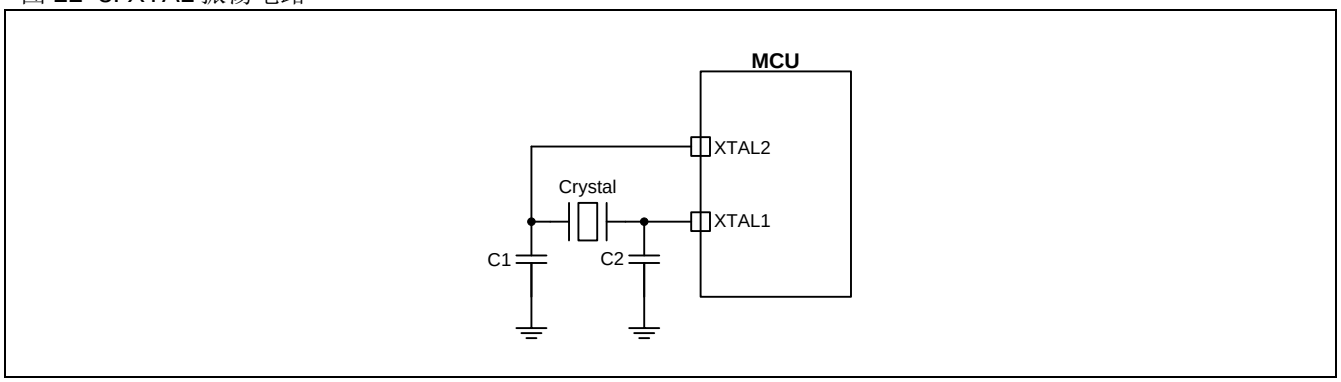


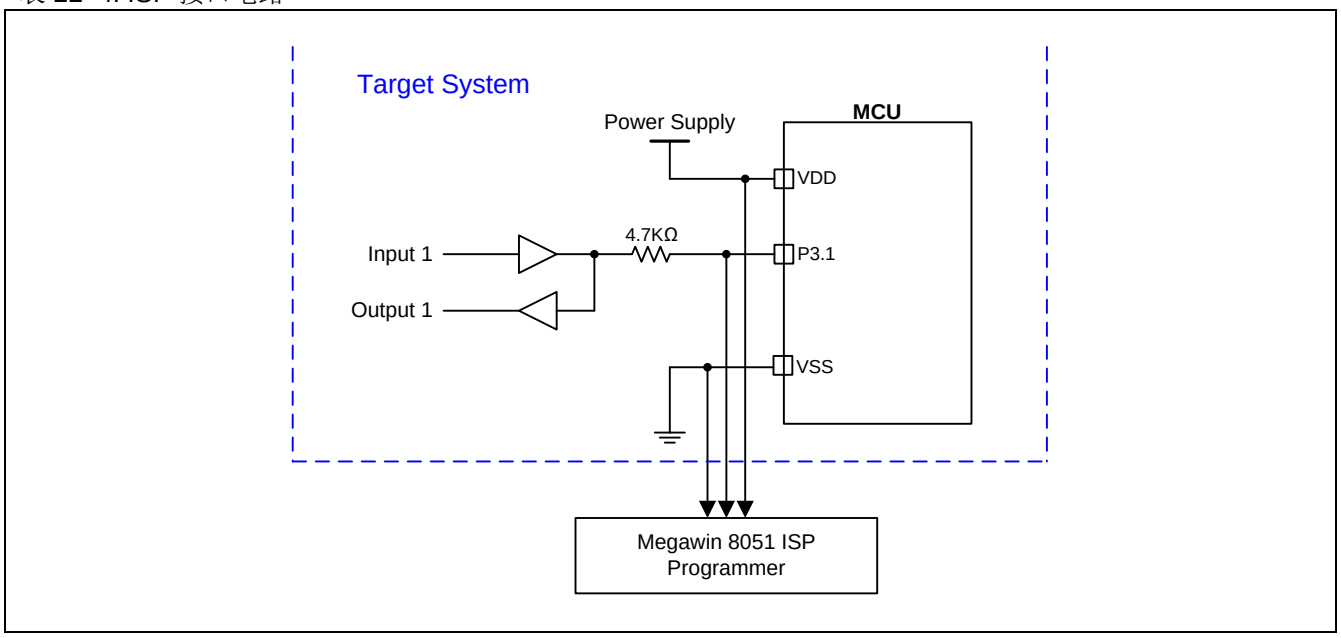

表 22–1. 振荡电路的电容 C1&C2 参照表

| 晶振 | C1, C2 电容值 |
|---|---|
| 16MHz ~ 25MHz | 10pF |
| 6MHz ~ 16MHz | 15pF |
| 2MHz ~ 6MHz | 33pF |

## 22.4. ISP 接口电路

**MA801_MA802** 在出厂前已经植入笙泉标准的 ISP 引导码(具有笙泉知识产权的 1 线 ISP 协议). 这个 ISP 接口允许 P3.1 口与使用者功能共享以便实现在系统写 Flash 的功能. 当程序在 ISP 存储器中运行 ISP 的通讯功能被执行,这时 AP 应用程序失效. 在这个状态,I 笙泉的 ISP 底层软件能安全的借用 P3.1 口. 大部分的应用, 使用者应用的外部电阻必须被隔绝与 ISP 接口电路的连接. 典型的隔绝连接结构如表 22–4.

表 22–4. ISP 接口电路



# 23. 电气特性

## 23.1. 最大绝对额定参数

MA801:

| 参数 | 额定值 | 单位 |
|---|---|---|
| 环境温度偏差 | -40 ~ +85 | °C |
| 存储温度 | -65 ~ + 150 | °C |
| IO 口和复位脚的对地电压 | -0.5 ~ VDD + 0.5 | V |
| VDD 脚的对地电压 | -0.5 ~ +6.0 | V |
| 芯片总电流 | 400 | mA |
| IO 口的最大吸收电流 | 40 | mA |

*注意：实际参数超过上述各项“绝对最大额定值”可能会对设备造成永久性损坏。这些参数是一个设备进行正常功能操作的最大额定值，任何超过上述各项的条件都不被建议，否则可能会影响设备运行的稳定性。

MA802:

| 参数 | 额定值 | 单位 |
|---|---|---|
| 环境温度偏差 | -40 ~ +85 | °C |
| 存储温度 | -65 ~ + 150 | °C |
| IO 口和复位脚的对地电压 | -0.3 ~ VDD + 0.3 | V |
| VDD 脚的对地电压 | -0.3 ~ +4.2 | V |
| 芯片总电流 | 400 | mA |
| IO 口的最大吸收电流 | 40 | mA |

*注意：实际参数超过上述各项“绝对最大额定值”可能会对设备造成永久性损坏。这些参数是一个设备进行正常功能操作的最大额定值，任何超过上述各项的条件都不被建议，否则可能会影响设备运行的稳定性。

## 23.2. 直流电气特性

MA801:

VDD = 5.0V±10%, VSS = 0V, $T_A$ = 25 ℃ 并且每条指令都执行 NOP 指令, 除非另行说明

| 标志 | 参数 | 测试条件 | 范围 | | | 单位 |
|---|---|---|---|---|---|---|
| | | | **最小** | 典型 | 最大 | |
| 输入/输出特性 | | | | | | |
| $V_{IH1}$ | 输入高电压 (All I/O Ports) | P0, P1, P2 和 P3 | 2.0 | | | V |
| $V_{IH2}$ | 输入高电压 (RST) | | 3.5 | | | |
| $V_{IL1}$ | 输入低电压 (所有 I/O 口) | P0, P1, P2 和 P3 | | | 0.8 | V |
| $V_{IL2}$ | 输入低电压(RST) | | | | 0.8 | V |
| $I_{IH}$ | 输入高的漏电流 (所有的仅输入口或漏极开路口) | $V_{PIN}$ = VDD | | 0 | 10 | uA |
| $I_{IL1}$ | 逻辑 0 输入电流 (准双向模式) | $V_{PIN}$ = 0.45V | | 17 | 50 | uA |
| $I_{IL2}$ | 逻辑 0 输入电流(所有的仅输入口或漏极开路口) | $V_{PIN}$ = 0.45V | | 0 | 10 | uA |
| $I_{H2L}$ | 逻辑 1 到 0 输入过渡电流 (准双向模式) | $V_{PIN}$ =1.8V | | 230 | 500 | uA |
| $I_{OH1}$ | 输出高电流 (准双向模式) | $V_{PIN}$ =2.4V | | 220 | | uA |
| $I_{OH2}$ | 输出高电流(所有的推挽输出口) | $V_{PIN}$ =2.4V | 12 | 20 | | mA |
| $I_{OL1}$ | 输出低电流 (所有 I/O 口) | $V_{PIN}$ =0.45V | 12 | 20 | | mA |
| $R_{RST}$ | 复位脚上的内部下拉电阻 | | | 100 | | Kohm |
| 能量功耗 | | | | | | |
| $I_{OP1}$ | 普通模式工作电流 | 系统时钟 = 12MHz @ 晶振 | | 12 | 30 | mA |
| $I_{IDLE1}$ | 空闲模式工作电流 | 系统时钟= 12MHz @晶振 | | 6 | 15 | mA |
| $I_{PD1}$ | 掉电模式电流 | | | 0.1 | 50 | uA |
| LVD 特征 | | | | | | |
| $V_{LVD}$ | LVD 检测电压 | 系统时钟= 12MHz @晶振 | | 3.7(1) | | V |
| 工作条件 | | | | | | |
| $V_{PSR}$ | 上电斜率 | $T_A$ = -40℃ to +85℃ | 0.05 | | | V/ms |
| $V_{OP1}$ | 工作速度 0–25MHz | $T_A$ = -40℃ to +85℃ | 4.5 | | 5.5 | V |
| $V_{OP2}$ | 工作速度 0-12MHz | $T_A$ = -40℃ to +85℃ | 4.2 | | 5.5 | V |

(1) 数据基于特性结果,非产品测试结果.

MA802:

VDD = 3.3V±10%, VSS = 0V, $T_A$ = 25 ℃并且每条指令都执行 NOP 指令, 除非另行说明

| 标志 | 参数 | 测试条件 | 范围 | | | 单位 |
|---|---|---|---|---|---|---|
| | | | 最小 | 典型 | 最大 | |
| 输入/输出特性 | | | | | | |
| $V_{IH1}$ | 输入高电压 (All I/O Ports) | P0, P1, P2 和 P3 | 2.0 | | | V |
| $V_{IH3}$ | 输入高电压 (RST) | | 2.8 | | | V |
| $V_{IL1}$ | 输入低电压 (所有 I/O 口) | P0, P1, P2 和 P3 | | | 0.8 | V |
| $V_{IL3}$ | 输入低电压(RST) | | | | 0.8 | V |
| $I_{IH}$ | 输入高的漏电流 (所有的仅输入口或漏极开路口) | $V_{PIN}$ = VDD | | 0 | 10 | uA |
| $I_{IL1}$ | 逻辑 0 输入电流 (准双向模式) | $V_{PIN}$ = 0.45V | | 7 | 50 | uA |
| $I_{IL2}$ | 逻辑 0 输入电流(所有的仅输入口或漏极开路口) | $V_{PIN}$ = 0.45V | | 0 | 10 | uA |
| $I_{H2L}$ | 逻辑 1 到 0 输入过渡电流 (准双向模式) | $V_{PIN}$ =1.4V | | 100 | 600 | uA |
| $I_{OH1}$ | 输出高电流 (准双向模式) | $V_{PIN}$ =2.4V | | 64 | | uA |
| $I_{OH2}$ | 输出高电流(所有的推挽输出口) | $V_{PIN}$ =2.4V | 4 | 8 | | mA |
| $I_{OL1}$ | 输出低电流 (所有 I/O 口) | $V_{PIN}$ =0.45V | 8 | 14 | | mA |
| $R_{RST}$ | 复位脚上的内部下拉电阻 | | | 100 | | Kohm |
| 能量功耗 | | | | | | |
| $I_{OP1}$ | 普通模式工作电流 | 系统时钟 = 12MHz @晶振 | | 9 | 15 | mA |
| $I_{IDLE1}$ | 空闲模式工作电流 | 系统时钟= 12MHz @晶振 | | 3.5 | 6 | mA |
| $I_{PD1}$ | 掉电模式电流 | | | 0.1 | 50 | uA |
| LVD 特征 | | | | | | |
| $V_{LVD}$ | LVD 检测电压 | 系统时钟= 12MHz @晶振 | | 2.3(1) | | V |
| 工作条件 | | | | | | |
| $V_{PSR}$ | 上电斜率 | $T_A$ = -40℃ to +85℃ | 0.05 | | | V/ms |
| $V_{OP1}$ | 工作速度 0–25MHz | $T_A$ = -40℃ to +85℃ | 2.7 | | 3.6 | V |
| $V_{OP2}$ | 工作速度 0-12MHz | $T_A$ = -40℃ to +85℃ | 2.4 | | 3.6 | V |

(1) 数据基于特性结果,非产品测试结果.

## 23.3. 外部时钟特征

MA801:

VDD = 4.5V ~ 5.5V, VSS = 0V, $T_A$ = -40℃ to +85℃, 除非另有说明

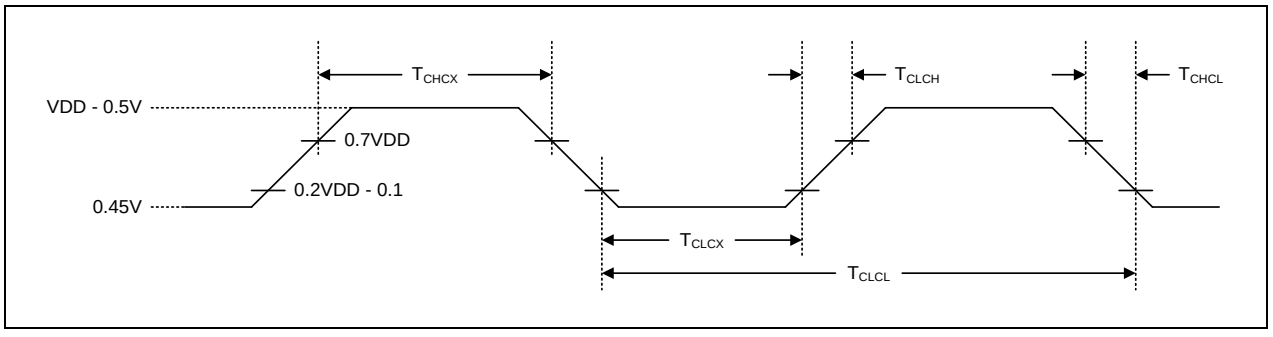
| 标志 | 参数 | 振荡器 震荡模式 | | 单位 |
|---|---|---|---|---|
| | | 最小. | 最大 | |
| $1/t_{CLCL}$ | 震荡频率 | 2 | 25 | MHz |
| $1/t_{CLCL}$ | 震荡频率 (VDD = 4.2V ~ 5.5V) | 2 | 12 | MHz |
| $t_{CLCL}$ | 时钟周期 | 40 | | ns |
| $t_{CHCX}$ | 高时间 | 0.4T | 0.6T | $t_{CLCL}$ |
| $t_{CLCX}$ | 低时间 | 0.4T | 0.6T | $t_{CLCL}$ |
| $t_{CLCH}$ | 上升时间 | | 5 | ns |
| $t_{CHCL}$ | 下降时间 | | 5 | ns |

For MA802:

VDD = 2.7V ~ 3.6V, VSS = 0V, $T_A$ = -40℃ to +85℃, 除非另有说明

| 标志 | 参数 | 振荡器 震荡模式 | | 单位 |
|---|---|---|---|---|
| | | 最小. | 最大 | |
| $1/t_{CLCL}$ | 震荡频率 | 2 | 25 | MHz |
| $1/t_{CLCL}$ | 震荡频率 (VDD = 2.4V ~ 3.6V) | 2 | 12 | MHz |
| $t_{CLCL}$ | 时钟周期 | 40 | | ns |
| $t_{CHCX}$ | 高时间 | 0.4T | 0.6T | $t_{CLCL}$ |
| $t_{CLCX}$ | 低时间 | 0.4T | 0.6T | $t_{CLCL}$ |
| $t_{CLCH}$ | 上升时间 | | 5 | ns |
| $t_{CHCL}$ | 下降时间 | | 5 | ns |

表 23–1. 外部时钟驱动波形



## 23.4. IRCO 特征

MA801:

| 参数 | 测试条件 | 范围 | | | 单位 |
|---|---|---|---|---|---|
| | | 最小 | 典型 | 最大 | |
| 电源电压 | | 4.5 | | 5.5 | V |
| IRCO 频率 | TA = +25℃ | 4.2(1) | 6(1) | 7.8(1) | MHz |

(1)数据基于特性结果,非产品测试结果.

MA802:

| 参数 | 测试条件 | 范围 | | | 单位 |
|---|---|---|---|---|---|
| | | 最小 | 典型 | 最大 | |
| 电源电压 | | 2.7 | | 3.6 | V |
| IRCO 频率 | TA = +25℃ | 4.2(1) | 6(1) | 7.8(1) | MHz |

(1) 数据基于特性结果,非产品测试结果.

## 23.5. Flash 特征

MA801:

| 参数 | 测试条件 | 范围 | | | 单位 |
|---|---|---|---|---|---|
| | | 最小 | 典型 | 最大 | |
| 电源电压 | TA = -40℃ to +85℃ | 4.2 | | 5.5 | V |
| Flash 写(檫/写)电压 | TA = -40℃ to +85℃ | 4.5 | | 5.5 | V |
| Flash 檫/写周期 | TA = -40℃ to +85℃ | 20,000 | | | 次数 |
| Flash 数据保存 | TA = +25℃ | 100 | | | 年度 |

MA802:

| 参数 | 测试条件 | 范围 | | | 单位 |
|---|---|---|---|---|---|
| | | 最小 | 典型 | 最大 | |
| 电源电压 | TA = -40℃ to +85℃ | 2.4 | | 3.6 | V |
| Flash 写(檫/写)电压 | TA = -40℃ to +85℃ | 2.7 | | 3.6 | V |
| Flash 檫/写周期 | TA = -40℃ to +85℃ | 20,000 | | | 次数 |
| Flash 数据保存 | TA = +25℃ | 100 | | | 年度 |

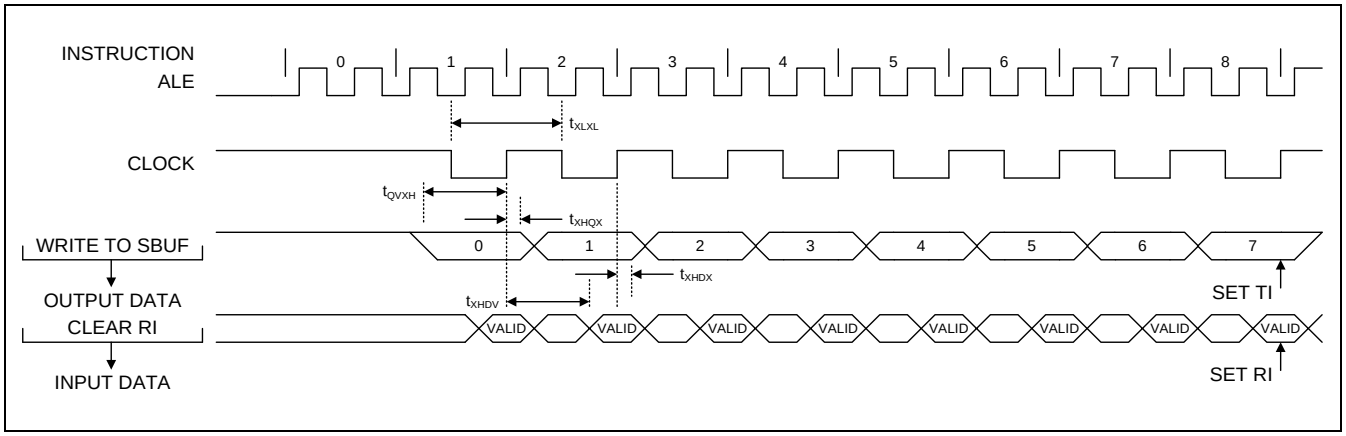
## 23.6. 串口时间特征

**MA801:** VDD = 5.0V±10%, VSS = 0V, $T_A$ = -40℃ to +85℃, 除非另有说明

**MA802:** VDD = 3.3V±10%, VSS = 0V, $T_A$ = -40℃ to +85℃, 除非另有说明

| 标志 | 参数 | URM0X3 = 0 | | URM0X3 = 1 | | 单位 |
|---|---|---|---|---|---|---|
| | | **最小** | 最大 | **最小** | 最大 | |
| $t_{XLXL}$ | 串口时钟周期 | 12T | | 4T | | $T_{SYSCLK}$ |
| $t_{QVXH}$ | 输出数据设置到时钟上升沿时间 | 10T-20 | | T-20 | | ns |
| $t_{XHQX}$ | 输出数据在时钟上升沿之后保持时间 | T-10 | | T-10 | | ns |
| $t_{XHDX}$ | 输入数据在时钟上升沿之后保持时间 | 0 | | 0 | | ns |
| $t_{XHDV}$ | 时钟上升沿到输入数据有效时间 | | 10T-20 | | 4T-20 | ns |

表 23–2. 移位寄存器模式时间波形



## 23.7. SPI 时间特性

**For MA801:** VDD = 5.0V±10%, VSS = 0V, $T_A$ = -40℃ to +85℃, 除非另有说明

**For MA802:** VDD = 3.3V±10%, VSS = 0V, $T_A$ = -40℃ to +85℃, 除非另有说明

| 标志 | 参数 | 最小 | 最大 | 单位 |
|---|---|---|---|---|
| 主机模式时序 | | | | |
| $t_{MCKH}$ | SPICLK 高时间 | 2T | | $T_{SYSCLK}$ |
| $t_{MCKL}$ | SPICLK 低时间 | 2T | | $T_{SYSCLK}$ |
| $t_{MIS}$ | MISO Valid to SPICLK Shift Edge | 2T+20 | | ns |
| $t_{MIH}$ | SPICLK Shift Edge to MISO Change | 0 | | ns |
| $t_{MOH}$ | SPICLK Shift Edge to MOSI Change | | 10 | ns |
| 从机模式时序 | | | | |
| $t_{SE}$ | nSS Falling to First SPICLK Edge | 2T | | $T_{SYSCLK}$ |
| $t_{SD}$ | Last SPICLK Edge to nSS Rising | 2T | | $T_{SYSCLK}$ |
| $t_{SEZ}$ | nSS Falling to MISO Valid | | 4T | $T_{SYSCLK}$ |
| $t_{SDZ}$ | nSS Rising to MISO High-Z | | 4T | $T_{SYSCLK}$ |
| $t_{CKH}$ | SPICLK 高时间 | 4T | | $T_{SYSCLK}$ |
| $t_{CKL}$ | SPICLK 低时间 | 4T | | $T_{SYSCLK}$ |
| $t_{SIS}$ | MOSI Valid to SPICLK Sample Edge | 2T | | $T_{SYSCLK}$ |
| $t_{SIH}$ | SPICLK Sample Edge to MOSI Change | 2T | | $T_{SYSCLK}$ |
| $t_{SOH}$ | SPICLK Shift Edge to MISO Change | | 4T | $T_{SYSCLK}$ |
| $t_{SLH}$ | Last SPICLK Edge to MISO Change (CPHA = 1 ONLY) | 1T | 2T | $T_{SYSCLK}$ |

表 23–3. SPI 主机传送时间波形 CPHA=0

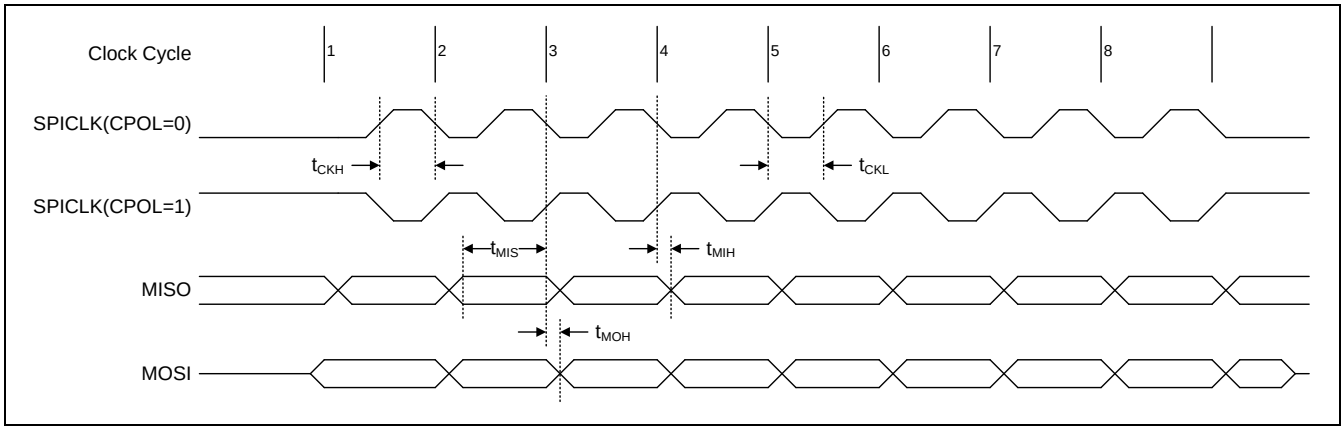



表 23–4. SPI 主机传送时间波形 CPHA=1

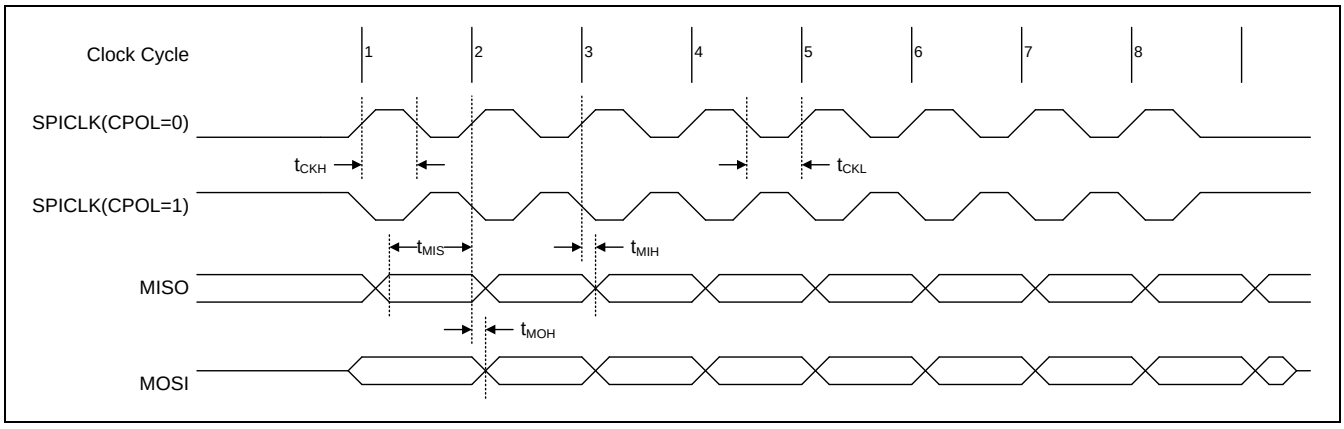


表 23–5. SPI 从机传送时间波形 CPHA=0

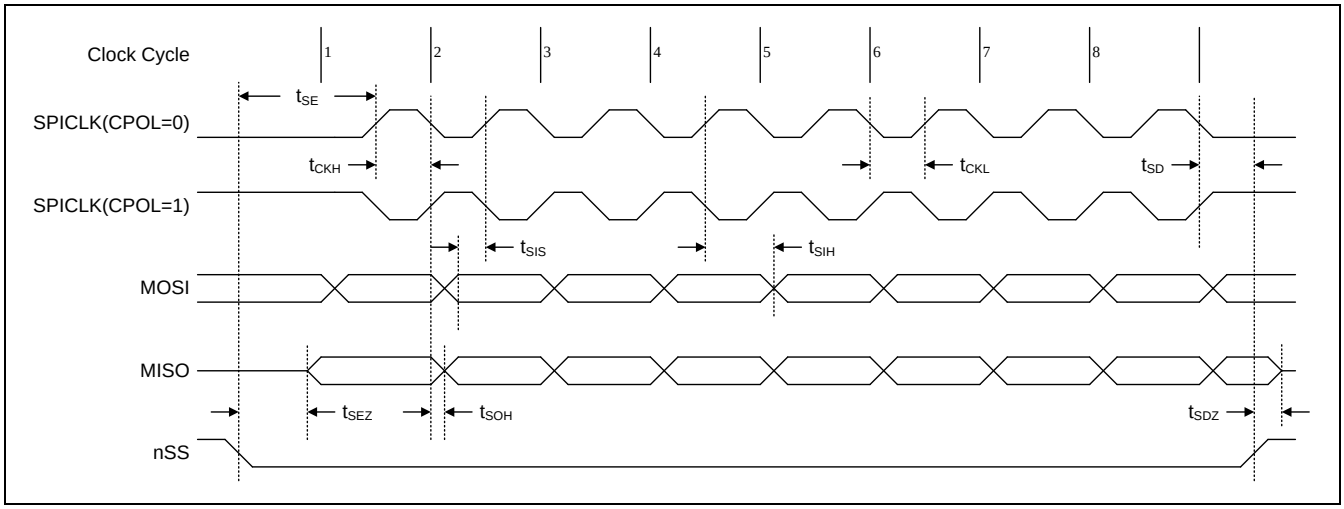
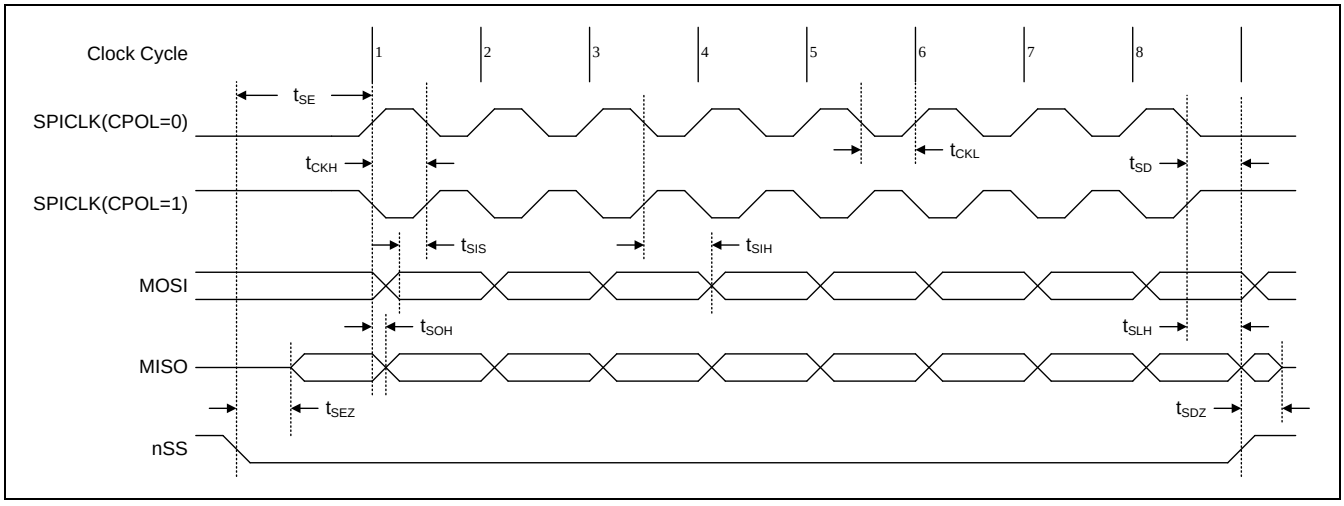



表 23–6. SPI 从机传送时间波形 CPHA=1



# 24. 指令集

表 24–1. 指令集

| 数据传送 | | | |
|---|---|---|---|
| 助记符 | 描述 | 字节数 | 指令周期 |
| MOV A, Rn | Acc ← Rn | 1 | 1 |
| MOV A, direct | Acc ← direct | 2 | 2 |
| MOV A, @Ri | Acc ← Ri | 1 | 2 |
| MOV A, #data | Acc ← data | 2 | 2 |
| MOV Rn,A | Rn ← Acc | 1 | 2 |
| MOV Rn,direct | Rn ← direct | 2 | 4 |
| MOV Rn,#data | Rn ← data | 2 | 2 |
| MOV direct,A | direct ← Acc | 2 | 3 |
| MOV direct,Rn | direct ← Rn | 2 | 3 |
| MOV direct,direct | direct ← direct | 3 | 4 |
| MOV direct,@Ri | direct ← Ri | 2 | 4 |
| MOV direct,#data | direct ← data | 3 | 3 |
| MOV @Ri,A | Ri ← Acc | 1 | 3 |
| MOV @Ri,direct | Ri ← direct | 2 | 3 |
| MOV @Ri,#data | Ri ← data | 2 | 3 |
| MOV DPTR,#data16 | DPTR ← 16bit data | 3 | 3 |
| MOVC A,@A+DPTR | Acc ← (A+DPTR)地址所指的数据 | 1 | 4 |
| MOVC A,@A+PC | Acc ← (A+PC)地址所指的数据 | 1 | 4 |
| PUSH direct | 堆栈 ← direct | 2 | 4 |
| POP direct | direct ← 堆栈 | 2 | 3 |
| XCH A,Rn | A 和 Rn 互换 | 1 | 3 |
| XCH A,direct | A 和 direct 互换 | 2 | 4 |
| XCH A,@Ri | A 和 Ri 互换 | 1 | 4 |
| XCHD A,@Ri | A 和 Ri 的低四互换 | 1 | 4 |
| 算术运算 | | | |
| ADD A,Rn | Acc ← Acc+Rn | 1 | 2 |
| ADD A,direct | Acc ← Acc+direct | 2 | 3 |
| ADD A,@Ri | Acc ← Acc+Ri | 1 | 3 |
| ADD A,#data | Acc ← Acc+data | 2 | 2 |
| ADDC A,Rn | Acc ← Acc+Rn+C | 1 | 2 |
| ADDC A,direct | Acc ← Acc+direct+C | 2 | 3 |
| ADDC A,@Ri | Acc ← Acc+Ri+C | 1 | 3 |
| ADDC A,#data | Acc ← Acc+data+C | 2 | 2 |
| SUBB A,Rn | Acc ← Acc-Rn-C | 1 | 2 |
| SUBB A,direct | Acc ← Acc-direct-C | 2 | 3 |
| SUBB A,@Ri | Acc ← Acc-Ri-C | 1 | 3 |
| SUBB A,#data | Acc ← Acc-data-C | 2 | 2 |
| INC A | Acc ← Acc +1 | 1 | 2 |
| INC Rn | Rn ← Rn +1 | 1 | 3 |
| INC direct | direct ← direct +1 | 2 | 4 |
| INC @Ri | Ri ← Ri +1 | 1 | 4 |
| INC DPTR | DPTR ← DPTR +1 | 1 | 1 |
| DEC A | Acc ← Acc - 1 | 1 | 2 |
| DEC Rn | Rn ← Rn -1 | 1 | 3 |
| DEC direct | direct ← direct -1 | 2 | 4 |
| DEC @Ri | Ri ← Ri -1 | 1 | 4 |
| MUL AB | 两数相乘，结果高八位存入 B,低八位存入 A | 1 | 4 |
| DIV AB | Acc 除以 B, 商存入 Acc,余数存入 B | 1 | 5 |

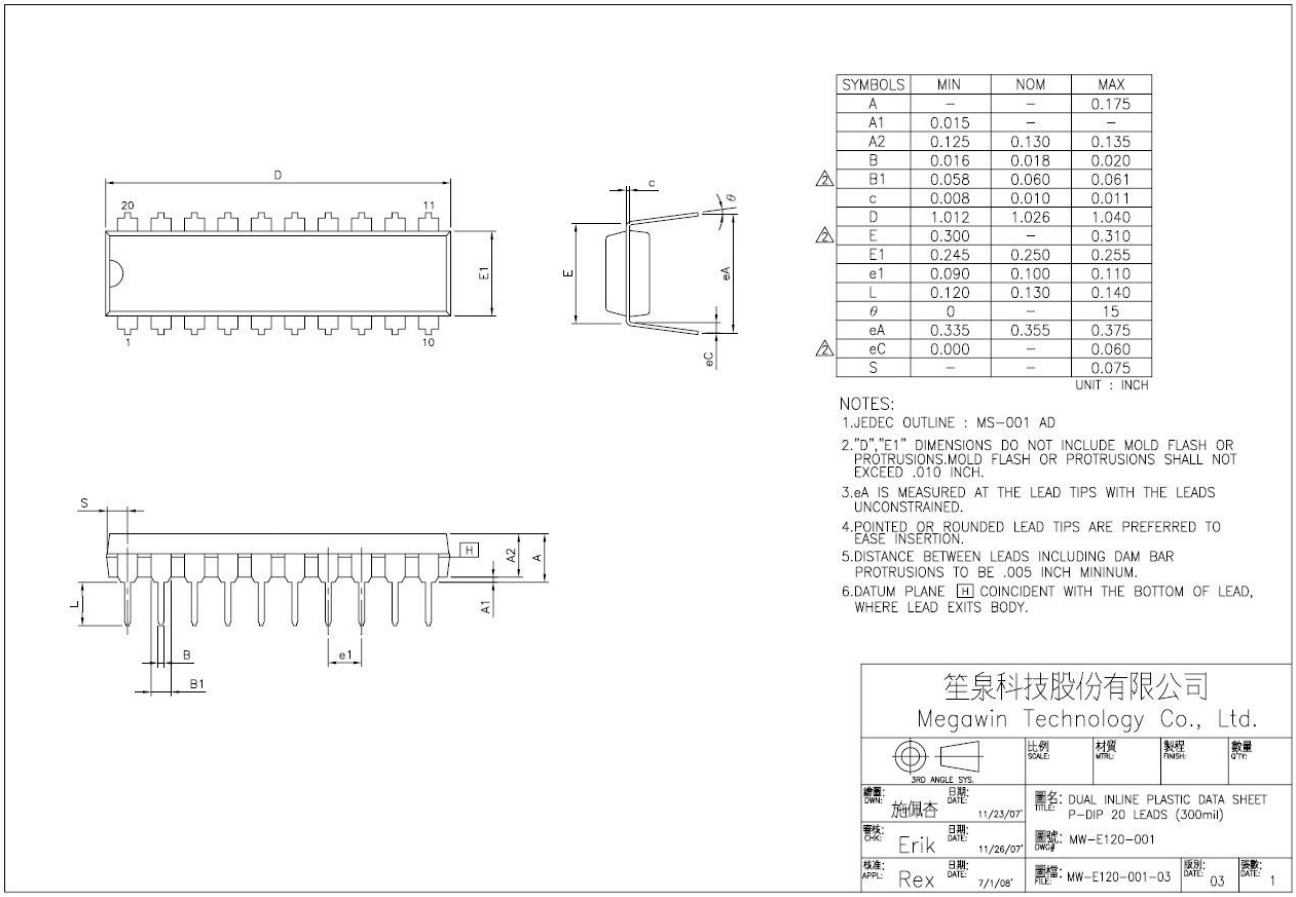
| | | | |
|---|---|---|---|
| DA A | Acc 作十进制调整 | 1 | 4 |
| 逻辑运算 | | | |
| ANL A,Rn | Acc ← Acc and Rn | 1 | 2 |
| ANL A,direct | Acc ← Acc and direct | 2 | 3 |
| ANL A,@Ri | Acc ← Acc and Ri | 1 | 3 |
| ANL A,#data | Acc ← Acc and data | 2 | 2 |
| ANL direct,A | Direct ← direct and Acc | 2 | 4 |
| ANL direct,#data | Direct ← direct and data | 3 | 4 |
| ORL A,Rn | Acc ← Acc or Rn | 1 | 2 |
| ORL A,direct | Acc ← Acc or direct | 2 | 3 |
| ORL A,@Ri | Acc ← Acc or Ri | 1 | 3 |
| ORL A,#data | Acc ← Acc or data | 2 | 2 |
| ORL direct,A | Direct ← direct or Acc | 2 | 4 |
| ORL direct,#data | Direct ← direct or data | 3 | 4 |
| XRL A,Rn | Acc ← Acc xor Rn | 1 | 2 |
| XRL A,direct | Acc ← Acc xor direct | 2 | 3 |
| XRL A,@Ri | Acc ← Acc xor Ri | 1 | 3 |
| XRL A,#data | Acc ← Acc xor data | 2 | 2 |
| XRL direct,A | Direct ← direct xor Acc | 2 | 4 |
| XRL direct,#data | Direct ← direct xor data | 3 | 4 |
| CLR A | 清除累加器 Acc | 1 | 1 |
| CPL A | 累加器反相 | 1 | 2 |
| RL A | 累加器向左旋转 | 1 | 1 |
| RLC A | 累加器和 C 左旋 | 1 | 1 |
| RR A | 累加器向右旋转 | 1 | 1 |
| RRC A | 累加器和 C 右旋 | 1 | 1 |
| SWAP A | 累加器的高低四位互换 | 1 | 1 |
| 位逻辑运算 | | | |
| CLR C | 清除进位标记 | 1 | 1 |
| CLR bit | 清除直接位元 | 2 | 4 |
| SETB C | 设定进位标记 | 1 | 1 |
| SETB bit | 设定直接位元 | 2 | 4 |
| CPL C | 进位标记取反 | 1 | 1 |
| CPL bit | 直接位元取反 | 2 | 4 |
| ANL C,bit | C ← C and bit | 2 | 3 |
| ANL C,/bit | C ← C and bit(反相) | 2 | 3 |
| ORL C,bit | C ← C or bit | 2 | 3 |
| ORL C,/bit | C ← C or bit(反相) | 2 | 3 |
| MOV C,bit | C ← bit | 2 | 3 |
| MOV bit,C | bit ← bit | 2 | 4 |
| 位逻辑跳转 | | | |
| JC rel | 如果 C=1 跳转到 rel | 2 | 3 |
| JNC rel | 如果 C=0 跳转到 rel | 2 | 3 |
| JB bit,rel | 如果 bit=1 跳转到 rel | 3 | 4 |
| JNB bit,rel | 如果 bit=0 跳转到 rel | 3 | 4 |
| JBC bit,rel | 如果 bit=1 跳转到 rel,并且清除 bit | 3 | 5 |
| 程序跳转 | | | |
| ACALL addr11 | 绝对式子程序调用 | 2 | 6 |
| LCALL addr16 | 远程子程序调用 | 3 | 6 |
| RET | 从子程序返回 | 1 | 4 |
| RETI | 从中断返回 | 1 | 4 |
| AJMP addr11 | 绝对式跳转 | 2 | 3 |
| LJMP addr16 | 远程跳转 | 3 | 4 |

| | | | |
|---|---|---|---|
| SJMP rel | 短程跳转 | 2 | 3 |
| JMP @A+DPTR | 间接跳转 | 1 | 3 |
| JZ rel | 如果 Acc=0 则跳到 rel | 2 | 3 |
| JNZ rel | 如果 Acc≠0 则跳到 rel | 2 | 3 |
| CJNE A,direct,rel | 如果 Acc≠direct 则跳到 rel | 3 | 5 |
| CJNE A,#data,rel | 如果 Acc≠data 则跳到 rel | 3 | 4 |
| CJNE Rn,#data,rel | 如果 Rn≠data 则跳到 rel | 3 | 4 |
| CJNE @Ri,#data,rel | 如果 Ri≠data 则跳到 rel | 3 | 5 |
| DJNZ Rn,rel | 如果(Rn-1)≠0 则跳到 rel | 2 | 4 |
| DJNZ direct,rel | 如果(direct-1)≠0 则跳到 rel | 3 | 5 |
| NOP | 无动作 | 1 | 1 |

# 25. 封装尺寸

## 25.1. DIP-20

图 25–1. DIP-20




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

UNIT : INCH

NOTES:

1.JEDEC OUTLINE : MS-001 AD

2."D","E1" DIMENSIONS DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 INCH.

3.eA IS MEASURED AT THE LEAD TIPS WITH THE LEADS UNCONSTRAINED.

4.POINTED OR ROUNDED LEAD TIPS ARE PREFERRED TO EASE INSERTION.

5.DISTANCE BETWEEN LEADS INCLUDING DAM BAR PROTRUSIONS TO BE .005 INCH MININUM.

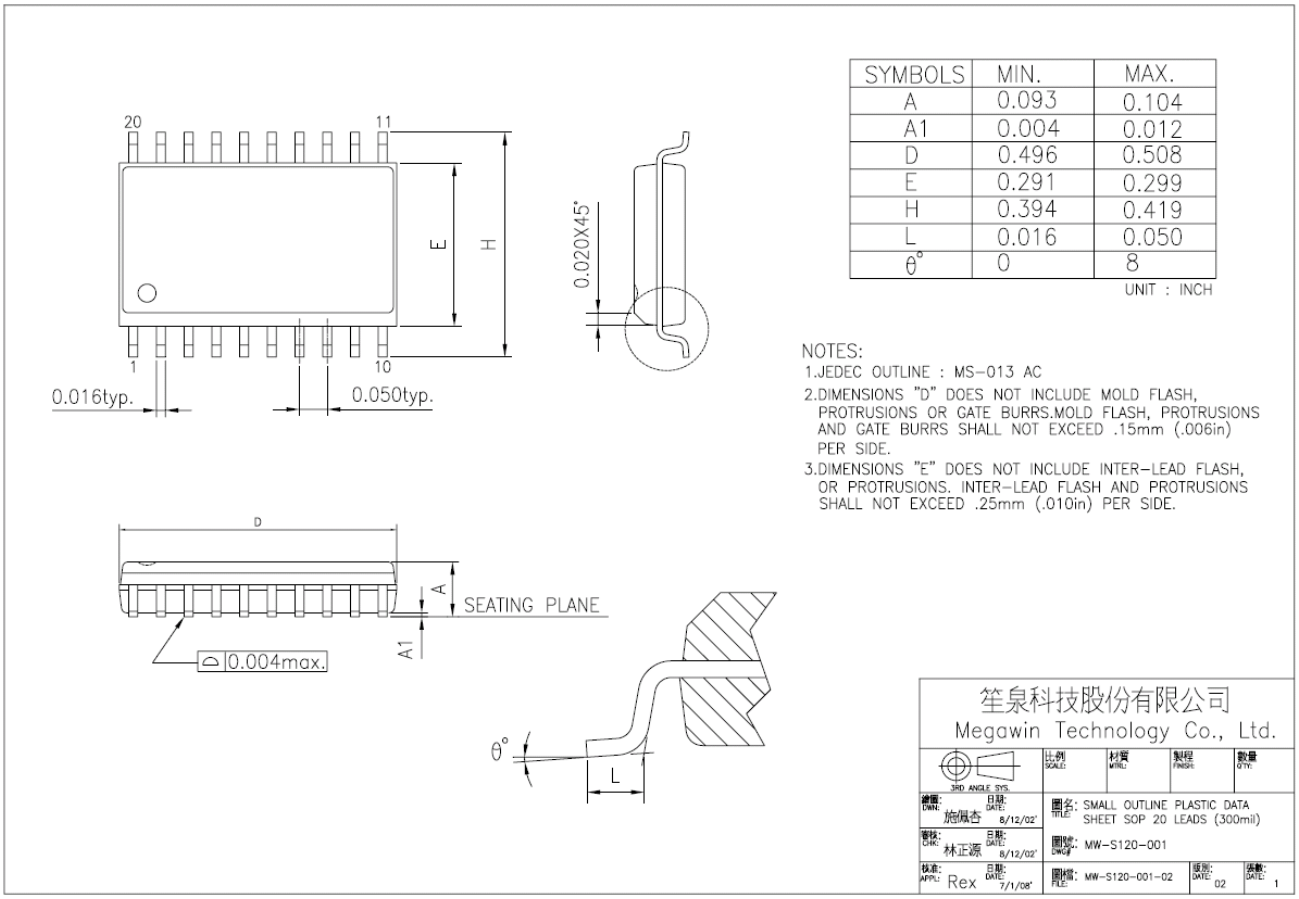
6.DATUM PLANE [H] COINCIDENT WITH THE BOTTOM OF LEAD, WHERE LEAD EXITS BODY.

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

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

## 25.2. SOP-20

图 25–2. 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 |

UNIT : INCH

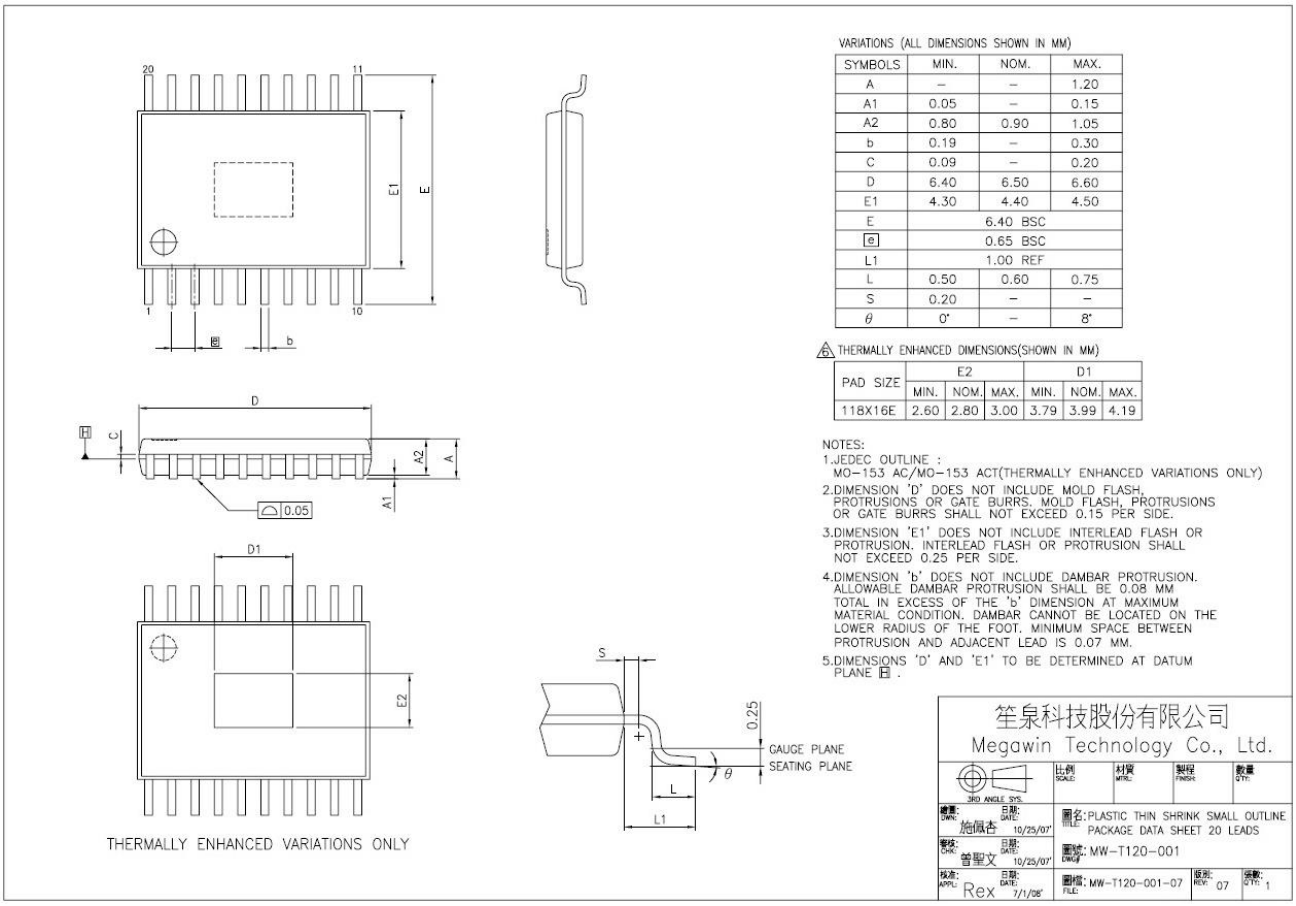
NOTES:
1.JEDEC OUTLINE : MS-013 AC
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED .15mm (.006in) PER SIDE.
3.DIMENSIONS "E" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED .25mm (.010in) 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: | 製程 FINISH: | 數量 QTY: |
| 繪圖: 施佩杏 日期: 8/12/02' | 圖名: SMALL OUTLINE PLASTIC DATA SHEET SOP 20 LEADS (300mil) | | | |
| 審核: 林正源 日期: 8/12/02' | 圖號: MW-S120-001 | | | |
| 核准: Rex 日期: 7/1/08' | 檔名: MW-S120-001-02 | 版次: 02 | 張數: 1 |

## 25.3. TSSOP-20

图 25–3. TSSOP-20




VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

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

△6 THERMALLY ENHANCED DIMENSIONS(SHOWN IN MM)

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

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

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

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

# 26. 修订历史

表 26–1. 修订历史

| 版本 | 说明 | 日期 |
|---|---|---|
| A1 | 1. 初稿 | 2005/09 |
| | | |
| A2 | 1. 脚位定义. | 2006/01 |
| | 2. 工作温度. | 2006/01 |
| | 3. 改正 CCAPPn 为 CAPPn. | 2006/01 |
| | 4. 改正 SFR PWMMSBn 为 PCAPWMn. | 2006/01 |
| | 5. 改正 PWM 图表. | 2006/01 |
| | | |
| A3 | 1. 修正合理的工作温度. | 2006/08 |
| | | |
| A4 | 1. 低电压检测加特别的注释. | 2006/12 |
| | | |
| A5 | 1. 修改存储温度. | 2007/03 |
| | | |
| A6 | 1. 工作频率范围上升到 24 MHz. | 2007/11 |
| | | |
| A7 | 1.增加写 Flash 时 2.7V 为必须要求. | 2007/12 |
| | 2.修改极限参数. | |
| | | |
| A8 | 1..改变格式. | 2008/12 |
| | | |
| A9 | 1. 改变格式. | 2012/11 |
| | 2. 增加示例代码. | 2012/11 |
| | 3. 工作频率上升到 25 MHz. | 2012/11 |
| | 3. .增加 SMOD0 描述. | 2012/11 |
| | | |
| | | |
| | | |
| | | |

## 免责声明

在此，笙泉（Megawin）代表“***Megawin Technology Co., Ltd.***”

## 生命支援

此产品并不是为医疗、救生或维持生命而设计的，并且当设备系统出现故障时，并不能合理地预示是否会对人身造成伤害。因此，当客户使用或出售用于上述应用的产品时，需要客户自己承担这样做的风险，笙泉公司并不会对不当地使用或出售我公司的产品而造成的任何损害进行赔偿。

## 更改权

笙泉保留产品的如下更改权，其中包括电路、标准单元、与/或软件 – 在此为提高设计的与/或性能的描述或内容。当产品在大批量生产时，有关变动将通过工程变更通知（ECN）进行通知。