Sonix（松翰）8bit 单片机
26 系列 I/O 型原理及基础课件（一）

主要内容：

1．Sonix 26xx 单片机简介；
2．Sonix 26xx 单片机特点；
3．Sonix 26xx 单片机内部结构；
4．Sonix 26xx 单片机资源；
5．Sonix 26xx 单片机 ROM .RAM 的映射；
6．Sonix CPU 寄存器（ACC，PFLAG，PC，Y，Z，R）；

为了让更多的工程师更加方便、快捷的了解和使用 SONIX 单片机，从而我们编写 SONIX 单片机系列的培训课件，主要详细的介绍了 SONIX 26 系列芯片的硬件模块、指令以及开发环境，供业界工程师交流经验，由于时间仓促，请提出宝贵的意见。

SONIX 官方网站：http:www.sonix.com.tw
Sonix 单片机咨询邮箱：howard_tone@hotmail.com
Sonix 单片机咨询电话：13725134515

一．Sonix 26xx 单片机简介:

Sonix 8 位微控制器具有高性能,低功耗的特点.指令执行时间为 1 个时钟震荡周期的结构使工作速度可达 16MIPS.高 EFT（**EFT 特性： EN61000-4-4 标准下稳超 4000V**）能力,适合在高干扰环境下工作，尤其适合小家电产品。IC 结构设计一流,包括一个 1-4K-word 的程序存储器（OTP ROM），48-128-byte 的数据存储器, 两个 8 位定时/计数器(T0,TC0)，一个看门狗定时器，三个中断源(T0、TC0、INT0)，以及 4 层堆栈缓存区。此外，用户可自行选择芯片的振荡形式,有四种不同的外部振荡结构提供给系统作为时钟源：高/低速晶体振荡器/陶瓷谐振器和 RC 振荡器等。还可以通过程序设定内部 RC 振荡器作为低速模式时钟源。

二．Sonix 26xx 单片机特点：

- **存储器配置**
  OTP ROM：1-4K * 16 bits.
  RAM： 48 -128* 8 bits.
  4 层堆栈缓存器

- **I/O 引脚配置**
  双向输入/输出：P0, P1, P2, P5.
  单向输入：P1.1.
  可编程楼极开路引脚：P1.0, P1.1
  唤醒功能：P0, P1 电平改变触发
  内部上拉：P0, P1, P2, P5.
  外部中断： P0

- **功能强大的指令集**
  每个指令周期为一个时钟周期（1T）
  大多数指令的执行时间均为一个指令周期
  JMP 指令可在整个 ROM 区执行
  CALL 指令可在整个 ROM 区执行
  查表功能（MOVC）可寻址整个 ROM 区

- **3 个中断源**
  两个内部中断源：T0, TC0
  一个外部中断源：INT0

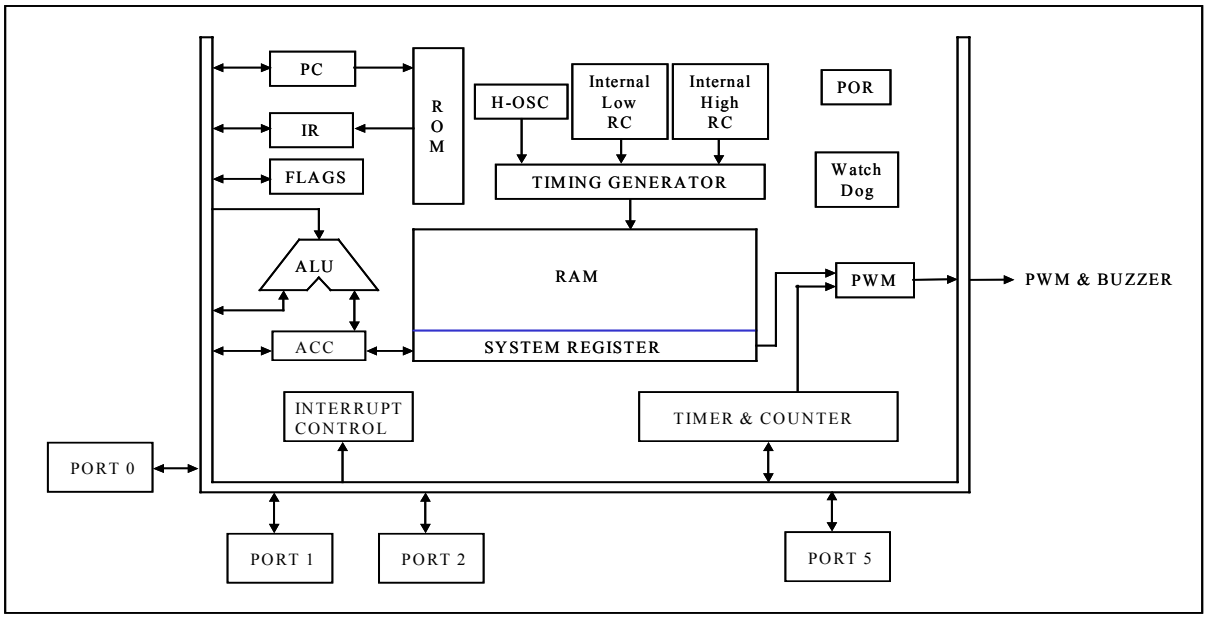
- **2 个 8 位定时/计数器**
  T0：基本定时器/RTC(实时时钟)
  TC0：自动装载定时/计数器/PWM0/Buzzer 输出

- **内置看门狗定时器**

- **系统时钟和操作模式**
  外部高速时钟：RC 最大 10 MHz
  外部高速时钟：晶体 最大 16 MHz
  内部低速时钟：RC 16KHz(3V), 32KHz(5V)
  内部高速时钟：RC 16MHz
  普通模式：高低速时钟都能运行
  低速模式：只有低速时钟运行
  睡眠模式：高低速时钟都停止
  绿色模式：由 T0 定时器周期性唤醒

三．Sonix 26xx 单片机内部结构：

SN8P2501



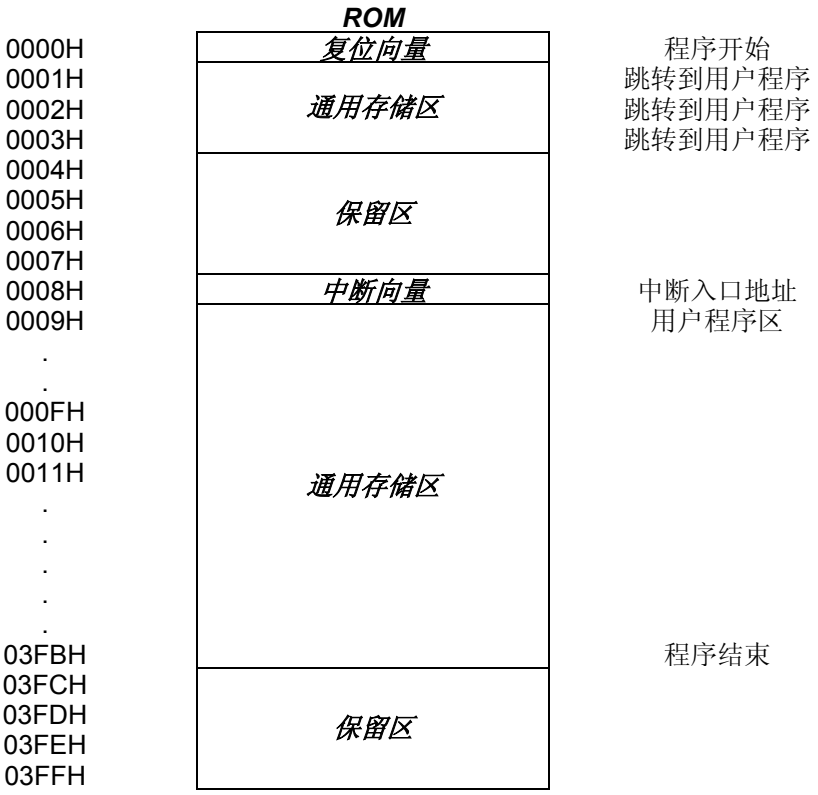
Sonix 26xx 单片机资源：

CHIP	ROM	RAM	堆栈层数	定时器		RTC	I/O	绿色模式	PWM	唤醒功能引脚数目	封装
				T0	TC0				Buzzer		
SN8P2501A	1K*16	48	4	V	V	V	12	V	V	5	DIP14/SOP14/SSOP16

SN8P1602B	**1K*16**	**48**	**4**	**-**	**V**	**-**	**14**	**V**	**-**	**6**	**DIP18/SOP18/SSOP20**
SN8P2602B	**1K*16**	**48**	**4**	**V**	**V**	**-**	**15**	**V**	**V**	**7**	**DIP18/SOP18/SSOP20**
SN8P2624	**2K*16**	**128**	**8**	**V**	**V**	**-**	**24**	**V**	**V**	**11**	**SK-DIP28/SOP28/SSOP28**
SN8P2604	**4K*16**	**128**	**8**	**V**	**V**	**-**	**24**	**V**	**V**	**11**	**SK-DIP28/SOP28/SSOP28**

四．Sonix 26xx 单片机程序 ROM 和 RAM 的 MAP：

- **1K words ROM**



1.1.1.1 复位向量地址（0000H）

一个字节的复位向量地址通常用来执行系统复位。

- **上电复位(NT0 = 1，NPD = 0)**
- **看门狗复位(NT0 = 0，NPD = 0)**
- **外部复位(NT0 = 1，NPD = 1)**

上电复位或看门狗溢出复位后，系统从地址 0000H 开始重新执行程序，所有的系统寄存器恢复为默认值。可以从 PFLAG 寄存器的 NT0、NPD 标志判断复位状态。下面的例子给出了如何在程序存储器里定义复位向量

➲ 例：定义复位向量

```
CHIP SN8P2501A
                ORG     0               ; 0000H
                JMP     START           ; 跳转到用户程序
                .                       ; 0004H ~ 0007H 保留

                ORG     10H
START:                                  ; 0010H, 用户程序的首地址
                .                       ; 用户程序
                .
                ENDP                    ; 程序结束
```

1.1.1.2 中断向量地址（0008H）

一旦有中断响应，程序计数器（PC）的值就会存入堆栈缓存器中并跳转至 0008H 处执行中断服务程序。用户使用时必须自行定义中断向量，并在 ORG 8 处的第一条指令必须是“JMP”或者“NOP”。下面的例子给出了如何在程序

中定义中断向量。

✸ 注： 当中断发生时，用户必须保存 **ACC** 和 **PFLAG** 寄存器的值。

✸ 注：**ORG 8** 处的第一条指令必须是**“JMP”**或者**“NOP”**。

➲ 例：定义中断向量（例 **1**）。中断服务程序位于 **ORG 8** 之后。

```
.DATA       ACCBUF      DS    1          ; 定义 ACCBUF 保存 ACC 的值
            PFLAGBUF    DS    1          ; 定义 PFLAGBUF 保存 PFLAG 的值
.CODE
            ORG         0                ; 0000H
            JMP         START            ; 跳转到用户程序
            .                            ; 0004H ~ 0007H 保留
            ORG         8                ; 中断入口地址
            NOP                          ; ORG 8 处的第一条指令
            B0XCH       A, ACCBUF        ; B0XCH 指令不会改变 PFLAG 的值
            B0MOV       A, PFLAG
            B0MOV       PFLAGBUF, A      ; 保存 PFLAG 的值
            .
            B0MOV       A, PFLAGBUF
            B0MOV       PFLAG, A         ; 恢复 PFLAG 的值
            B0XCH       A, ACCBUF        ; B0XCH 指令不会改变 PFLAG 的值
            RETI                         ; 中断返回
START:                                   ; 用户程序起始地址
            .                            ; 用户程序
            JMP         START            ; 用户程序结束

            ENDP                         ; 程序结束
```

➲ 例：定义中断向量。中断服务程序位于用户程序后面。

```
.DATA       ACCBUF      DS    1          ; 定义 ACCBUF 保存 ACC 的值
            PFLAGBUF    DS    1          ; 定义 PFLAGBUF 保存 PFLAG 的值
.CODE
            ORG         0                ; 0000H
            JMP         START            ; 跳转到用户程序
            .                            ; 0004H ~ 0007H 保留
            ORG         08
            JMP         MY_IRQ           ; 0008H, 中断向量地址

            ORG         10H
START:                                   ; 0010H, 用户程序起始地址
            .                            ; 用户程序
            JMP         START            ; 用户程序结束
MY_IRQ:                                  ; 中断服务程序起始地址
            B0XCH       A, ACCBUF        ; B0XCH 指令不会改变 PFLAG 的值
            B0MOV       A, PFLAG
            B0MOV       PFLAGBUF, A      ; 保存 PFLAG 的值
            .
            B0MOV       A, PFLAGBUF
            B0MOV       PFLAG, A         ; 恢复 PFLAG 的值
            B0XCH       A, ACCBUF        ; B0XCH 指令不会改变 PFLAG 的值
            RETI                         ; 中断服务程序结束

            ENDP                         ; 程序结束
```

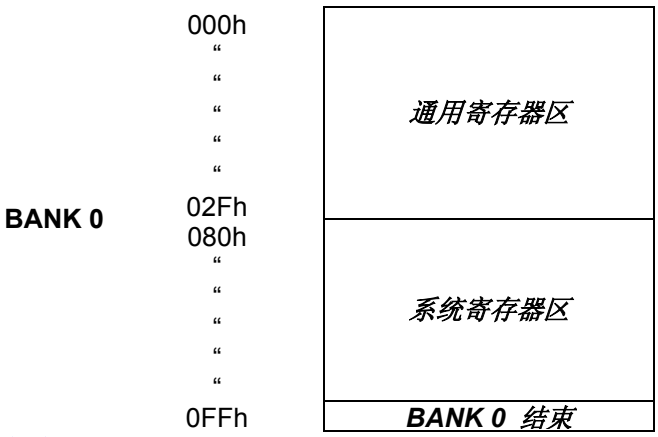
✸ 注意：从上面的程序中可以得出 **SONIX** 的主要编程规则，有以下几点：

1. 地址 **0000H** 处的 **“JMP”**指令使程序从头开始执行。
2. **0004H~0007H** 是系统保留区，不允许用户使用。

4..2 数据存储器（RAM）：

➢ **48*8-bit RAM**

地址	*RAM 位置*



5. CPU 寄存器（ACC，PFLAG，PC，Y，Z，R）

5.1 累加器

累加器 ACC 是一个 8 位专用数据寄存器，用来进行算术逻辑运算或数据存储器之间数据的传送和处理。如果对 ACC 的操作结果为零（Z）或者有进位产生（C 或 DC），那么这些标志将会影响 PFLAG 寄存器。

由于 ACC 不在数据存储器（RAM）中，所以执行“B0MOV”指令不能够访问 ACC，必须通过 “MOV”指令对 ACC 进行读/写。

➲ **例：读/写 ACC 中的数据**

```
; 把 ACC 中的数据送到 BUF 中

                MOV     BUF, A

; 给 ACC 送立即数

                MOV     A, #0FH

; 把 BUF 的数据送给 ACC

                MOV     A, BUF
```

中断响应时，系统不会自动保存 ACC 和 PFLAG，所以如果中断发生，必须将 ACC 和 PFLAG 寄存器值存储在由用户自定义的数据存储器中，如下所示：

➲ **例：保护 ACC 和工作寄存器**

```
ACCBUF          EQU     00H             ; ACCBUF 用来保存 ACC 的数据
PFLAGBUF        EQU     01H             ; PFLAGBUF 用来保存 PFLAG 的数据

INT_SERVICE:
                B0XCH   A, ACCBUF       ; 保存 ACC 的值
                B0MOV   A, PFLAG        ; 保存 PFLAG 的值
                B0MOV   PFLAGBUF,A
                .       .
                .
                .
                B0MOV   A, PFLAGBUF     ; 恢复 PFLAG 的值
                B0MOV   PFLAG,A
                B0XCH   A, ACCBUF       ; 用 BOXCH 指令恢复 ACC 的值不会改变 PFALG 的值

                RETI                    ; 中断返回
```

✷ 注：为了保护并恢复 ACC，必须使用 “B0XCH”指令，否则会影响 PFLAG。

1.1.1.3 程序状态寄存器（PFLAG）

PFLAG 包括复位标志，进位标志（C），辅助进位标志（DC）和零标志（Z），如果操作结果为 0 或是有进位、借位发生，就存入 PFLAG 寄存器中。

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	NT0	NPD	-	-	-	C	DC	Z
读/写	R/W	R/W	-	-	-	R/W	R/W	R/W
复位后	-	-	-	-	-	0	0	0

☞ 复位标志

NT0	NPD	复位状态
0	0	看门狗超时复位
0	1	保留
1	0	由 LVD 复位
1	1	由外部复位引脚复位

☞ 进位标志

C = 1：执行算术加法后有进位发生，执行算术减法后没有借位或移位指令后移出逻辑“1”
C = 0：执行算术加法后没有进位发生，执行算术减法后有借位或移位指令后移出逻辑“0”

☞ 辅助进位标志

DC = 1：执行算术加法操作产生由低字节向高字节的进位或执行算术减法操作没有从高字节借位
DC = 0：执行算术加法操作没有产生由低字节向高字节的进位或执行算术减法操作从高字节借位

☞ 零标志

Z=1：指令执行后，运算结果为零
Z=0：指令执行后，运算结果非零

5.2 程序计数器 PC

程序计数器 PC 是一个 10 位专用二进制计数器，分为 2 位的高字节和 8 位的低字节，PC 指向下一条将要执行的指令的地址 ，一般的，在程序执行过程中，PC 会随着指令的执行自动加 1。

此外，执行程序调用（CALL）和跳转（JMP）指令时，下一条将要执行的指令地址会被装入 PC 的 0～9 位。

	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PC	-	-	-	-	-	-	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
复位后	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
					PCH				PCL							

☞ 单地址跳转

单地址跳转功能共有 9 条指令： CMPRS、INCS、INCMS、DECS、DECMS、BTS0、BTS1、B0BTS0 和 B0BTS1 ，如果运算的结果符合跳转条件，则程序计数器加 2，会跳过当前指令的下一条指令。

如果位测试结果符合跳转条件，那么 PC 将加 2，跳过当前指令的下一条指令：

```
                B0BTS1      FC              ; 如果 C=1 则跳过下一条指令
                JMP         C0STEP          ; 否则跳转到 C0STEP.
                .
C0STEP:         NOP

                B0MOV       A, BUF0         ; 把 BUF0 的值赋给 ACC.
                B0BTS0      FZ              ; 如果 Z=0，则跳过下一条指令
                JMP         C1STEP          ; 否则跳到 C1STEP.
                .
C1STEP:         NOP
```

如果 ACC 与立即数或存储器中的内容相等，那么 PC 将加 2，跳过下一条指令

```
                CMPRS       A, #12H         ; 如果 ACC = 12H. 则跳过下一条指令
                JMP         C0STEP          ; 否则跳到 C0STEP.
                .
C0STEP:         NOP
```

如果增加 1 后的结果是从 0xffh 到 0x00h，那么 PC 将加 2，跳过下一条指令

INCS:

```
                INCS        BUF0
                JMP         C0STEP          ; 如果 ACC 不为 0 则跳到 C0STEP
                …
C0STEP:         NOP
```

INCMS:

```
                INCMS       BUF0
                JMP         C0STEP          ; 如果 BUF0 不为 0 则跳到 C0STEP
                …
C0STEP:         NOP
```

如果减小 1 后的结果是从 0xffh 到 0x00h，那么 PC 将加 2，跳过下一条指令

DECS:

```
                DECS        BUF0
                JMP         C0STEP          ; 如果 ACC 不为 0 则跳到 C0STEP
                …
C0STEP:         NOP
```

DECMS:

```
                DECMS       BUF0
                JMP         C0STEP          ; 如果 BUF0 不为 0 则跳到 C0STEP
                …
```

```
C0STEP:         NOP
```

☞ **多地址跳转**

用户可以通过 JMP 和“ ADD PCL，A ”指令实现多地址跳转。 “ADD PCL, A”执行后若有进位发生，进位标志并不会影响 PCH 寄存器。

➲ **例： 如果 PC = 0323H (PCH = 03H、PCL = 23H)**

```
; PC = 0323H
                MOV       A, #28H
                B0MOV     PCL, A          ; 跳转到 0328H
                .         .
                .         .
; PC = 0328H    .         .
                MOV       A, #00H
                B0MOV     PCL, A          ; 跳转到 0300H
```

➲ **例： 如果 PC = 0323H (PCH = 03H、PCL = 23H)**

```
; PC = 0323H
                B0ADD     PCL, A          ; PCL = PCL + ACC, PCH 的值不会改变.
                JMP       A0POINT         ; ACC = 0, 跳转到 A0POINT
                JMP       A1POINT         ; ACC = 1, 跳转到 A1POINT
                JMP       A2POINT         ; ACC = 2, 跳转到 A2POINT
                JMP       A3POINT         ; ACC = 3, 跳转到 A3POINT
                .         .               ;
```

5.3 Y, Z 寄存器

Y 寄存器和 Z 寄存器是 8 位缓存器，主要有以下主要功能：

- 通用工作寄存器
- 通过寄存器@YZ 用作访问 RAM 的数据指针
- 通过 MOVC 查表指令，可访问 ROM 中的数据

084H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Y	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

083H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Z	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

➲ **例：间接寻址模式访问 RAM bank 0 中 025H 单元。**

```
B0MOV     Y, #00H         ; 用 Y 确定 BANK0
B0MOV     Z, #25H         ; 用 Z 确定 RAM 中的位置
B0MOV     A, @YZ          ; 25H 的数据放到 ACC 中
```

➲ **例：用寄存器@YZ 将数据存储器中 bank 0 的通用数据存储器清零**

```
B0MOV     Y, #0           ; Y = 0, bank 0
B0MOV     Z, #07FH        ; Y = 7FH, 数据存储区的最大地址
```

```
CLR_YZ_BUF:
                CLR         @YZ             ; @YZ =0

                DECMS       Z               ;
                JMP         CLR_YZ_BUF      ;

                CLR         @YZ
END_CLR:                                    ; 完成对 BANK0 中的数据的清零
                .
```

5.4 R 寄存器

R 寄存器是一个 8 位缓存器，有两个主要功能：

- 工作寄存器
- 存放 ROM 查表的高字节数据

（MOVC 指令执行完后，指定的 ROM 地址中数据的高 8 位将存放在 R 寄存器中，低 8 位存放在 ACC 中）

082H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

✸ **注：请参考使用 R 寄存器的查表功能“查表说明”。**