
嵌入式系统中的低功耗设计

东南大学信息科学与工程学院

张圣清

zhangsq@seu.edu.cn

主要内容

- 低功耗设计的概念
 - 嵌入式系统的低功耗设计
 - MSP430系统的低功耗设计
-

主要内容

- ❑ 低功耗设计的概念
 - ❑ 嵌入式系统的低功耗设计
 - ❑ MSP430系统的低功耗设计
-

低功耗设计的重要性

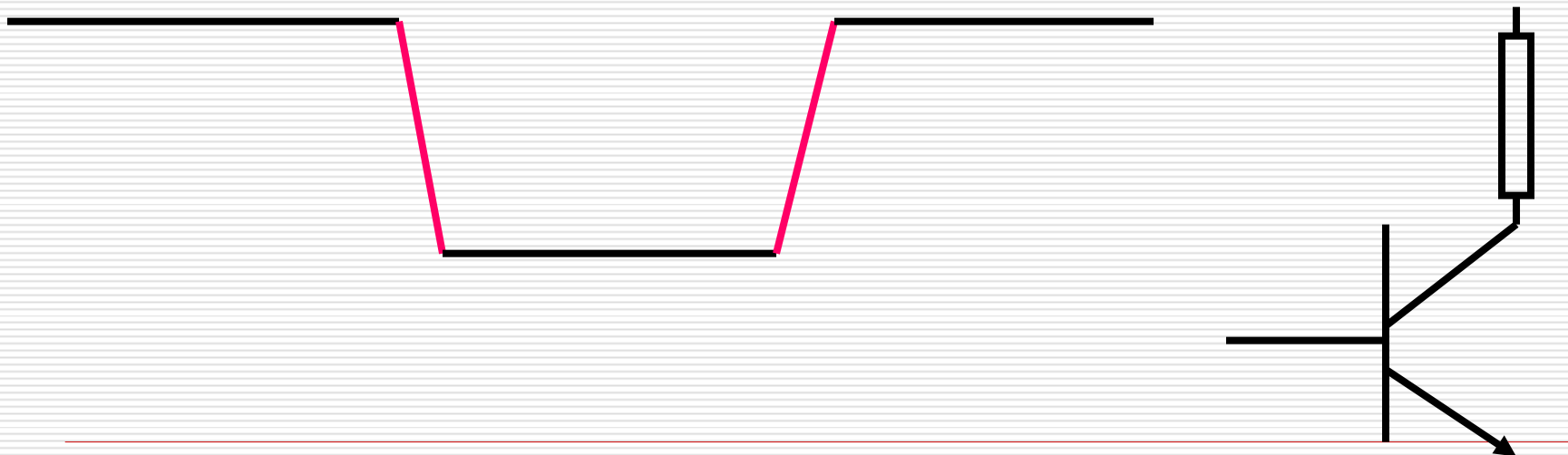
- ❑ 低功耗设计并不仅仅是省电;
 - ❑ 降低了电源模块和散热模块的成本,使产品小型化;
 - ❑ 延长了电池工作时间;
 - ❑ 减少了电磁辐射和热噪声,降低了系统的实现难度;
 - ❑ 随着设备温度的降低,设备的寿命得以延长;
-

低功耗设备的特点

1. 首先要求体积小、重量轻、便于携带。
 2. 采用低功耗电路的设计方法，
 1. 硬件
 2. 软件
 3. 电池驱动的需要。
 4. 安全的需要。
 5. 降低EMI。
 6. 节能的需要。
 7. 采用LCD液晶显示器。
 8. 采用低功耗、高抗干扰的CMOS集成电路。
-

功耗产生的原因

1. 耗能元件，如电阻等
2. $P=U \cdot I$



与功耗有关的因素

1. 电阻上消耗的功率
 2. 有源器件的开关转换阶段
 3. 集成电路内部和外部电容的充放电
 4. 系统的性能指标、负载能力、被处理信号的工作频率、电路的工作频率、电源的管理水平、零部件的性能、散热条件、接口的物理性能等对系统功耗起着重要的作用。
-

元件工艺的低功耗

1. 在集成电路设计中采用低功耗的电路设计
 2. 电源设计
 3. 存储器(SDRAM、ROM)的高速化与低功耗
 4. 高集成度的完全单片化设计
 5. 内部电路可选择性地工作
 6. 宽电源电压范围
 7. 具有高速和低速两套时钟
 8. 在线改变CPU的工作频率
 9. 后备功能(idle, power-down)
 10. 使用低功耗的元件
-

降低功耗的措施

□ 功耗的组成：动态+静态

- 动态电源管理
 - 动态电压缩放
 - 低功耗硬件设计
 - 低功耗软件设计
-

主要内容

- 低功耗设计的概念
 - 嵌入式系统的低功耗设计
 - MSP430系统的低功耗设计
-

嵌入式系统的低功耗设计

- ❑ 硬件系统的低功耗设计
 - ✓ 处理器的选择
 - ✓ 接口器件和电路的选择
 - ❑ 动态电源管理
 - ❑ 供电电路的设计
 - ❑ 软件系统的低功耗设计
-

嵌入式系统的低功耗设计

- ❑ 硬件系统的低功耗设计
 - ✓ 处理器的选择
 - ✓ 接口器件和电路的选择
 - ❑ 动态电源管理
 - ❑ 供电电路的设计
 - ❑ 软件系统的低功耗设计
-

处理器的选择

CPU的性能和功耗(动态和静态)

CPU的供电电压和工作时钟

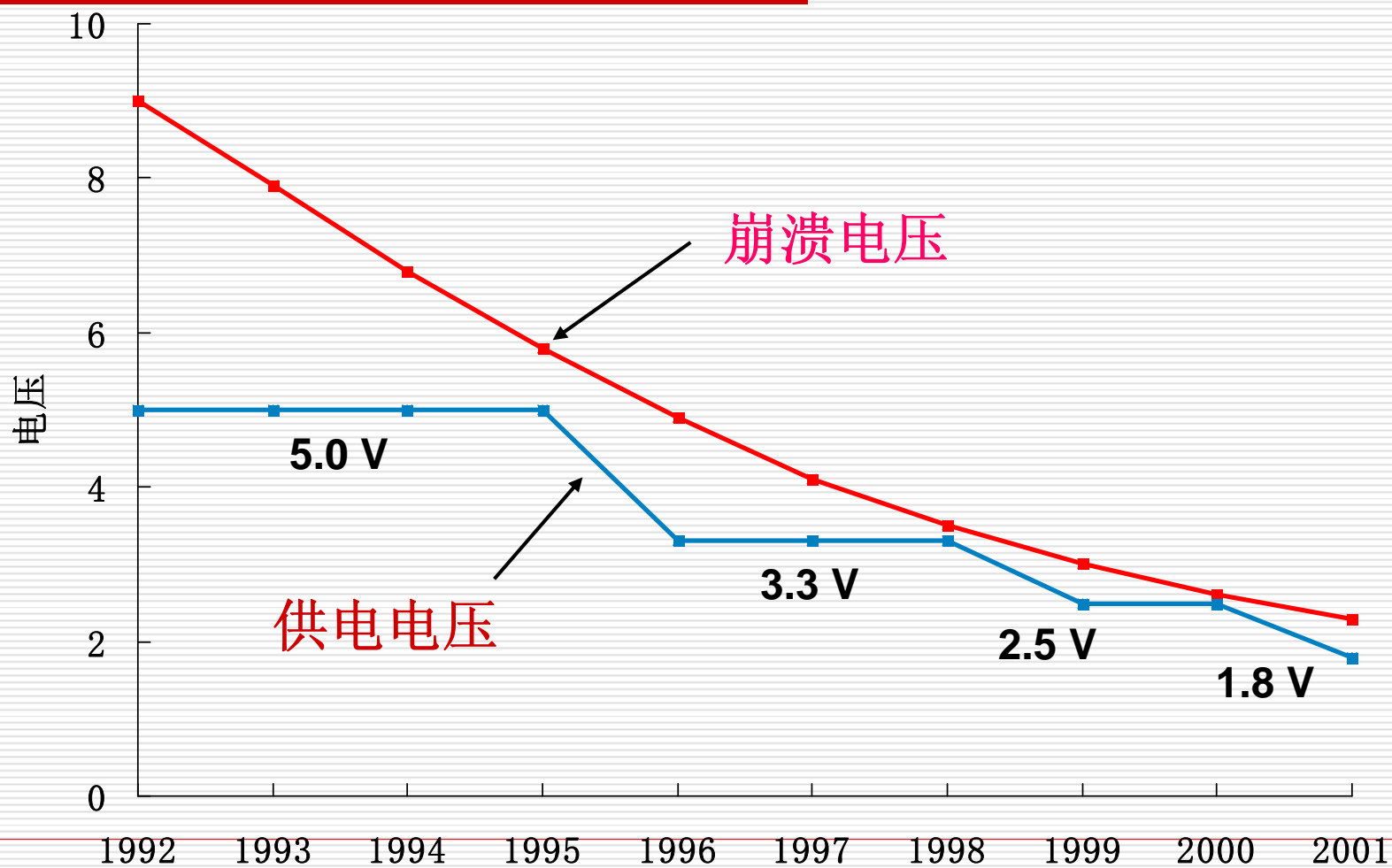
CPU的总线宽度

低功耗原则:够用即可, 尽量选择总线宽度小, 供电电压低, 工作时钟慢的, 支持低功耗模式.

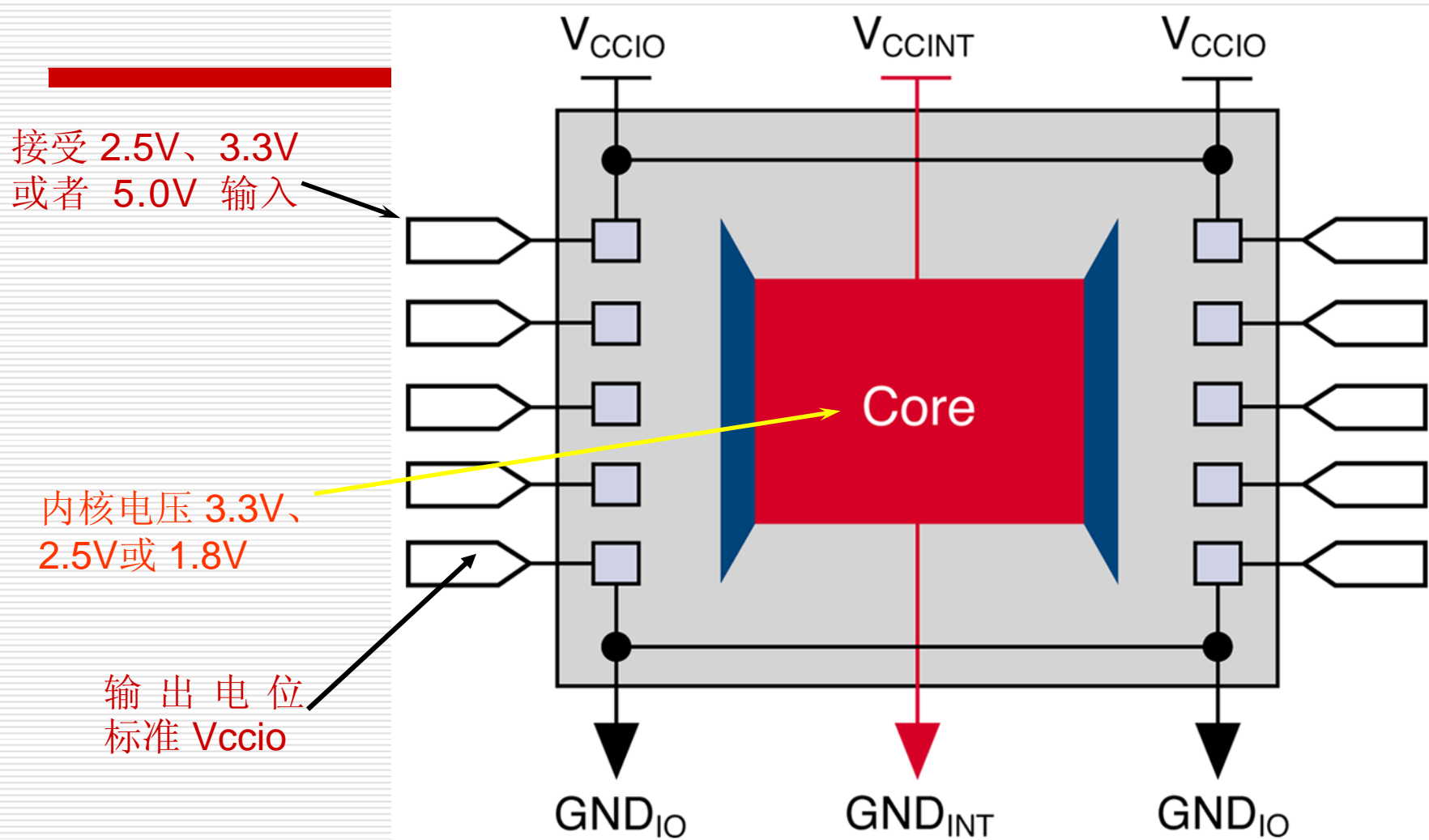
降低处理器的时钟频率

振荡器	频率	振荡模式	上电期/ μA	运行期	睡眠期/ μA
阻容	261KHz	RC	51.2	396 μA	0.32
阻容	1.13MHz	RC	51.4	510 μA	0.3
晶体	32768Hz	LP	51.2	23.5 μA	0.3
晶体	50KHz	LP	61.4	39.4 μA	0.28
晶体	1MHz	XT	92	443 μA	0.35
晶体	8MHz	HS	123	2.11mA	0.3
陶瓷	455KHz	XT	38.4	421 μA	0.34
陶瓷	8MHz	HS	143	2.5mA	0.29

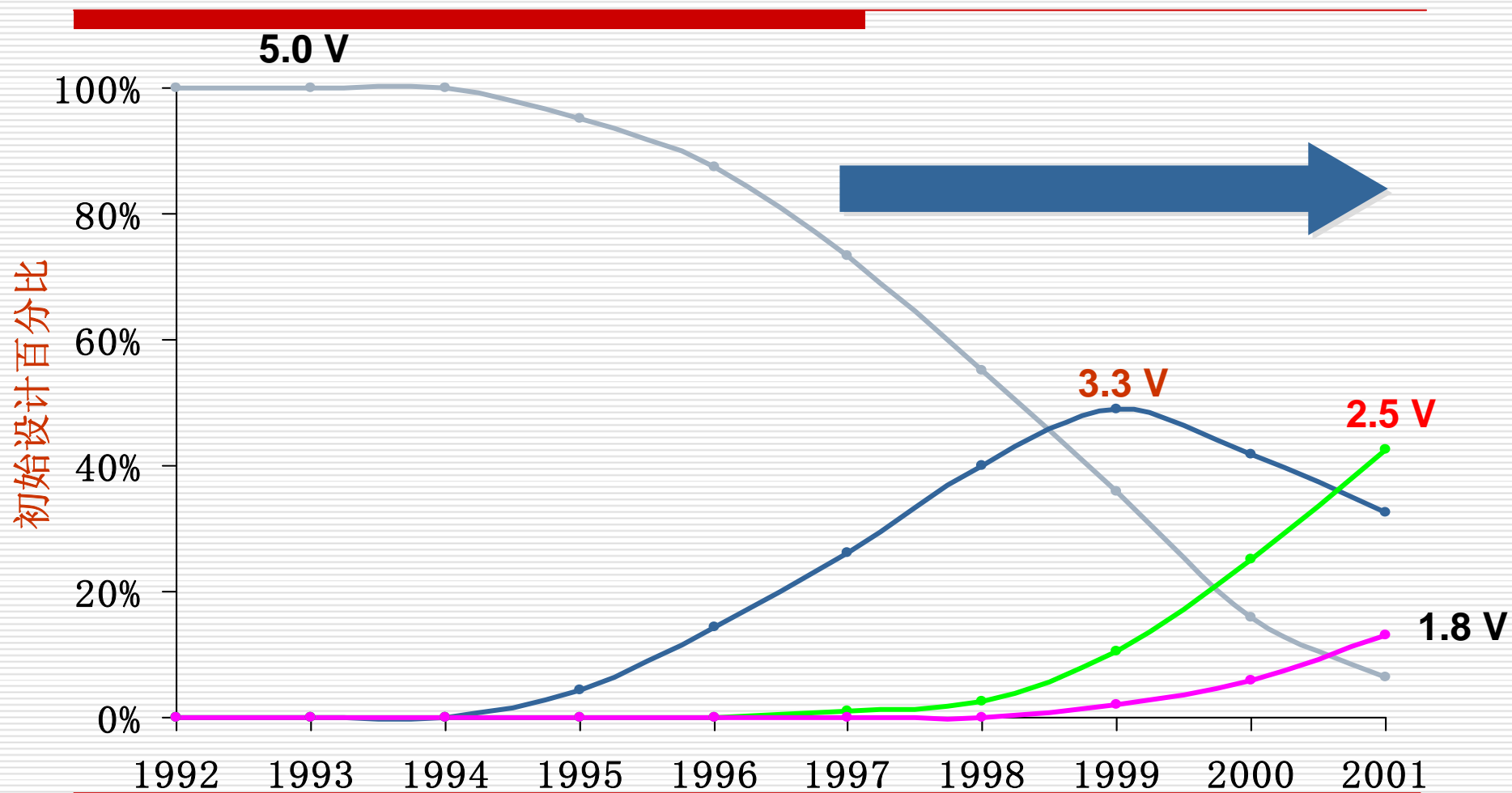
工艺改进促使供电电压降低



多电压兼容系统



不同芯核电压器件流行趋势



硬件系统的低功耗设计

□ 选择低功耗的器件

■ CMOS电路的特点和选用

□ 不用的引脚的处理-输出为高原则

□ 选用高速低频工作方式

■ 与电磁兼容性矛盾

■ 选用低功耗的处理器

■ 选用低功耗的通信收发器(网络功能)

■ 选用低功耗的存储器件

□ 慎用上拉和下拉电阻

确实需要的才使用,并合理选择阻值

□ 空闲IO的处理

不用的I/O口如果悬空的话,受外界的一点点干扰就可能成为反复振荡的输入信号了,而MOS器件的功耗基本取决于门电路的翻转次数。如果把它上拉的话,每个引脚也会有微安级的电流,所以最好的办法是设成输出,当然外面不能接其它有驱动的信号。

硬件系统的低功耗设计

☐ 选择低功耗的电路形式

■ 集成化的电路有利于降低功耗

☐ 集成化器件比分离件功耗低

■ 功率放大器

☐ A类

☐ B类

☐ AB类

☐ C类（高频功率放大器）

☐ D类

硬件系统的低功耗设计

❑ 模拟器件

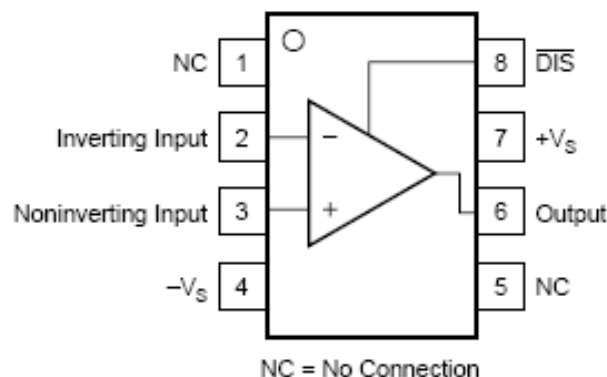
✓ 单电源、低电压供电（例LM324）

缺点：降低了系统的动态范围

✓ 选用低功耗器件,最好有关断脚

15V	220mw
10V	90mw
5V	15mw

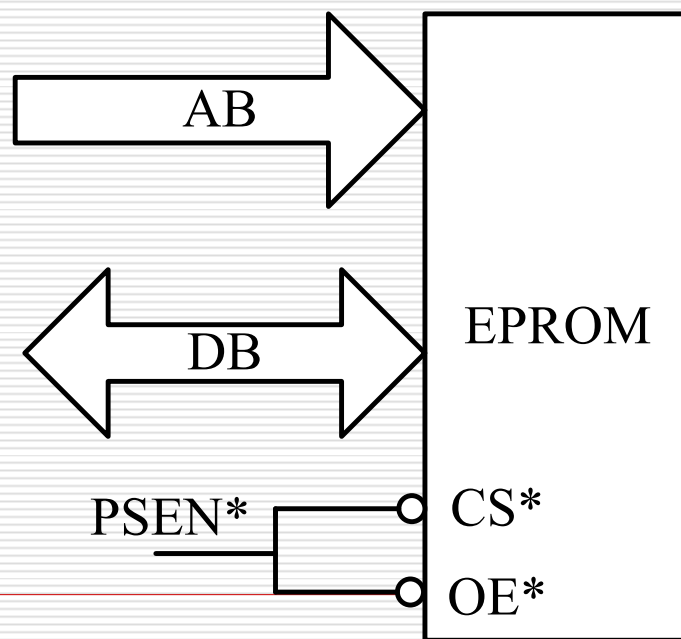
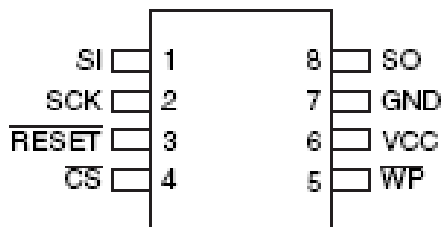
LM324供电电压与功耗的关系



硬件系统的低功耗设计

□ 数字器件

- ✓ 低电压供电
- ✓ 选用低功耗器件,接好芯片的片选脚



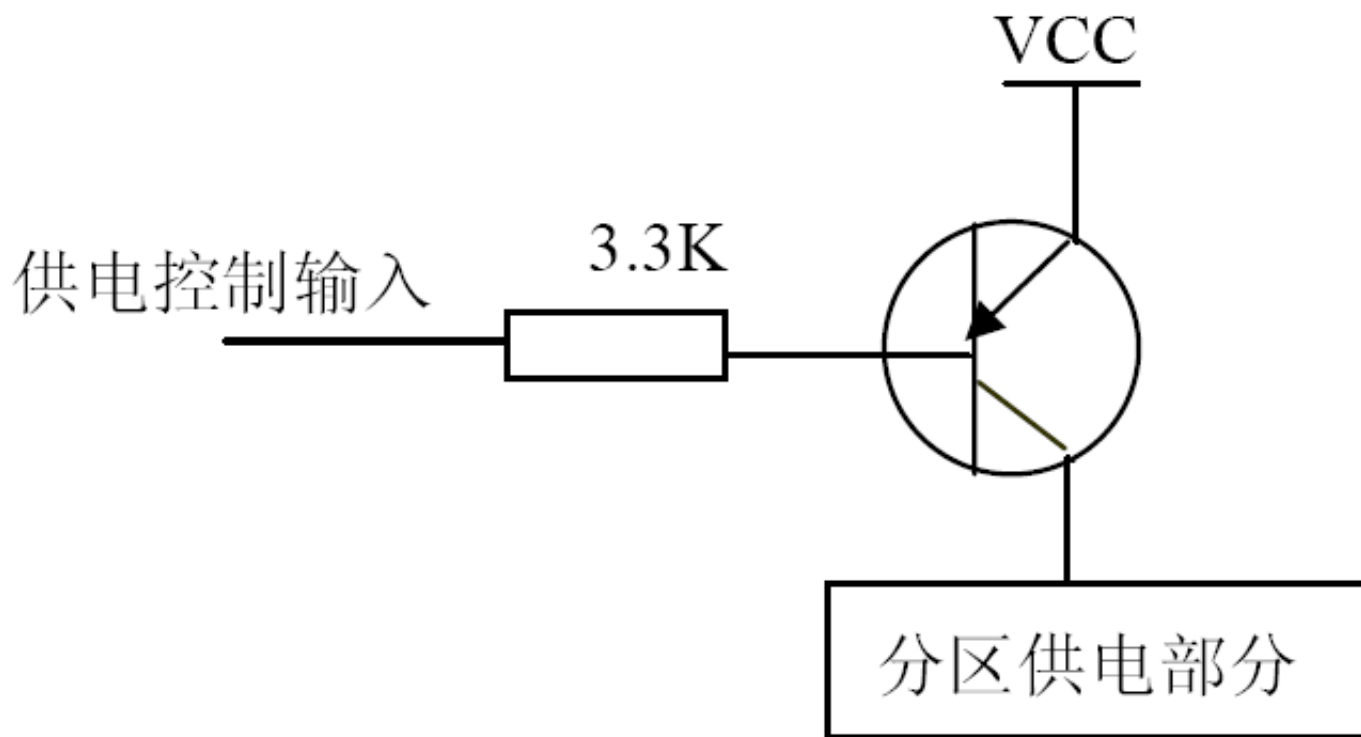
硬件系统的低功耗设计

有效利用I/O器件的待机方式

- 可编程器件（8255，8251 etc.）
 - 通信收发器器件(maxim的RS232收发器)
 - 其他可编程器件
-

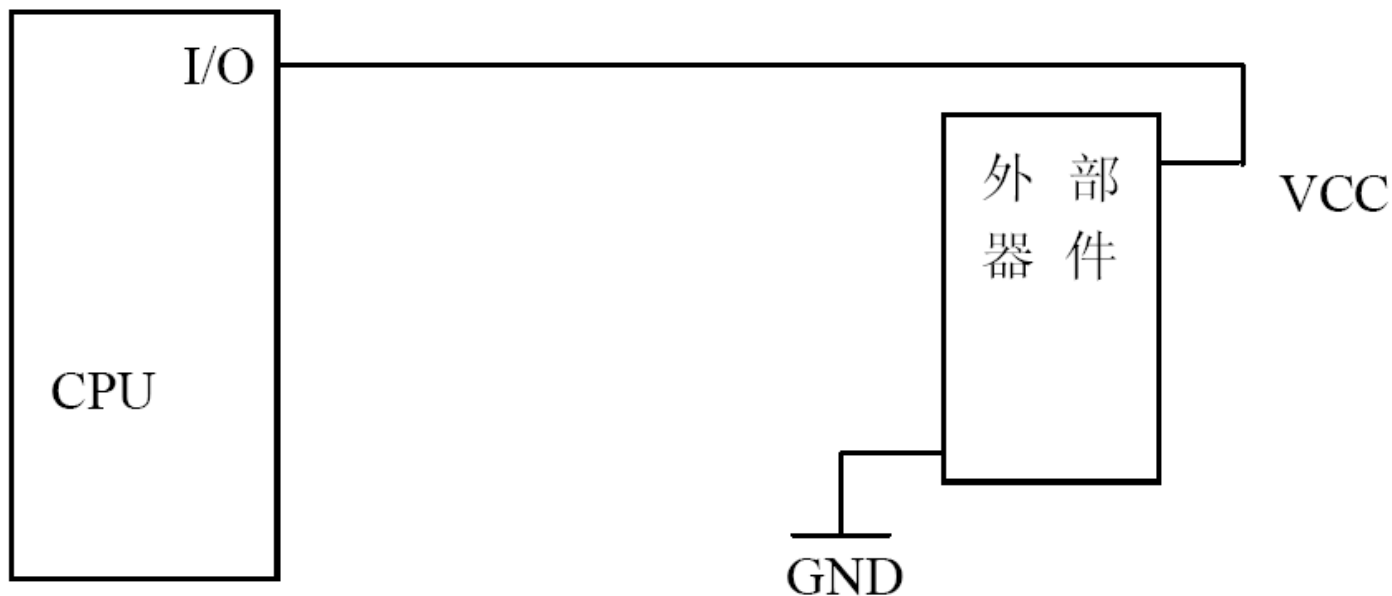
硬件系统的低功耗设计

□ 分区供电降低功耗



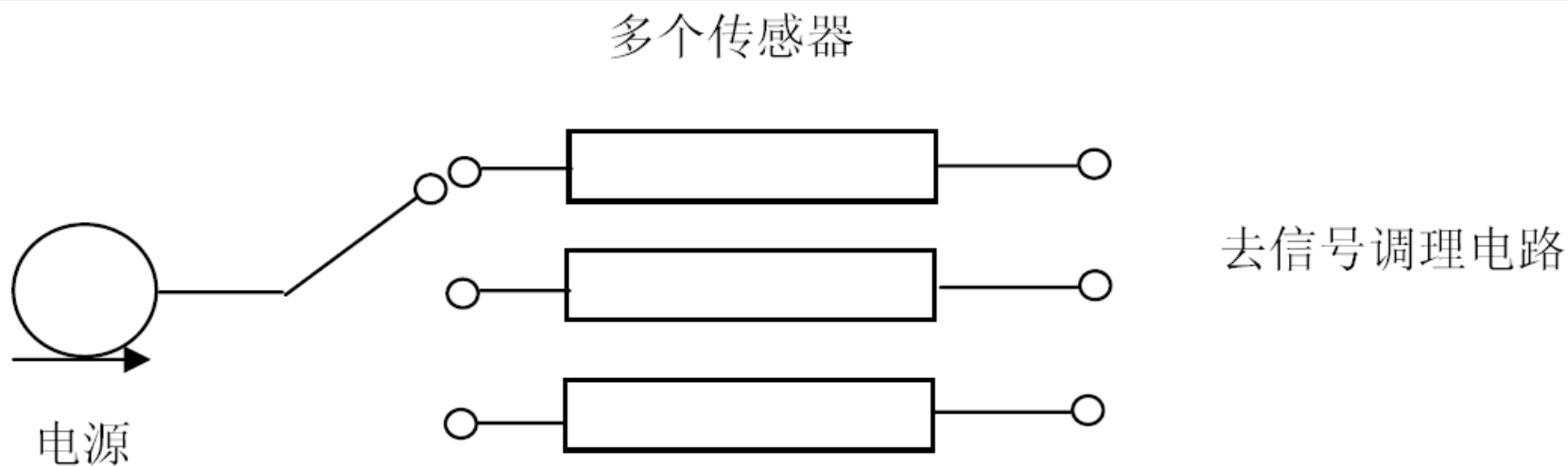
硬件系统的低功耗设计

□ 利用I/O引脚为外部设备供电



硬件系统的低功耗设计

传感器分时供电



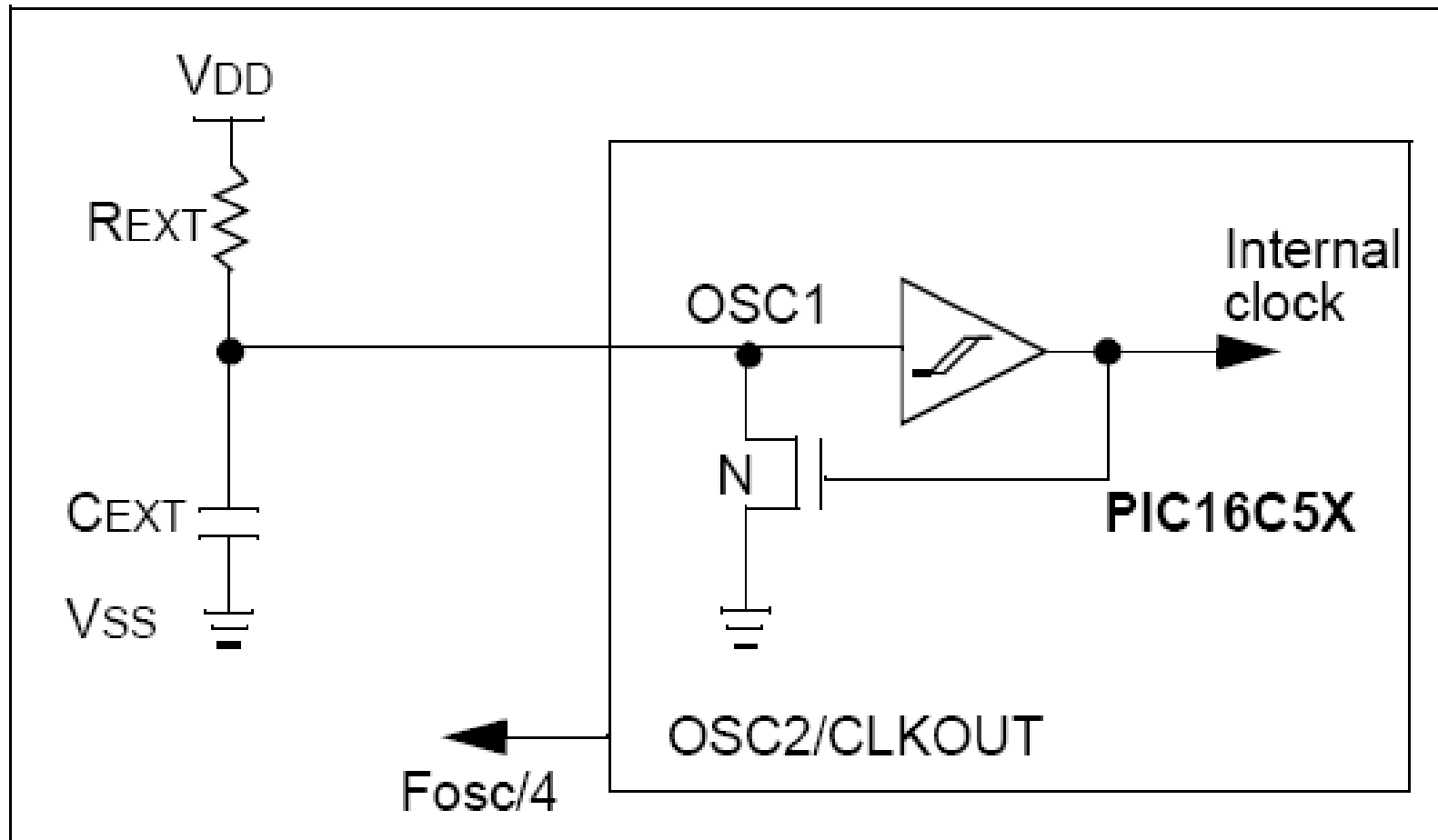
嵌入式系统的低功耗设计

- 硬件系统的低功耗设计
 - ✓ 处理器的选择
 - ✓ 接口器件和电路的选择
 - 动态电源管理
 - 供电电路的设计
 - 软件系统的低功耗设计
-

动态电源管理—选择合适的工作模式

1. 全速工作：消耗的功率最大
 2. 空闲方式
 1. 处理器的工作停止
 2. 可以响应中断
 3. 可以通过中断退出
 3. 掉电方式
 1. 不响应中断
 2. 只能复位退出
-

动态电源管理—动态改变CPU的时钟



动态电源管理

- 关闭不使用的外设
 - 降低系统的持续工作电流
 - 改变状态用电，维持状态不用电
 - 在一些机电系统设计中，尽量使系统在状态转换时消耗电流，在维持工作时刻不消耗电流。例如IC卡水表、煤气表等，在打开和关闭开关时给相应的机构上电，开关的开和关状态通过机械机构保持，而不通过电流保持，可以进一步降低电能的消耗。
-

嵌入式系统的低功耗设计

- ❑ 硬件系统的低功耗设计
 - ✓ 处理器的选择
 - ✓ 接口器件和电路的选择
 - ❑ 动态电源管理
 - ❑ 供电电路的设计
 - ❑ 软件系统的低功耗设计
-

电源电路设计

□ 线性稳压(包括LDO)

□ DC-DC

线性稳压

□ 优点:

电路结构简单，所需元件数量少，输入和输出压差可以很大

□ 缺点:

其致命弱点就是效率低，功耗高。其效率完全取决于输入输出电压差

LM7805, LM1117等等

DC to DC

- DC to DC电路的特点是效率高，升降压灵活，但缺点是电路相对复杂，干扰较大。一般常见的由Boost和Buck两种电路，前者用于升压，后者用于降压

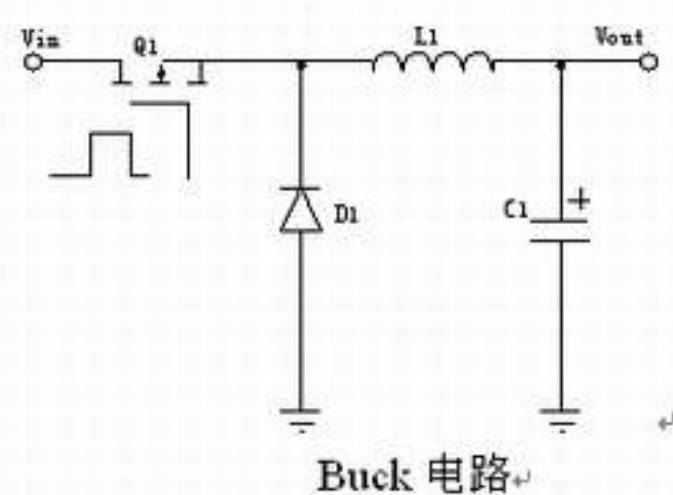
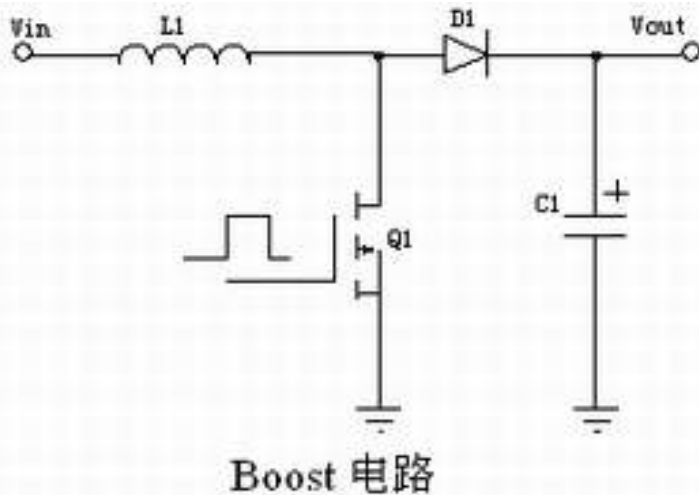


图2. Boost 和 Buck 电路

关于电池供电系统

□ 使用电池的考虑

- 工作电压范围多大？
 - 电流消耗速率为多少？
 - 是否有尺寸约束？
 - 系统将使用多长时间？
 - 可以承受的电池价格是多少？
 - 系统在什么温度范围工作？
-

关于电池供电系统

	碳锌	碱性	镍镉	锂	镍氢	锌空气	氧化银
表称电压/V	1.5	1.5	1.2	3.0	1.2	1.4	1.6
工作电压/V	1.25~1.15	1.25~1.15	1.25~1.00	2.5~3.0	1.25~1.00	1.35~1.00	1.5
终止电压/V	0.8	0.9	0.9	1.75	0.9	0.9	0.9
工作温度/°C	-5	-20	-40	-30	-20	0	-20
	-45	-55	-70	-70	-50	-45	-50
能量密度 /wh · kg	70	85	30	300	55	300	100
容量/Ah	0.06	0.03	0.15	0.035	0.5	0.05	0.015
	-18	-45	-4	-4	-5	-0.52	-0.21
相对价价格	低	低	中	高	高	高	高
可否充电	否	否	可	可/否	可	否	否

嵌入式系统的低功耗设计

- ❑ 硬件系统的低功耗设计
 - ✓ 处理器的选择
 - ✓ 接口器件和电路的选择
 - ❑ 动态电源管理
 - ❑ 供电电路的设计
 - ❑ 软件系统的低功耗设计
-

软件系统的低功耗设计

1. 编译低功耗优化技术
 2. 硬件软件化
 3. 尽量减少处理器的工作时间
 4. 采用快速的算法
 5. 通信系统中提高通信的波特率
 6. 数据采集系统中尽量降低采集的速率
 7. 延时程序的设计考虑
 8. 软件设计成中断驱动方式
 9. 睡眠方式
 10. 静态显示
-

软件系统的低功耗设计

编译低功耗优化技术

1. 不同的指令执行的时间不同，消耗的功率不同
2. 编译优化分类：
 1. 代码大小
 2. 执行时间
 3. 节省功率
目前实现起来比较困难

软件系统的低功耗设计

硬件软件化

- 只要是硬件电路，必然消耗功率
 - 方案
 - 硬件完成的工作由软件实现：信号的抗干扰处理、滤波等
 - 综合考虑
 - 例：多媒体信号解码
 - 硬件解码：CPU性能要求低
 - 软件解码：CPU性能要求高
 - 比较功耗？
-

软件系统的低功耗设计

尽量减少处理器的工作时间

- ❑ 嵌入式系统中，CPU的运行时间对系统的功耗影响极大，故应尽可能的缩短CPU的工作时间，较长地处于空闲方式或掉电方式是软件设计降低单片机系统功耗的关键。在开机时靠中断唤醒CPU，让它尽量在短时间内完成对信息或数据的处理，然后就进入空闲或掉电方式，在关机状态下让它完全进入掉电方式，用定时中断、外部中断或系统复位将它唤醒。
 - ❑ 这种设计软件的方法是所谓的事件驱动的程序设计方法。
-

软件系统的低功耗设计

采用快速的算法

- ❑ 数字信号处理中的运算，采用如FFT和快速卷积等，可以大量节省运算时间，从而减少功耗；
 - ❑ 在精度允许的情况下，使用简单函数代替复杂函数作近似，也是减少功耗的一种方法。
 - ❑ 使用简单的数据类型（整型/浮点型）
 - ❑ 使用查表的方法
-

软件系统的低功耗设计

通信系统中提高通信的波特率

- ❑ 在多机通信中，通过对通信模块的合理设计，尽量地提高传送的波特率（即每秒钟串行口传送的数据位数）提高通信速率，意味着通信时间的缩短。
 - ❑ 发送、接收均应采用外部中断处理方式，而不采用查询方式。
-

软件系统的低功耗设计

数据采集系统中尽量降低采集的速率

- A/D转换需要消耗功率，采集的速度快，则
 - 消耗的功率多
 - 产生的数据量大，处理数据需要大量的CPU时间，消耗功率
 - 数据处理，抗干扰处理等
 - 数据的传输需要消耗功率
-

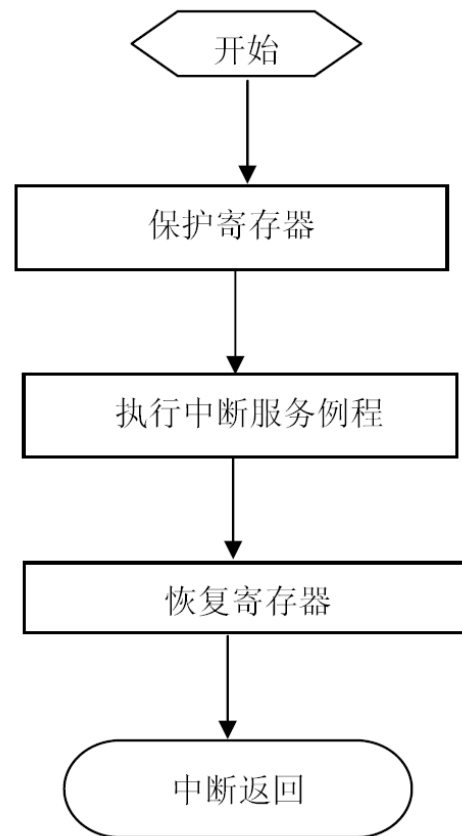
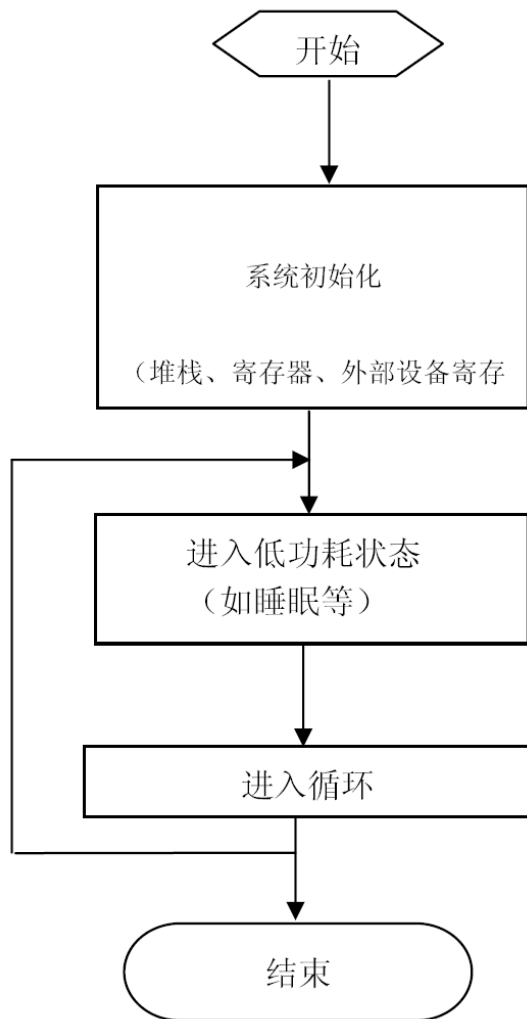
软件系统的低功耗设计

延时程序的设计考虑

- 延时方法
 - 软件
 - 硬件
 - 降低功耗的方案
 - 硬件定时器方法
 - 定时器工作于中断方式
-

软件系统的低功耗设计

软件设计成中断驱动方式



中断服务程序

软件系统的低功耗设计

使用睡眠方式

- ❑ 许多单片机设有低功耗模式，即睡眠方式（SLEEP）。使用单片机的睡眠方式可以减少单片机的工作时间，降低单片机功耗。
 - ❑ 便携式智能仪器在较长时间不进行测量采样和数据处理时，可以让单片机执行一条“SLEEP”指令，进入低功耗模式。
-

软件系统的低功耗设计

显示方法

☐ LED显示方式

■ 静态

- ☐ 需要的电路多

- ☐ 电路消耗功率

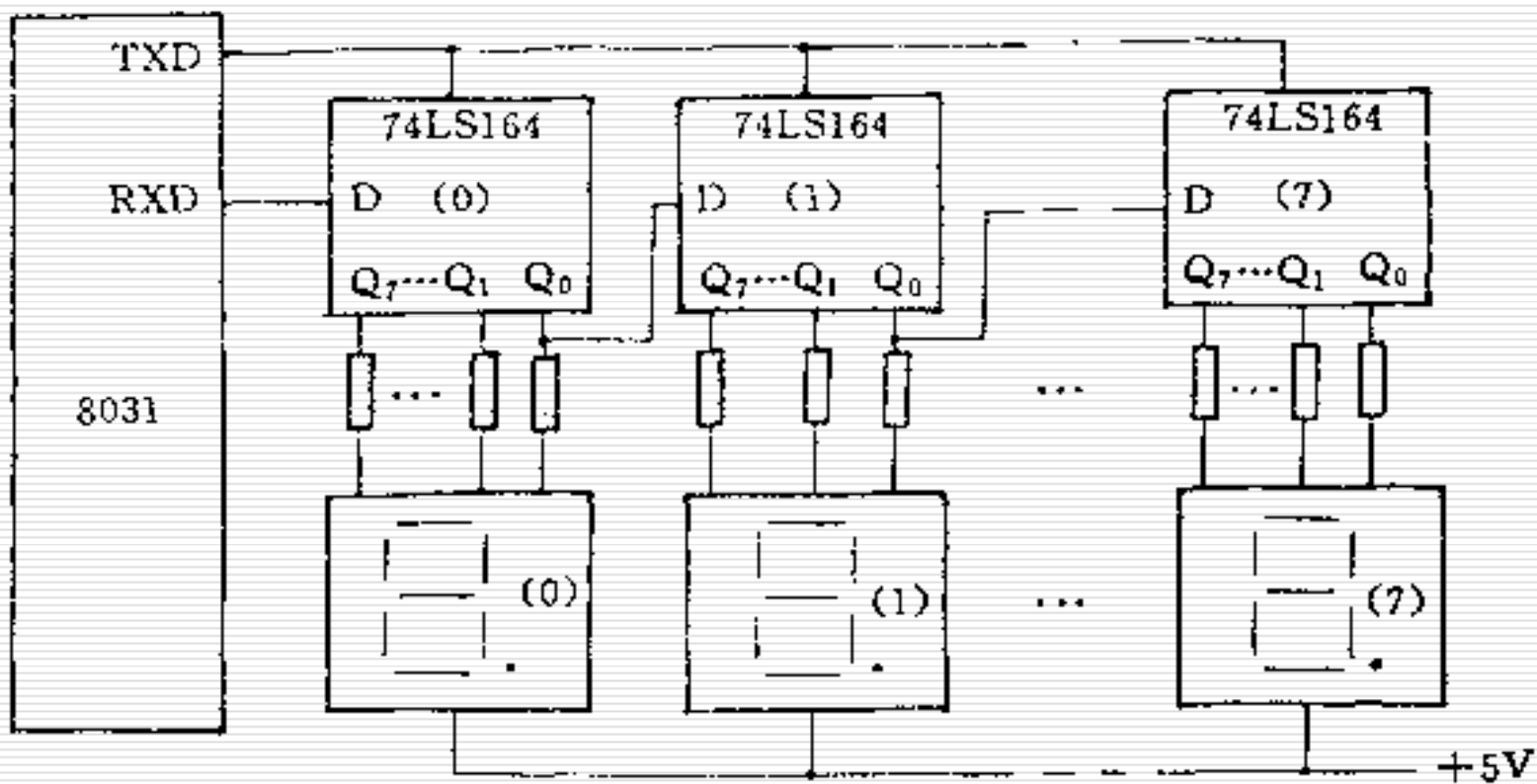
■ 动态

- ☐ 需要软件刷新扫描

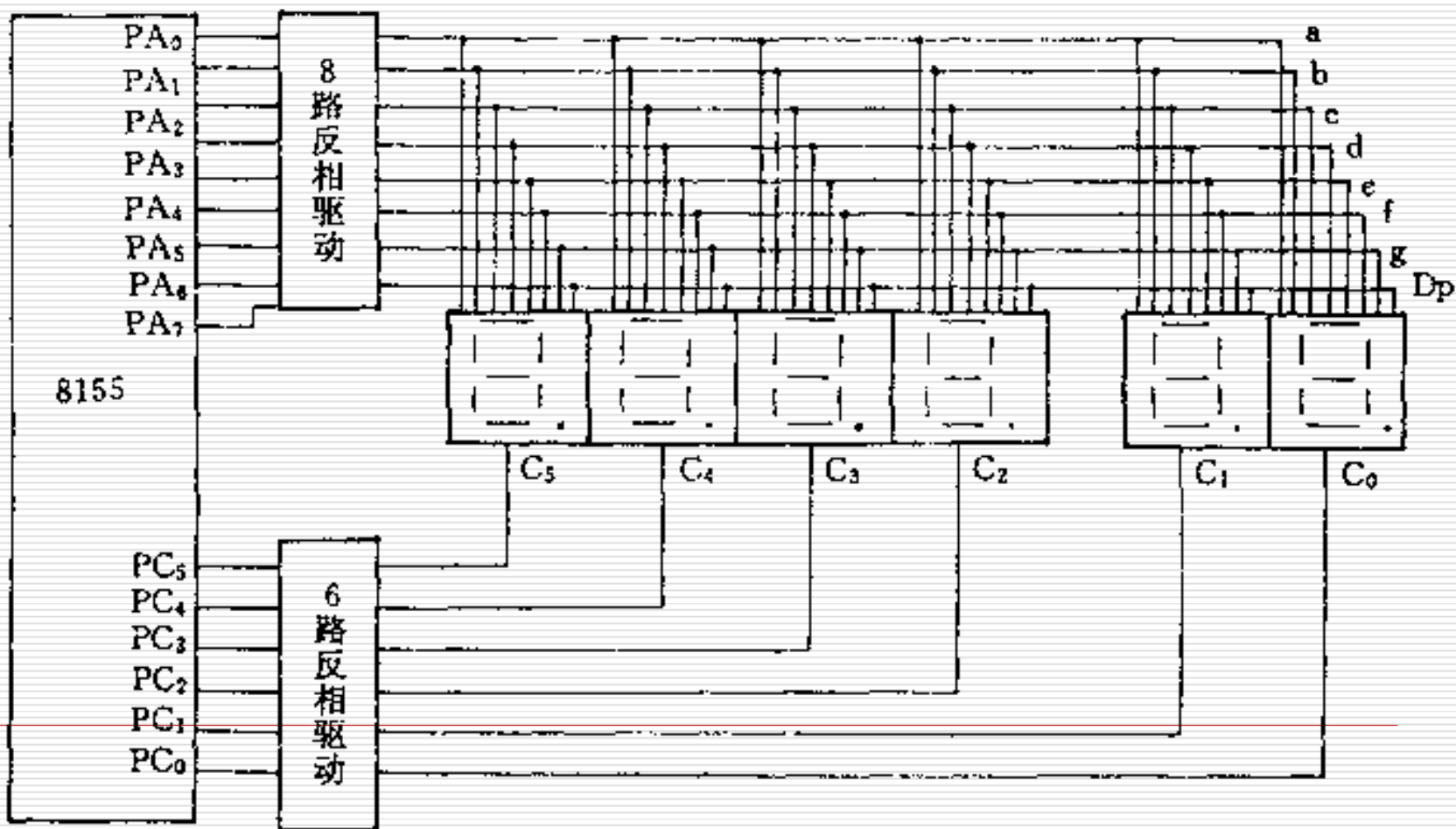
- ☐ 刷新操作消耗功率

■ 综合计算，选择方案

LED的静态显示接口



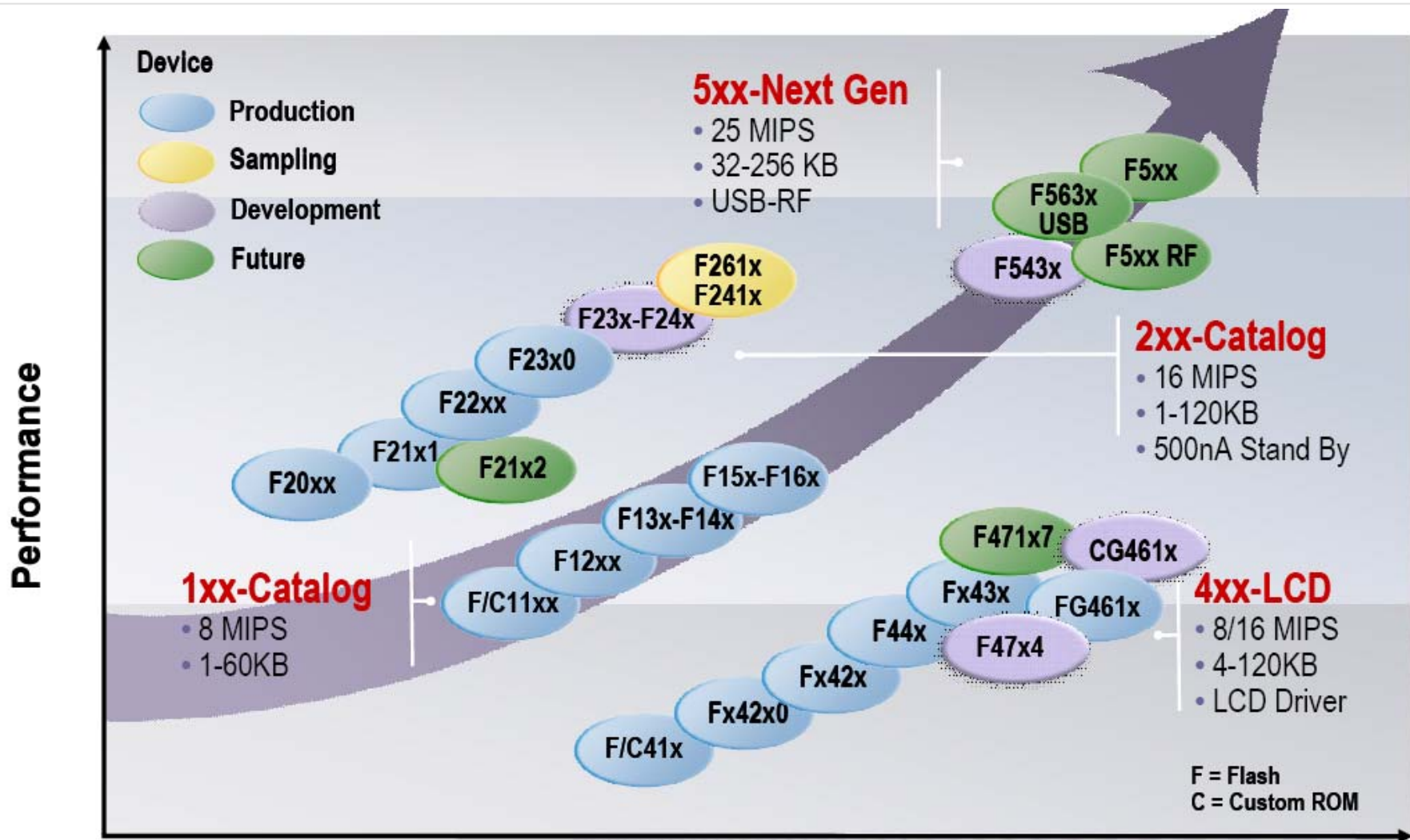
LED动态显示接口



主要内容

- 低功耗设计的概念
 - 嵌入式系统的低功耗设计
 - MSP430系统的低功耗设计
-

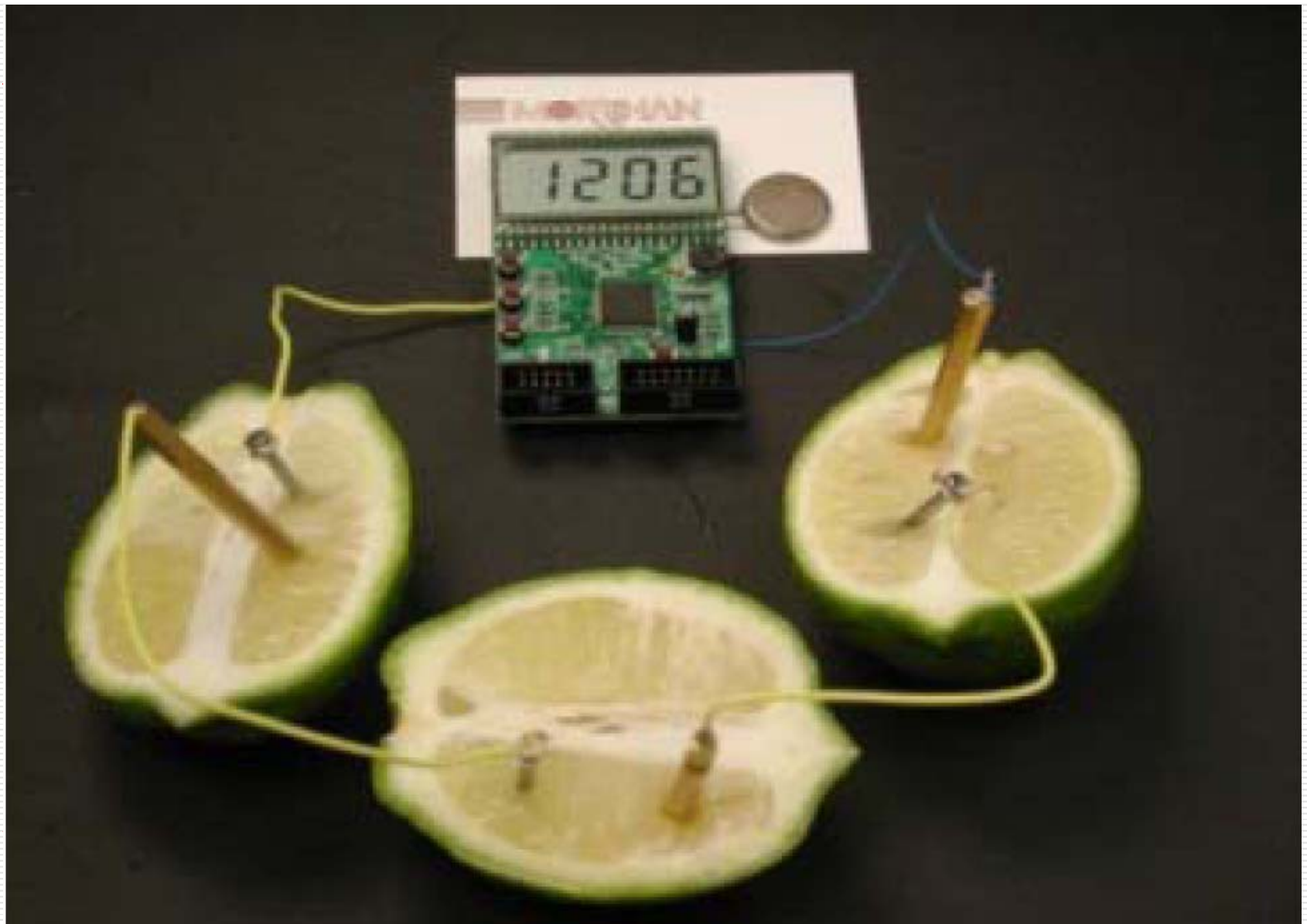
MSP430的发展历程



MSP430系列单片机的重要特性

- ❑ 超低功耗
- ❑ 强大的处理能力
- ❑ 高性能模拟技术及丰富的片上外围模块
- ❑ 系统工作稳定
- ❑ 方便高效的开发环境

特性1-超低功耗



MSP430的低功耗参数

- ❑ 0.1 μ A掉电模式
 - ❑ 0.8 μ A待机模式
 - ❑ 250 μ A/1MIPS
 - ❑ <50nA的端口漏电流
-

超低功耗的重要性

- 延长电池寿命
- 更小型的产品
- 更简单的电源
- 降低**EMI** 简化**PCB**
- 减轻负担



MSP430的超低功耗特性

□ 时钟系统

□ 工作模式

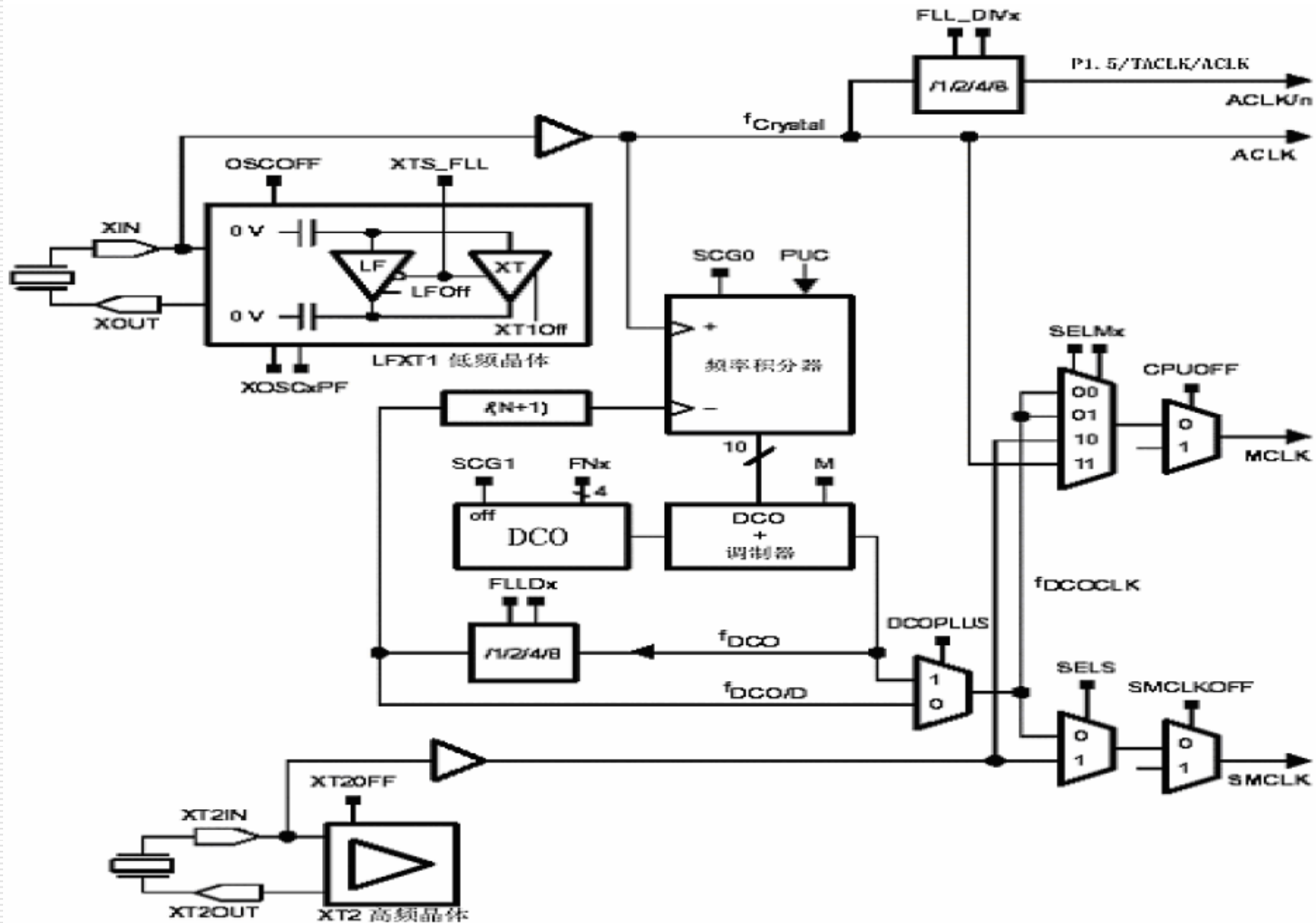
□ 中断能力

□ 外围模块

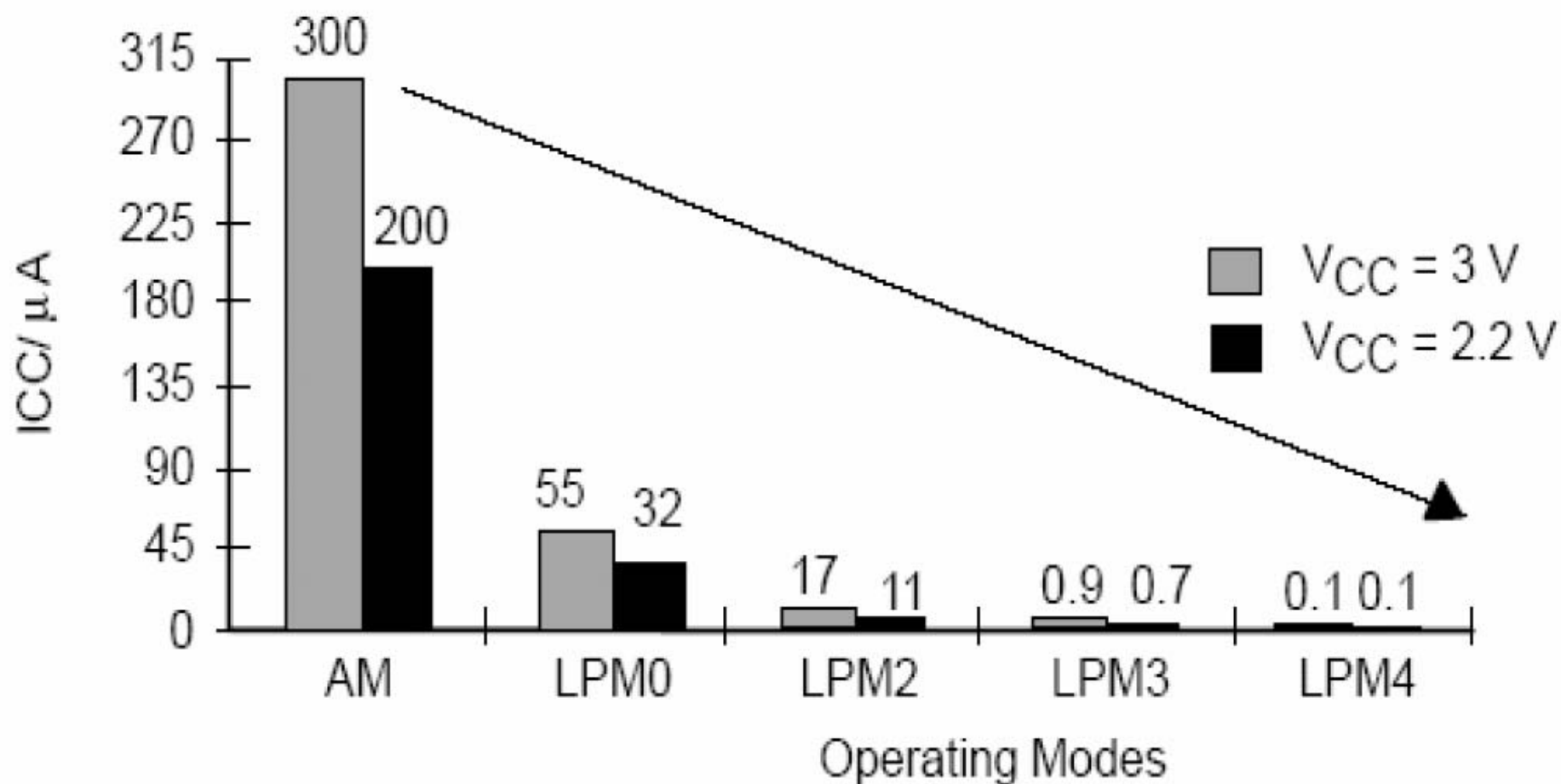
超低功耗—时钟系统

- 保持打开的低频ACLK
 - 根据需要启动的DCO
 - DCO从启动到稳定 $<1\mu\text{S}$
-

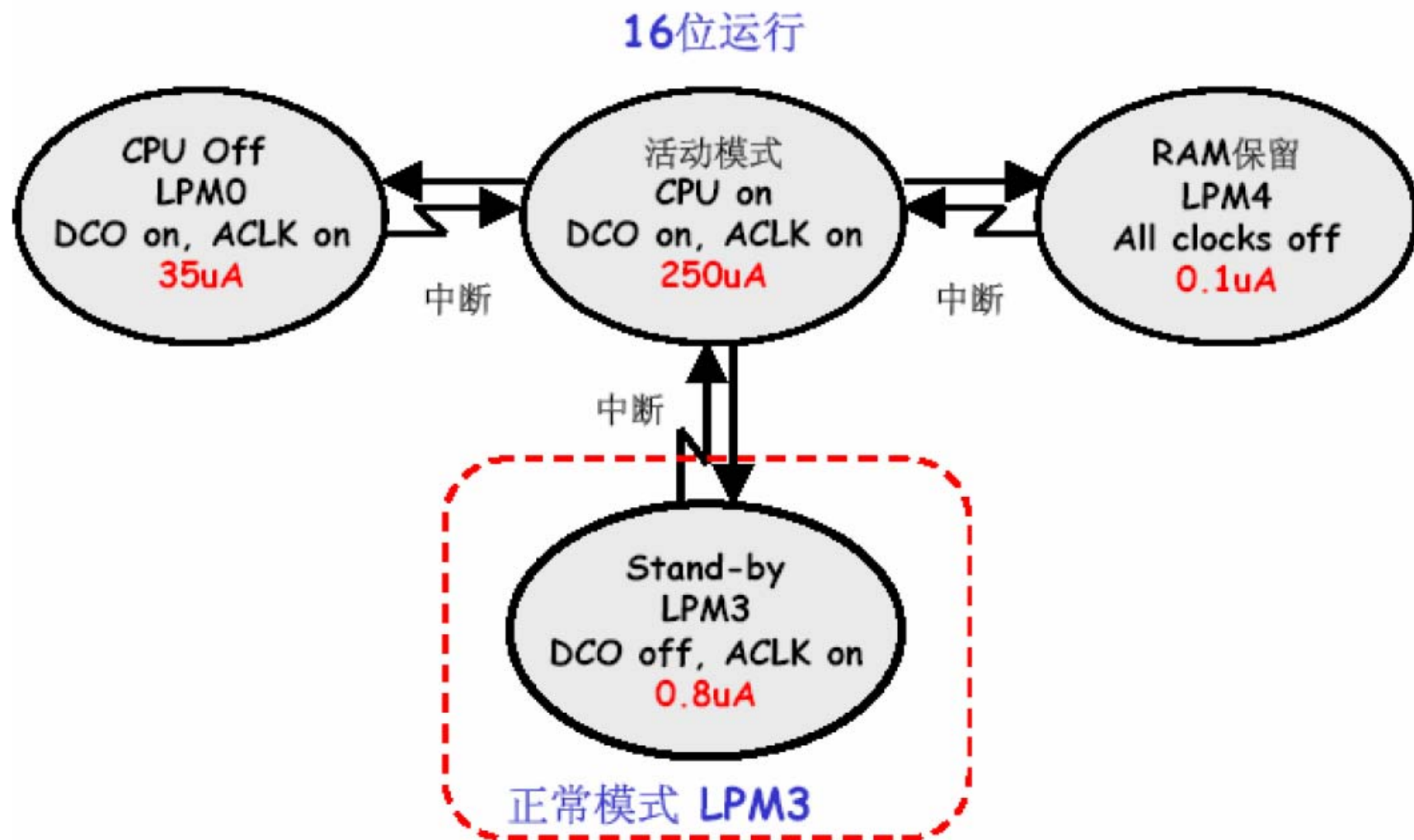
超低功耗—时钟系统



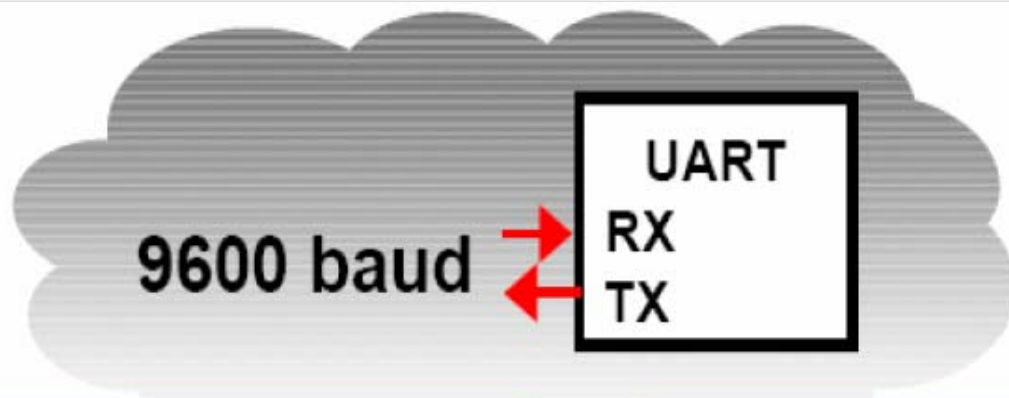
超低功耗—工作模式



超低功耗—中断能力



超低功耗—中断能力



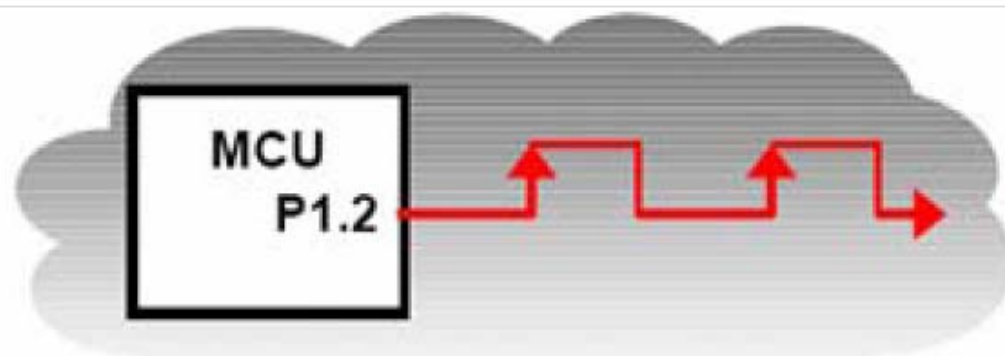
```
// Polling UART Receive  
for (;;)   
{  
    while (!(IFG2&URXIFG0));  
    TXBUF0 = RXBUF0;  
}
```

100% CPU Load

```
// UART Receive Interrupt  
#pragma vector=UART_VECTOR  
__interrupt void rx (void)  
{  
    TXBUF0 = RXBUF0;  
}
```

0.1% CPU Load

超低功耗—中断能力



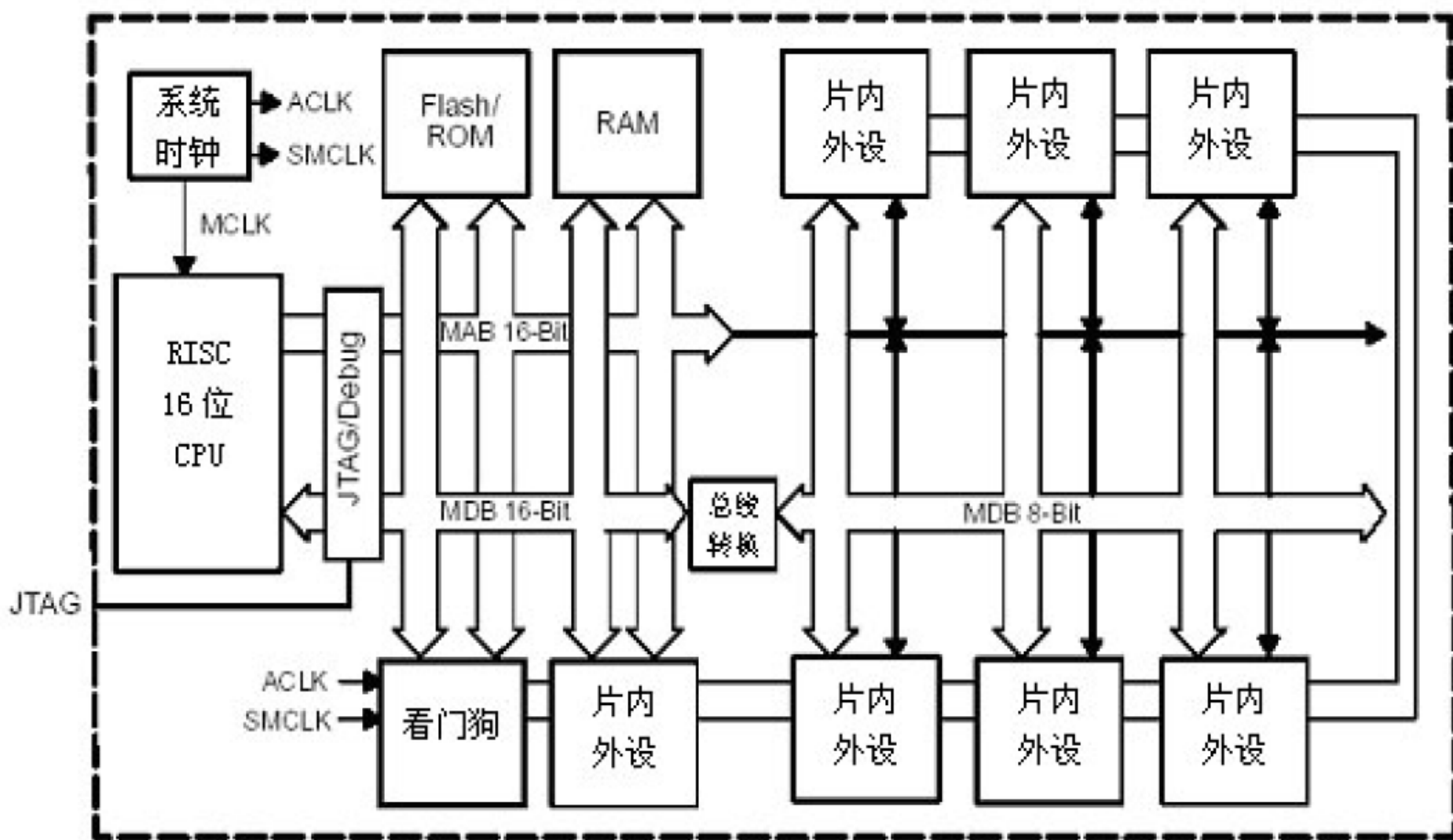
```
// Endless Loop
for (;;)
{
    P1OUT |= 0x04; // Set
    delay1();
    P1OUT &= ~0x04; // Reset
    delay2();
}
```

100% CPU Load

```
// Setup output unit
CCTL1 = OUTMOD0_1;
_BIS_SR(CPUOFF);
```

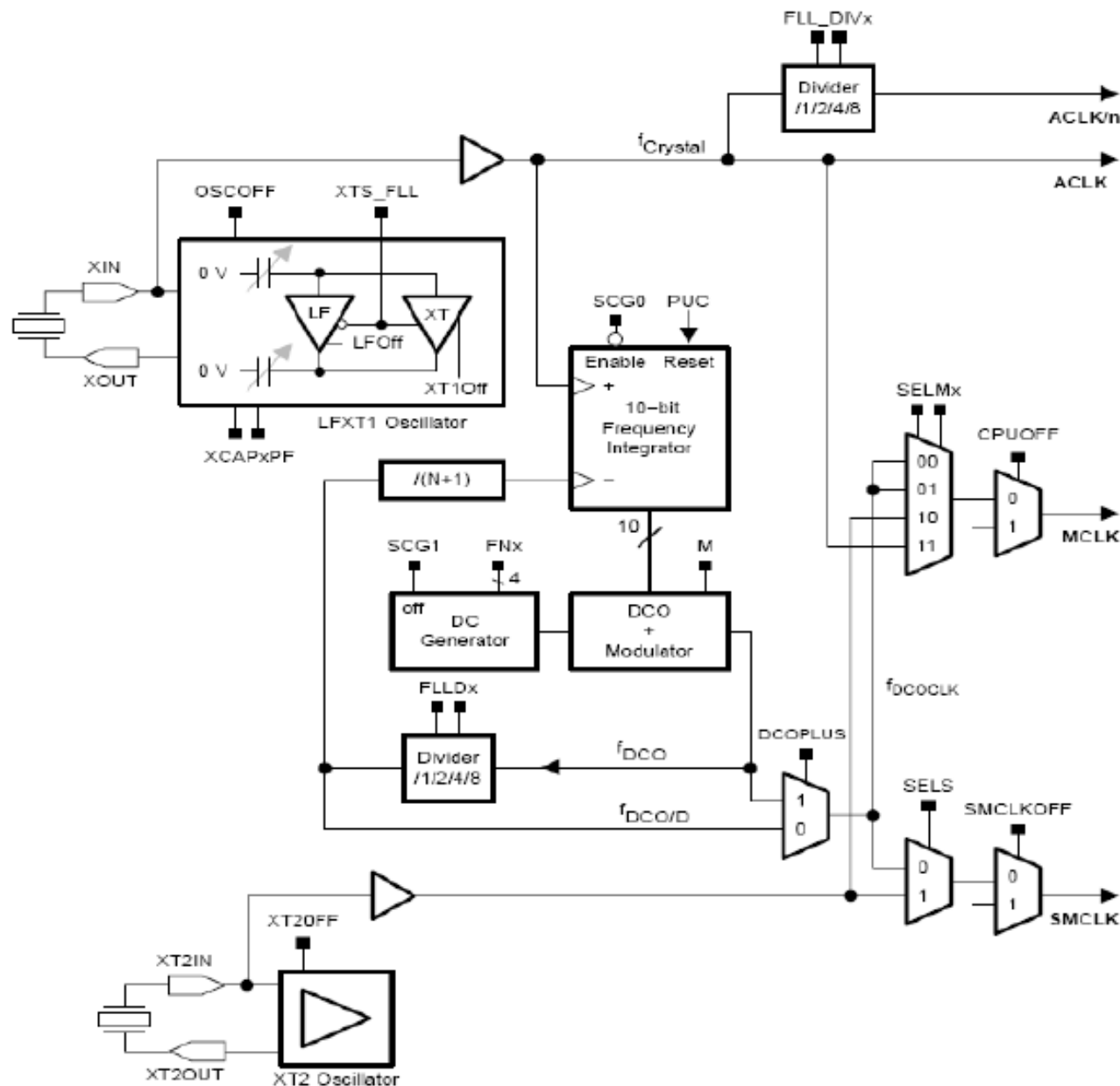
Zero CPU Load

超低功耗—外围模块



MSP430F4XX系列时钟模块

1. 三个时钟源
2. 内部DCO, FLL
3. 时钟1/2/4/8分频系数可选



MSP430F4XX时钟寄存器

- ❑ 系统时钟控制寄存器SCTQCTL
 - ❑ 系统时钟频率积分寄存器0 SCFI0
 - ❑ 系统时钟频率积分寄存器1 SCFI1
 - ❑ FLL+控制寄存器0 FLL+CTL0
 - ❑ FLL+控制寄存器1 FLL+CTL1
-

FLL+模块应用举例

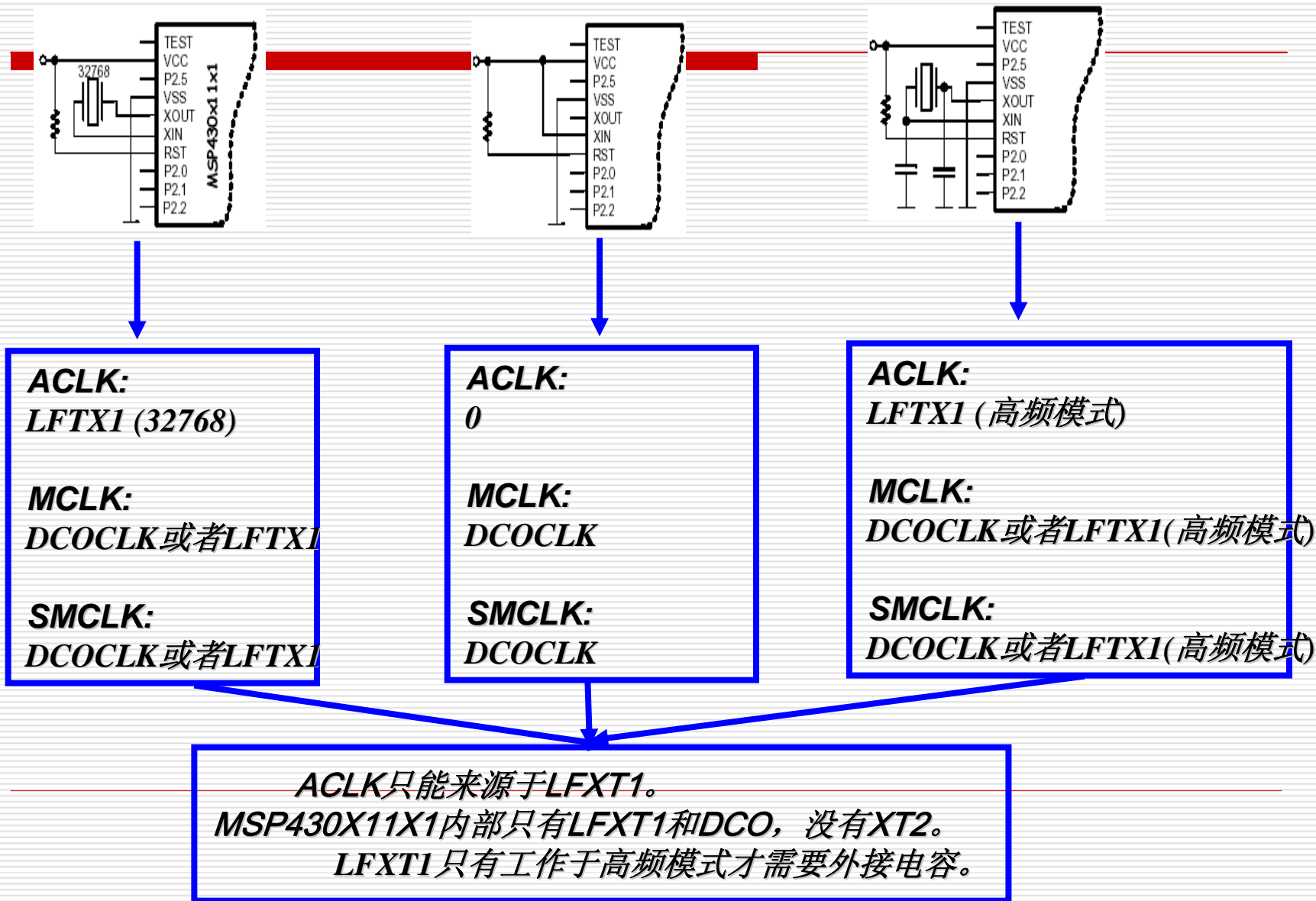
例1 设: $ACLK = LFXT1 = 32768\text{Hz}$, 令 $MCLK = SMCLK = DCOCLK = (n+1) \times ACLK$, 并将MCLK和ACLK分别通过P1.1和P1.5输出。

程序代码如下

```
#include "msp430x44x.h"
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // 停止看门狗
    SCFI0 |= FN_2;
    FLL_CTL0 = XCAP18PF;
    SCFQCTL = 74; // (74+1) × 32768 = 2.45Mhz
    P1DIR = 0x22; // P1.1 & P1.5 输出
    P1SEL = 0x22; // P1.1 & P1.5输出 MCLK & ACLK
    while(1);
}
```

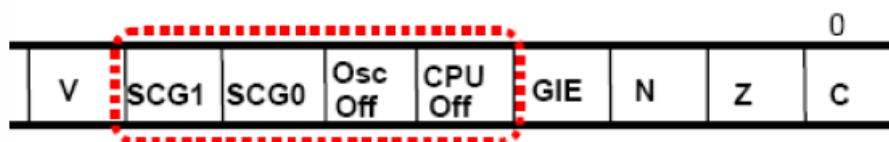
~~内部DCO = 2.45Mhz, P1.1--> MCLK = 2.45Mhz, P1.5--> ACLK = 32khz~~

根据实际连接情况，确定ACLK、SMCLK和MCLK时钟源。



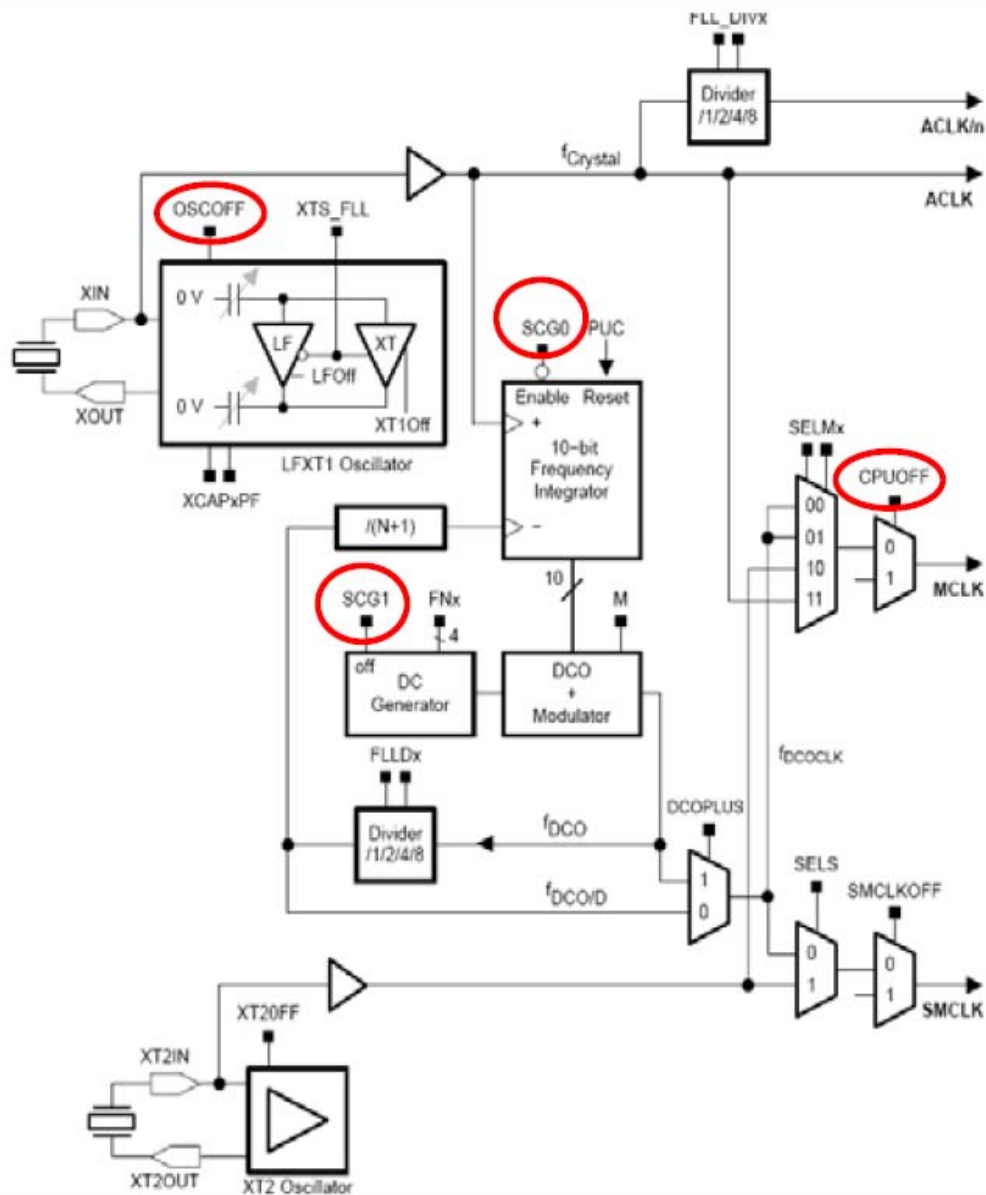
MCU工作模式

SR寄存器



各种工作模式

SCG1	SCG0	OSCOFF	CPUOFF	Mode
0	0	0	0	Active
0	0	0	1	LPM0
0	1	0	1	LPM1
1	0	0	1	LPM2
1	1	0	1	LPM3
1	1	1	1	LPM4




汇编模式下工作模式的转换

库文件定义:

<code>#define LPM0</code>	<code>(CPUOFF)</code>
<code>#define LPM1</code>	<code>(SCG0+CPUOFF)</code>
<code>#define LPM2</code>	<code>(SCG1+CPUOFF)</code>
<code>#define LPM3</code>	<code>(SCG1+SCG0+CPUOFF)</code>
<code>#define LPM4</code>	<code>(SCG1+SCG0+OSCOFF+CPUOFF)</code>

主程序进入模式: `BIS #LPM3, SR`

中断程序退出: `BIC #LPM3,0(SP)`



为什么不用`BIC #LPM3,SR`?

C语言下工作模式的转换

库文件定义:

```
#define LPM0_bits      (CPUOFF)
#define LPM1_bits      (SCG0+CPUOFF)
#define LPM2_bits      (SCG1+CPUOFF)
#define LPM3_bits      (SCG1+SCG0+CPUOFF)
#define LPM4_bits      (SCG1+SCG0+OSCOFF+CPUOFF)
```

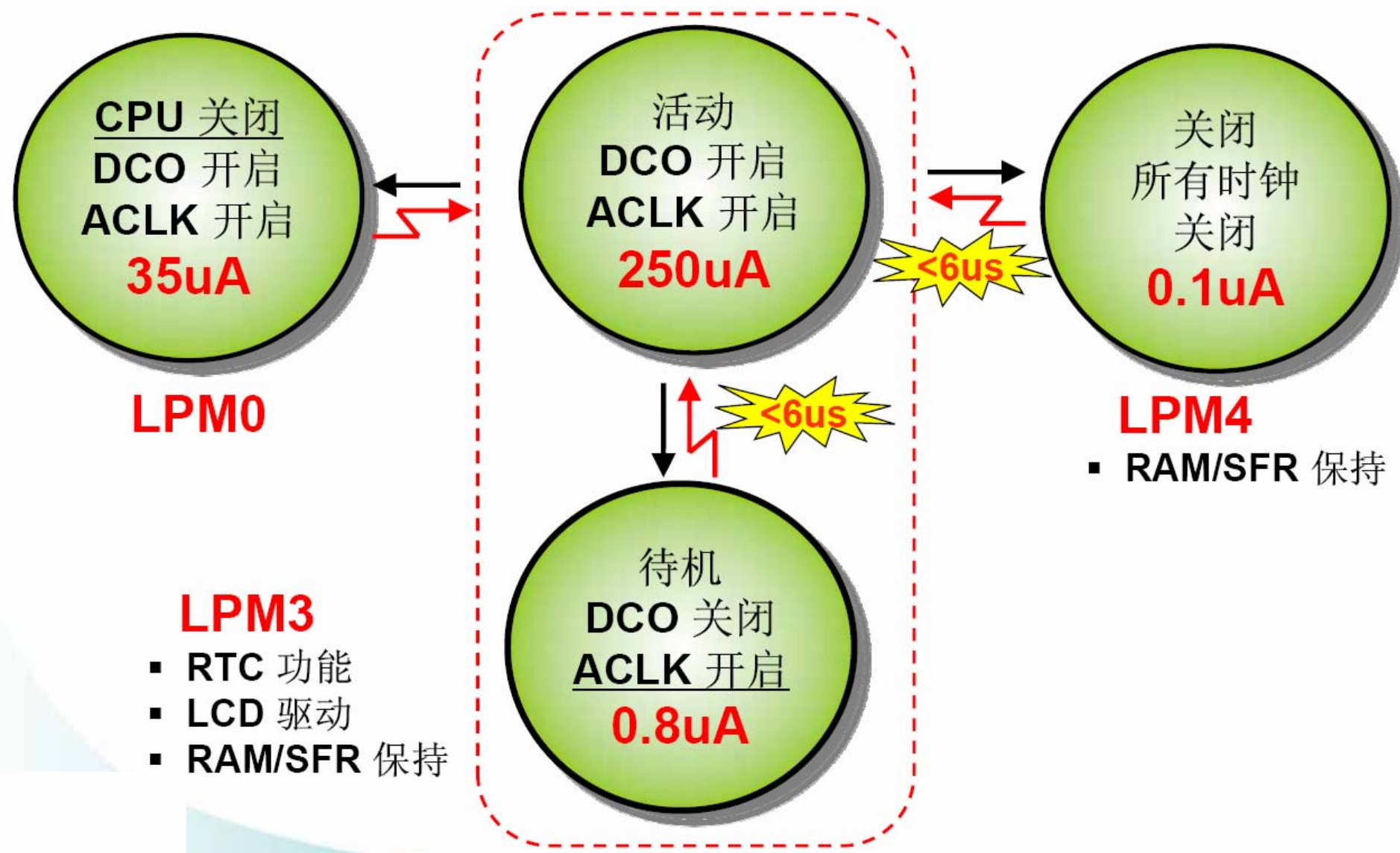
C语言操作:

```
#define LPM0      _BIS_SR(LPM0_bits)
#define LPM0_EXIT _BIC_SR_IRQ(LPM0_bits)
#define LPM1      _BIS_SR(LPM1_bits)
#define LPM1_EXIT _BIC_SR_IRQ(LPM1_bits)
#define LPM2      _BIS_SR(LPM2_bits)
#define LPM2_EXIT _BIC_SR_IRQ(LPM2_bits)
#define LPM3      _BIS_SR(LPM3_bits)
#define LPM3_EXIT _BIC_SR_IRQ(LPM3_bits)
#define LPM4      _BIS_SR(LPM4_bits)
#define LPM4_EXIT _BIC_SR_IRQ(LPM4_bits)
```

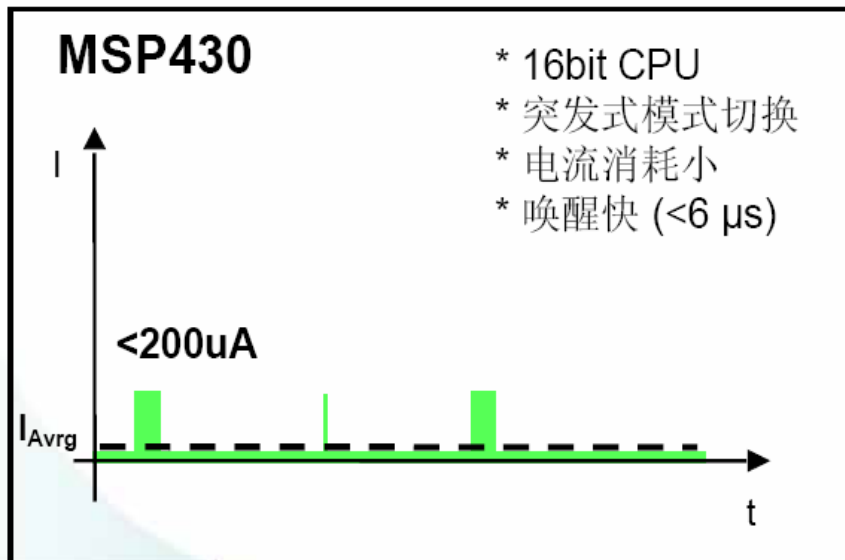
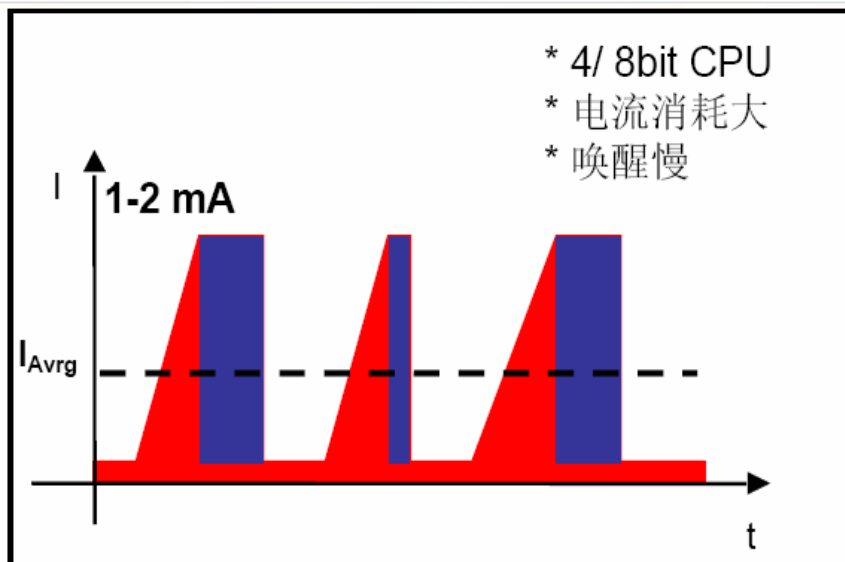
进入模式: LPM3 或 `_BIS_SR(LPM0_bits)` 或 `_BIS_SR(LPM0_bits + GIE)`

退出模式: `LPM3_EXIT` 或 `_BIC_SR_IRQ(LPM3_bits)`

各种模式下的转换



各种模式下的功耗



$I_{CC} / \mu A$

450

400

350

300

250

200

150

100

50

0

200

32

32

11

0.7

0.1

$V_{CC} = 2.2V$

1 μsec 周期

Active
Mode

LPM0

LPM1

LPM2

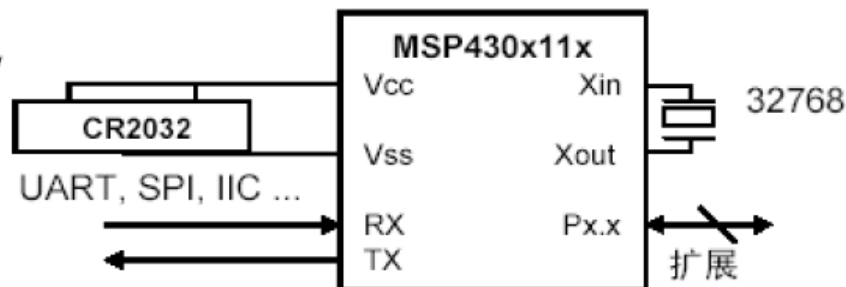
LPM3

LPM4

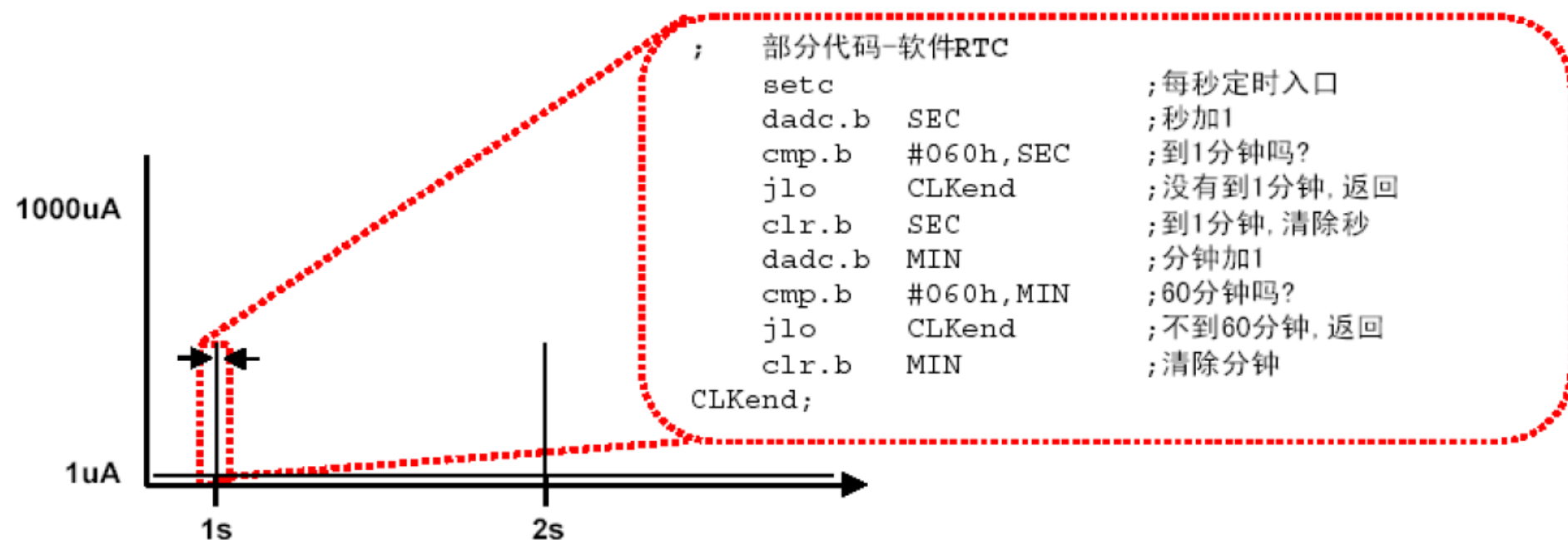
低功耗模式下功耗的计算

❑ 1.50uA - 正常的工作模式用LPM3 @3V

❑ 0.03uA - RTC 更新 (< 100us)



❑ 1.53uA - 平均电流 = $1.50\mu\text{A} + \frac{100}{1000000} * 250\mu\text{A}$



在LPM3模式中, 最大启动时间 $DCO < 6\mu\text{s}$

MSP430的低功耗原则(1)

一般的低功耗原则：

最大化LPM3的时间，用32KHz晶振作为ACLK时钟，

DCO用于CPU激活后的突发短暂运行

用接口模块代替软件驱动功能。

用中断控制程序运行

用可计算的分支代替标志位测试产生的分支

用快速查表代替冗长的软件计算

在冗长的软件计算中使用单周期的CPU寄存器

避免频繁的子程序和函数调用

尽可能直接用电池供电

MSP430的低功耗原则 (2)

□ 设计外设时的常规原则:

将不用的FETI输入端连接到VSS

JTAG端口TMS、TCK和TDI不要连接到VSS

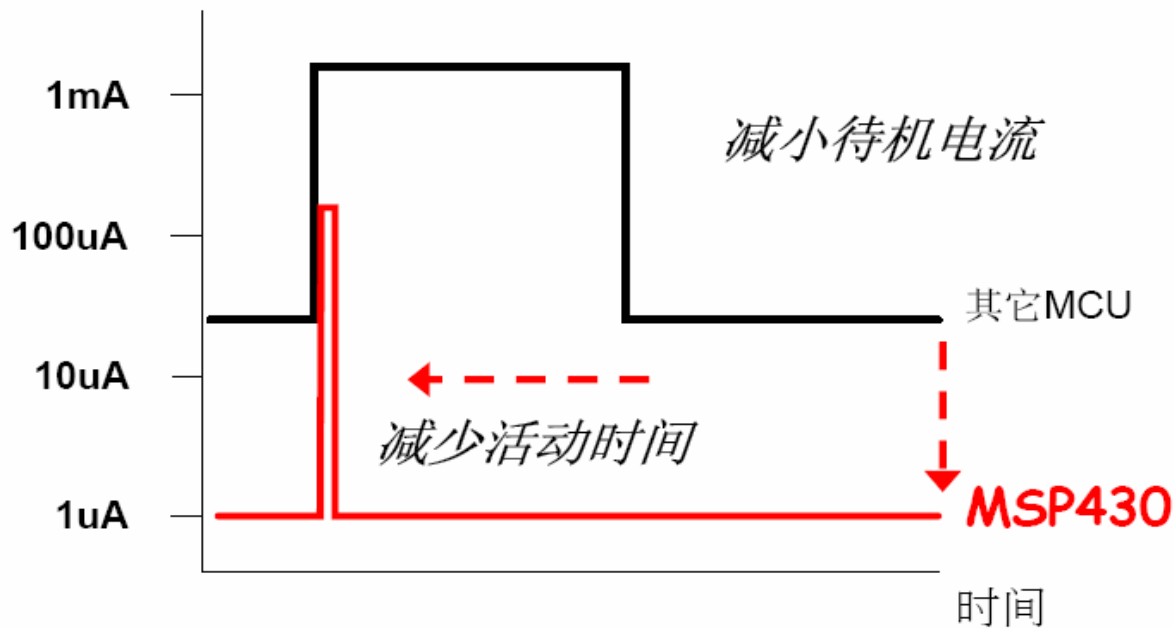
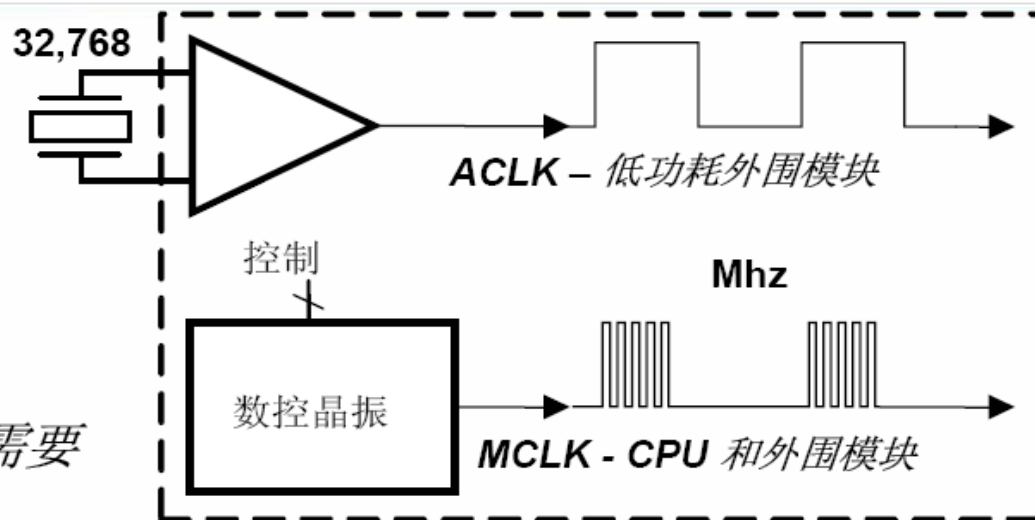
CMOS输入端不能有浮空节点，将所有输入端接适当的电平

不论对于内核还是对于各外围模块，选择尽可能低的运行频率，如果不影响功能应设计自动关机

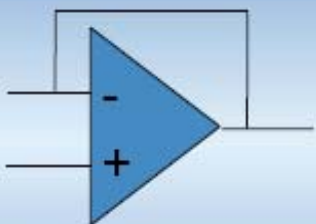
一般的设计考虑

☒ 总是开启

☒ 根据需要



系统外围器件的低功耗



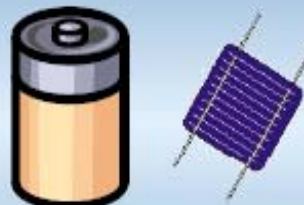
放大器

- ☐ OPA333
- ☐ OPA334



低功耗无线

- ☐ CC1101
- ☐ CC2500
- ☐ CC2500



电源管理

- TPS61200

**RS-485 / RS-232**

- ☐ MAX3222E
- ☐ SN65HVD33



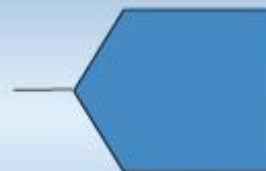
温度传感器

- TMP275



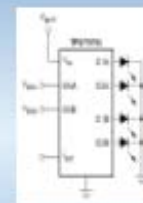
参考

- REF33XX
(Preview)



数据转换

- ☐ ADS1225
- ☐ ADS8326
- ☐ DAC8560



LED 背光

- TPS7510X

MSP430F24X的低功耗示例

```
#include "msp430x24x.h"
void main(void)
{
    BCSCTL1 |= DIVA_2;           // ACLK/4 = 8.192 kHz
    WDTCTL = WDT_ADLY_1000;      // WDT-ALCK Delay: 1s*4 interval timer
                                // interval period= 4sec=1/4Hz(freq)
    IE1 |= WDTIE;                // Enable WDT interrupt
    P1DIR = 0xFF;                // All P1.x outputs
    P1OUT = 0;                   // All P1.x reset
    while(1)
    {
        __bis_SR_register(LPM3_bits + GIE); // Enter LPM3, enable interrupts
        P1OUT ^= 0x01;           // Set P1.0 LED on
    }
}

#pragma vector = WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    //Wake up the system
    __bic_SR_register_on_exit(LPM3_bits); // Clear LPM3 bits from 0(SR)
}
```

MSP430F24X IO端口的低功耗示例

```
❑ #include "msp430x24x.h"

❑ void main(void)
❑ {
❑     WDTCTL = WDTPW + WDT HOLD;           // Stop watchdog timer
❑     P1DIR |= 0x01;                       // Set P1.0 to output direction
❑     P1IE |= 0x08;                        // P1.3 interrupt enabled
❑     P1IES |= 0x08;                       // P1.3 Hi/lo edge
❑     P1IFG &= ~0x08;                     // P1.3 IFG cleared

❑     __bis_SR_register(LPM4_bits + GIE);  // Enter LPM4 w/interrupt
❑ }

❑ // Port 1 interrupt service routine
❑ #pragma vector=PORT1_VECTOR
❑ __interrupt void Port_1(void)
❑ {
❑     P1OUT ^= 0x01;                       // P1.0 = toggle
❑     P1IFG &= ~0x08;                     // P1.3 IFG cleared
❑ }
```


MSP430F24X WDT的低功耗示例

```
❑ #include "msp430x24x.h"  
  
❑ void main(void)  
❑ {  
❑     P1DIR |= 0x01;           // Set P1.0 to output  
❑     P1OUT ^= 0x01;           // Toggle P1.0  
❑     __bis_SR_register(LPM4_bits + GIE);  
        // Stop all clocks  
❑ }
```

MSP430F24X UART的低功耗示例(1)

```
❑ #include "msp430x24x.h"

❑ void main(void)
❑ {
❑     WDTCTL = WDTPW + WDT HOLD;           // Stop WDT
❑     if (CALBC1_1MHZ == 0xFF || CALDCO_1MHZ == 0xFF)
❑     {
❑         while(1);           // If calibration constants erased do not load, trap CPU!!
❑     }
❑     BCSCTL1 = CALBC1_1MHZ;           // Set DCO
❑     DCOCTL = CALDCO_1MHZ;
❑
❑     P1DIR = 0xFF;           // All P1.x outputs
❑     P1OUT = 0;           // All P1.x reset
❑     P2SEL = 0x02;           // P2.1 = SMCLK, others GPIO
❑     P2DIR = 0xFF;           // All P2.x outputs
❑     P2OUT = 0;           // All P2.x reset
❑     P3SEL = 0x30;           // P3.4,5 = USCI_A0 TXD/RXD
❑     P3DIR = 0xFF;           // All P3.x outputs
❑     P3OUT = 0;           // All P3.x reset
❑
```

MSP430F24X UART的低功耗示例(2)

```
□ UCA0CTL1 |= UCSSEL_2;           // SMCLK
□ UCA0BR0 = 8;                     // 1MHz 115200
□ UCA0BR1 = 0;                     // 1MHz 115200
□ UCA0MCTL = UCBRS2 + UCBRS0;      // Modulation UCBRSx = 5
□ UCA0CTL1 &= ~UCSWRST;            // **Initialize USCI state machine**
□ IE2 |= UCA0RXIE;                // Enable USCI_A0 RX interrupt

□ __bis_SR_register(LPM4_bits + GIE); // Enter LPM4, interrupts enabled
□ }

□ // Echo back RXed character, confirm TX buffer is ready first
□ #pragma vector=USCIAB0RX_VECTOR
□ __interrupt void USCI0RX_ISR(void)
□ {
□     while (!(IFG2&UCA0TXIFG));    // USCI_A0 TX buffer ready?
□     UCA0TXBUF = UCA0RXBUF;        // TX -> RXed character
□ }
```

MSP430F24X TimerA的低功耗示例

```
□ #include <msp430x24x.h>

□ void main(void)
□ {
□     WDTCTL = WDTPW + WDTHOLD;           // Stop WDT
□     P1DIR |= 0x01;                      // P1.0 output
□     TACTL = TASSEL_1 + MC_2 + TAIE;      // ACLK, contmode, interrupt

□     _BIS_SR(LPM3_bits + GIE);           // Enter LPM3 w/ interrupt
□ }

□ // Timer_A3 Interrupt Vector (TAIV) handler
□ #pragma vector=TIMERA1_VECTOR
□ __interrupt void Timer_A(void)
□ {
□     switch( TAIV )
□     {
□         case 2: break;                  // CCR1 not used
□         case 4: break;                  // CCR2 not used
□         case 10: P1OUT ^= 0x01;         // overflow
□             break;
□     }
□ }
```

MSP430F24X ADC的低功耗示例(1)

```
❑ #include <msp430x24x.h>

❑ #define Num_of_Results 8

❑ volatile unsigned int A0results[Num_of_Results]; // These need to be global in
❑ volatile unsigned int A1results[Num_of_Results]; // this example. Otherwise, the
❑ volatile unsigned int A2results[Num_of_Results]; // compiler removes them because
❑ volatile unsigned int A3results[Num_of_Results]; // they are not used

❑ void main(void)
❑ {
❑     WDTCTL = WDTPW+WDTHOLD;           // Stop watchdog timer
❑     P6SEL = 0x0F;                     // Enable A/D channel inputs
❑     ADC12CTL0 = ADC12ON+MSC+SHT0_8;   // Turn on ADC12, extend sampling time
❑                                     // to avoid overflow of results
❑     ADC12CTL1 = SHP+CONSEQ_3;         // Use sampling timer, repeated sequence
❑     ADC12MCTL0 = INCH_0;              // ref+=AVcc, channel = A0
❑     ADC12MCTL1 = INCH_1;              // ref+=AVcc, channel = A1
❑     ADC12MCTL2 = INCH_2;              // ref+=AVcc, channel = A2
❑     ADC12MCTL3 = INCH_3+EOS;          // ref+=AVcc, channel = A3, end seq.
❑     ADC12IE = 0x08;                   // Enable ADC12IFG.3
❑     ADC12CTL0 |= ENC;                  // Enable conversions
❑     ADC12CTL0 |= ADC12SC;              // Start convn - software trigger
❑     _BIS_SR(LPM0_bits + GIE);         // Enter LPM0, Enable interrupts
❑ }
```

MSP430F24X ADC的低功耗示例(2)

- ☐
- ☐ #pragma vector=ADC12_VECTOR
- ☐ ~~__interrupt void ADC12ISR (void)~~
- ☐ {
- ☐ static unsigned int index = 0;

```

☐ A0results[index] = ADC12MEM0;           // Move A0 results,
IFG is cleared
☐ A1results[index] = ADC12MEM1;           // Move A1 results,
IFG is cleared
☐ A2results[index] = ADC12MEM2;           // Move A2 results,
IFG is cleared
☐ A3results[index] = ADC12MEM3;           // Move A3 results,
IFG is cleared
☐ index = (index+1)%Num_of_Results;           // Increment
results index, modulo; Set Breakpoint1 here
☐ if (index == 0)
☐     _NOP();                               // Set Breakpoint2 here
☐ }

```

主要内容

- 低功耗设计的概念
 - 嵌入式系统的低功耗设计
 - MSP430系统的低功耗设计
-

思考题

1,功耗产生的原因

2,如何降低系统的功耗

软件方面

硬件方面

3,分析:以手机为例,降低功耗从哪些方面入手,损失了哪些手机指标

4,MSP430单片机低功耗设计的基本原则及低功耗设计的方法

谢谢各位！