第6章 时钟系统和低功耗模式

第1节 有关概念介绍

一、 主频，外频， 倍频系数
二、 T状态
三、 总线周期
四、 指令周期

第2节 时钟系统模块

一、时钟系统模块结构图
二、时钟系统模块相关寄存器
三、时钟系统模块输出引脚
四、振荡器失效检测
五、基本时钟设置举例

第3节 低功耗模式

一、低功耗控制
二、MSP430工作模式
三、低功耗模式的进入与退出
四、低功耗模式编程举例

第1节　有关概念介绍

一、 主频，外频，倍频系数

二、 T状态

三、 总线周期

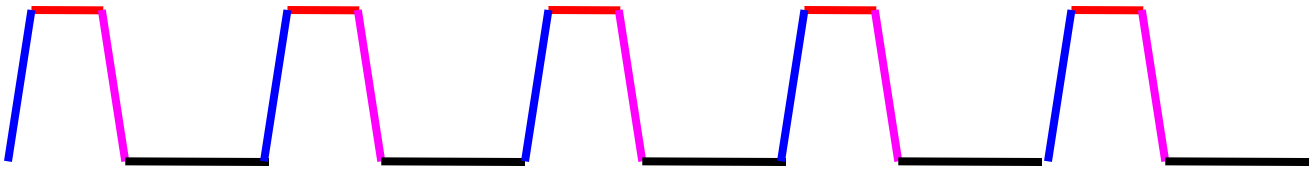
四、 指令周期

一、主频，外频，倍频系数

● CPU是在时钟信号的控制下工作

没有时钟信号或时钟信号不正常，CPU将不能正常工作！

CLK



时钟信号是一个按一定电压幅度、一定时间间隔发出的脉冲信号

频率f：1秒内的脉冲个数

周期 T = 1/ f

f	32.768KHz	8MHz	133MHz
T	30.5ms	125ns	7.5ns

占空比：高电平在一个周期中的比例

● CPU所有的操作都以时钟信号为基准

CPU 按严格的时间标准发出地址，控制信号，

存储器、接口模块也按严格的时间标准送出或接收数据.

这个时间标准就是由时钟信号确定。

- CPU的主频或内频指CPU内部的工作频率。

  主频是表示CPU工作速度的重要指标，

  在 CPU其它性能指标相同时，主频越高，CPU 的速度越快

- CPU的外频或系统频率指CPU的外部总线频率。

  外频越高，微处理器与系统内存数据交换的速度越快，

  因而计算机的运行速度也越快。

- 倍频系数指CPU主频和外频的相对比例系数。

  8088/8086/80286/80386的主频和外频值相同;

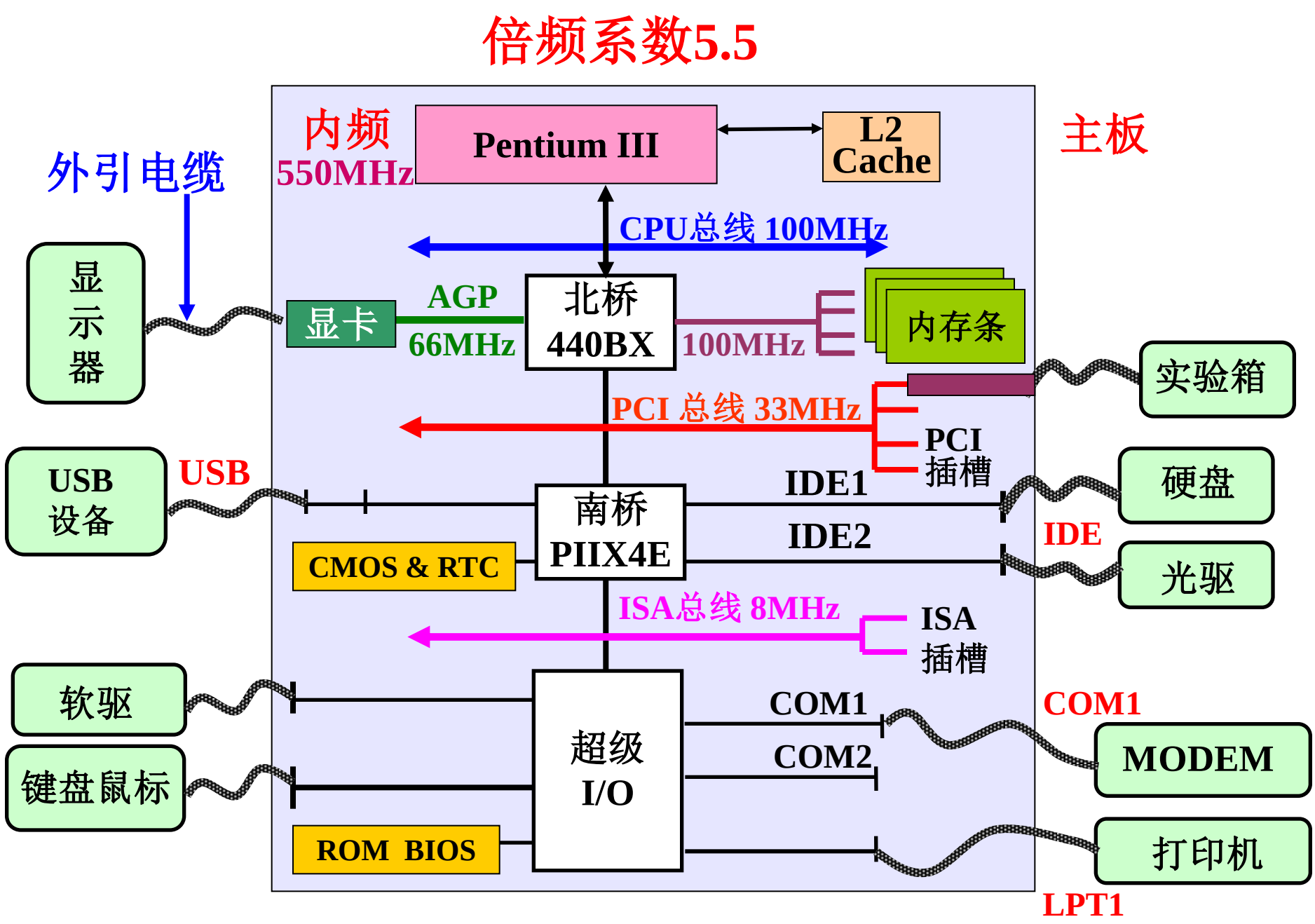
  从80486DX2开始，CPU的主频和外频不再相同，

  将外频按一定的比例倍频后得到CPU的主频，即：

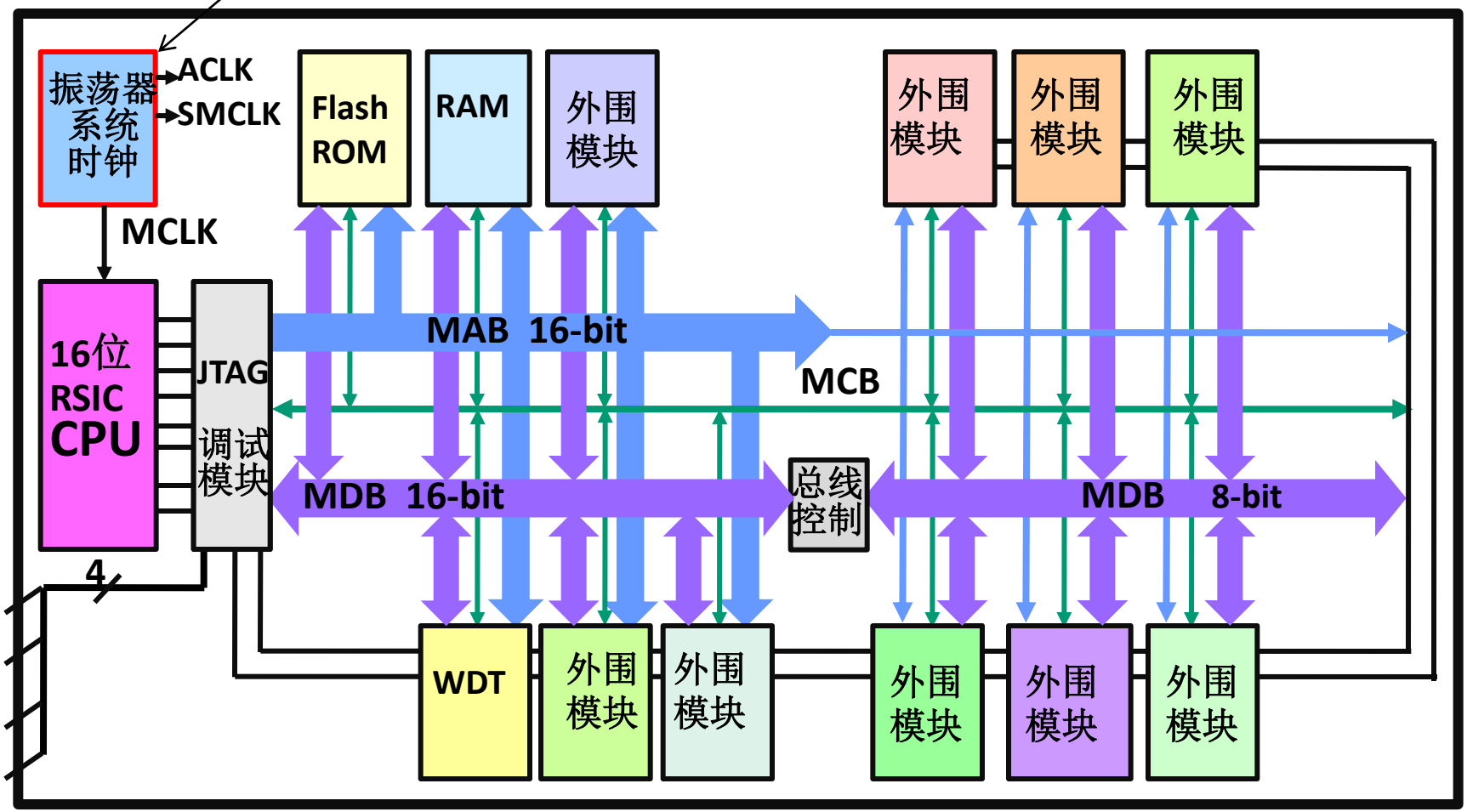
  CPU主频 = 外频 × 倍频系数

  现在PC机外频一般是200～333MHz，倍频是6~11

- PC机各子系统时钟(存储系统，显示系统，总线等)是由系统频率按照一定的比例分频得到。


倍频系数5.5
内频
550MHz
Pentium III
L2
Cache
主板
外引电缆
CPU总线 100MHz
显示器
显卡
AGP
66MHz
北桥
440BX
100MHz
内存条
实验箱
PCI 总线 33MHz
PCI
插槽
USB
设备
USB
IDE1
硬盘
南桥
PIIX4E
IDE2
IDE
CMOS & RTC
光驱
ISA总线 8MHz
ISA
插槽
软驱
超级
I/O
COM1
COM1
键盘鼠标
COM2
MODEM
ROM BIOS
打印机
LPT1

MSP430F1xx基本时钟模块 Basic Clock Module

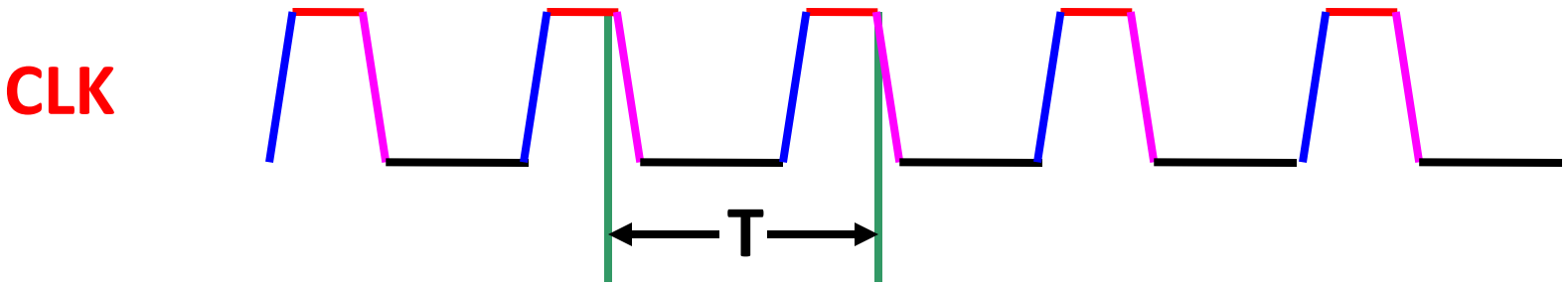


二、T状态

- 相邻两个脉冲之间的时间间隔，

  称为一个时钟周期，又称T状态（T周期）。

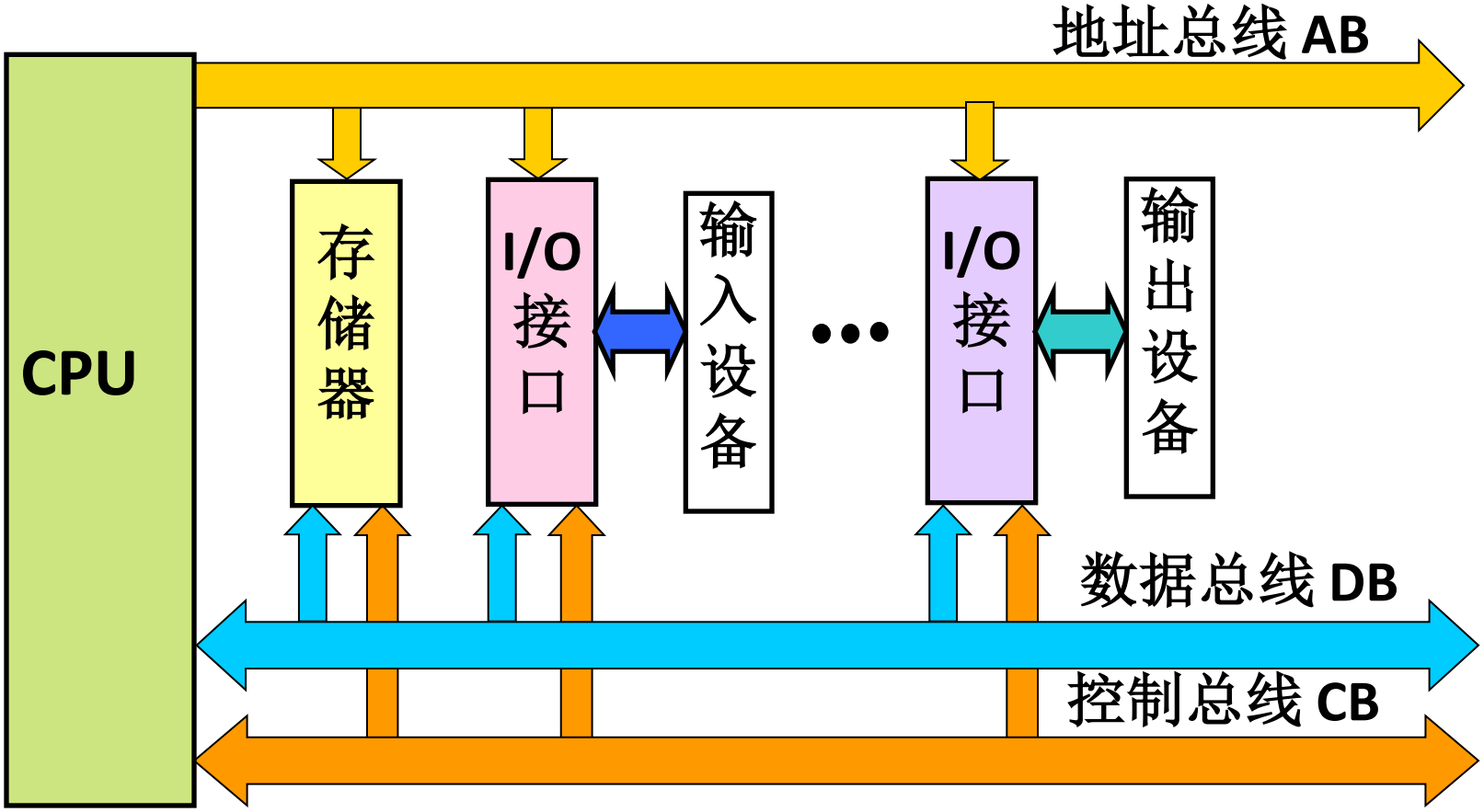
  时钟周期是CPU处理动作的最小时间单位。



- 每个T状态包括：下降沿、低电平、上升沿、高电平

三、总线周期

- CPU通过总线完成与存储器、I/O端口之间的操作，这些操作统称为总线操作。

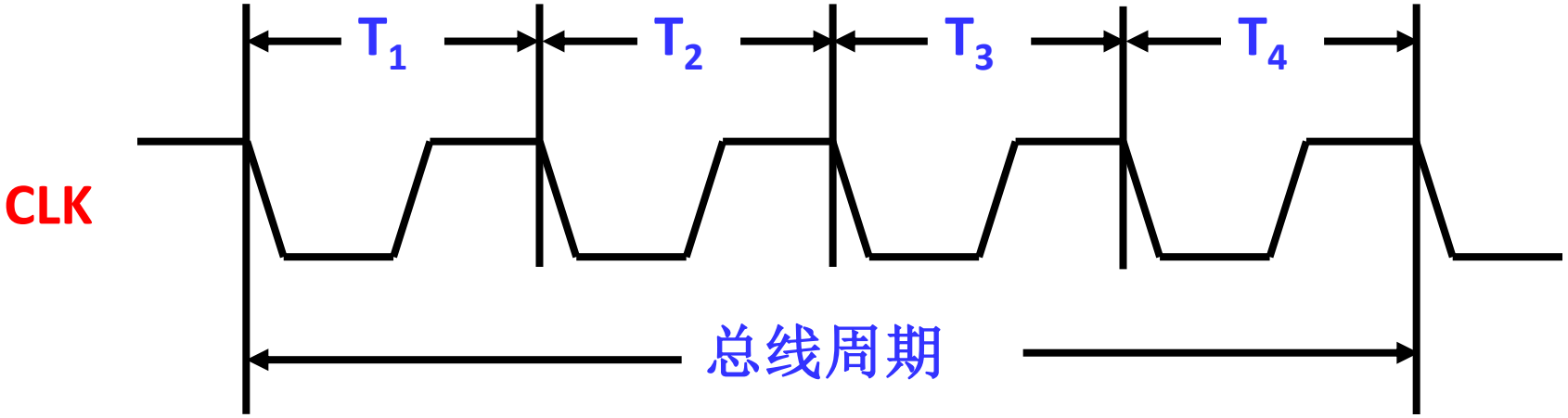


●执行一个总线操作所需要的时间称为总线周期。

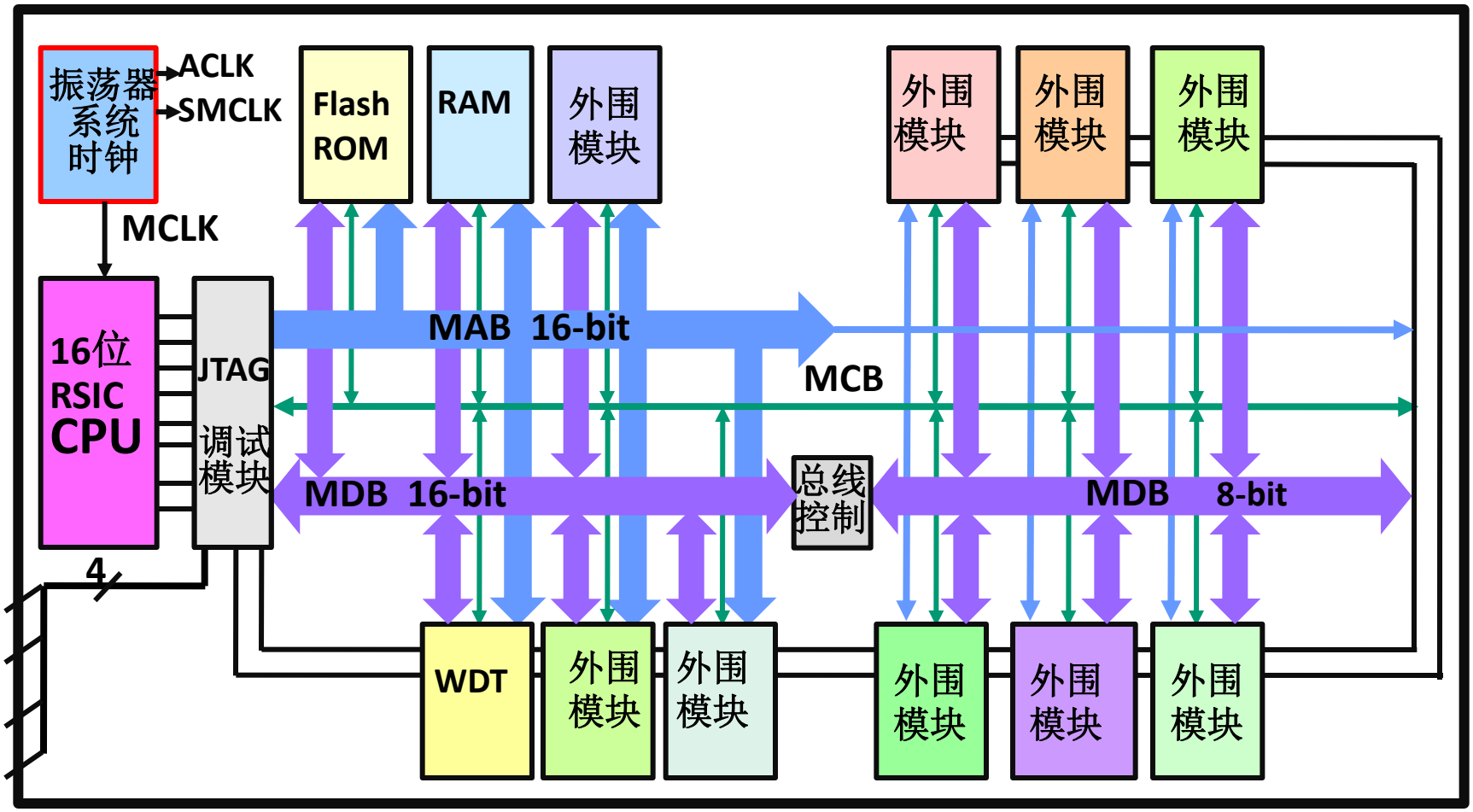
不同的总线操作有不同的总线周期。

总 线 操 作	总线周期
读存储器操作 (取指令、取操作数)	存储器读周期
写存储器操作 (将结果存放到内存)	存储器写周期
读 I/O 端口操作 (取 I/O 端口中的数)	I/O 端口读周期
写 I/O 端口操作 (往 I/O 端口写数)	I/O 端口写周期
中断响应操作	中断响应周期

- 一个基本的总线周期通常包含几个T状态，
  按时间的先后顺序分别称为T_1、T_2、T_3、T_4 ……



MSP430 单片机结构



- 各模块通过地址(MAB)、数据(MDB)和控制(MCB)三大总线互连

四、指令周期

- 执行一条指令所需要的时间称为指令周期。

  执行一条指令的时间:

  是取指令、执行指令、取操作数、存放结果所需时间的总和。

  用所需的时钟周期数表示。

例

MOV R4, R6	1个T周期
MOV 2(R5),R15	3个T周期
ADD 4(R5), 8(R15)	6个T周期

- 指令中操作数的寻址方式影响指令周期

MSP430 CPU各种寻址方式的指令周期和长度

MSP430x1xx Family User's Guide P3-73

源操作数	目的操作数	周期数	指令长度	例子
Rn	Rm 寄存器	1	1	MOV R5, R8
	PC 指令指针	2	1	BR R9
	x(Rm) 寄存器变址	4	2	ADD R5,4(R6)
	EDE 绝对地址	4	2	XOR R8, EDE
	&EDE 符号地址	4	2	MOV R5, &EDE
@Rn	Rm	2	1	AND @R4, R5
	PC	2	1	BR @R8
	x(Rm)	5	2	XOR @R5, 8(R6)
	EDE	5	2	MOV @R5, EDE
	&EDE	5	2	XOR @R5, &EDE
@Rn+	Rm	2	1	ADD @R5+ , R6
	PC	3	1	BR @R9+,
	x(Rm)	5	2	XOR @R5, 8(R6)
	EDE	5	2	MOV @R9+, EDE
	&EDE	5	2	MOV @R9+, &EDE

源操作数	目的操作数	周期数	指令长度	例子
#N	Rm	2	2	MOV #20, R9
	PC	3	2	BR #2AEh
	x(Rm)	5	3	MOV #300h, 0(SP)
	EDE	5	3	ADD #33, EDE
	&EDE	5	3	ADD #33, &EDE
x(Rn)	Rm	3	2	MOV 2(R5), R7
	PC	3	2	BR 2(R6)
	x(Rm)	6	3	ADD 4(R4), 6(R9)
	TONI	6	3	MOV 4(R7), TONI
	&EDE	6	3	MOV 2(R4), &TONI
EDE	Rm	3	2	AND EDE, R6
	PC	3	2	BR EDE
	x(Rm)	6	3	MOV EDE, 0(SP)
	TONI	6	3	CMP EDE, TONI
	&TONI	6	3	MOV EDE, &TONI
&EDE	Rm	3	2	MOV &EDE, R8
	PC	3	2	BR &EDE
	x(Rm)	6	3	MOV &EDE, 0(SP)
	TONI	6	3	MOV &EDE, TONI
	&TONI	6	3	MOV &EDE, &TONI

MSP430X CPU各种寻址方式的指令周期和长度

MSP430x6xx Family User's Guide

Table 6-9. MSP430 Format II Instruction Cycles and Length

Addressing Mode	No. of Cycles			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
Rn	1	3	4	1	`SWPB R5`
@Rn	3	3	4	1	`RRC @R9`
@Rn+	3	3	4	1	`SWPB @R10+`
#N	N/A	3	4	2	`CALL #LABEL`
X(Rn)	4	4	5	2	`CALL 2(R7)`
EDE	4	4	5	2	`PUSH EDE`
&EDE	4	4	6	2	`SXT &EDE`

MSP430X CPU各种寻址方式的指令周期和长度

MSP430x6xx Family User's Guide

Table 6-10. MSP430 Format I Instructions Cycles and Length

Addressing Mode		No. of Cycles	Length of Instruction	Example
Source	Destination			
Rn	Rm	1	1	MOV R5,R8
	PC	3	1	BR R9
	x(Rm)	4(1)	2	ADD R5,4(R6)
	EDE	4(1)	2	XOR R8,EDE
	&EDE	4(1)	2	MOV R5,&EDE
@Rn	Rm	2	1	AND @R4,R5
	PC	4	1	BR @R8
	x(Rm)	5(1)	2	XOR @R5,8(R6)
	EDE	5(1)	2	MOV @R5,EDE
	&EDE	5(1)	2	XOR @R5,&EDE
@Rn+	Rm	2	1	ADD @R5+,R6
	PC	4	1	BR @R9+
	x(Rm)	5(1)	2	XOR @R5,8(R6)
	EDE	5(1)	2	MOV @R9+,EDE
	&EDE	5(1)	2	MOV @R9+,&EDE

MSP430X CPU各种寻址方式的指令周期和长度(续)

MSP430x6xx Family User's Guide

Table 6-10. MSP430 Format I Instructions Cycles and Length

Addressing Mode		No. of Cycles	Length of Instruction	Example
Source	Destination			
#N	Rm	2	2	MOV #20,R9
	PC	3	2	BR #2AEh
	x(Rm)	5[1]	3	MOV #0300h,0(SP)
	EDE	5[1]	3	ADD #33,EDE
	&EDE	5[1]	3	ADD #33,&EDE
x(Rn)	Rm	3	2	MOV 2(R5),R7
	PC	5	2	BR 2(R6)
	TONI	6[1]	3	MOV 4(R7),TONI
	x(Rm)	6[1]	3	ADD 4(R4),6(R9)
	&TONI	6[1]	3	MOV 2(R4),&TONI
EDE	Rm	3	2	AND EDE,R6
	PC	5	2	BR EDE
	TONI	6[1]	3	CMP EDE,TONI
	x(Rm)	6[1]	3	MOV EDE,0(SP)
	&TONI	6[1]	3	MOV EDE,&TONI
&EDE	Rm	3	2	MOV &EDE,R8
	PC	5	2	BR &EDE
	TONI	6[1]	3	MOV &EDE,TONI
	x(Rm)	6[1]	3	MOV &EDE,0(SP)
	&TONI	6[1]	3	MOV &EDE,&TONI

(1) MOV, BIT, and CMP instructions execute in one fewer cycle.

- 执行指令的过程中，
  有时需从存储器或I/O端口读取或存放数据，
  故一个指令周期通常包含若干个总线周期

思考 执行 **ADD 4(R5), 8(R15)**过程中，包含什么总线周期？

- 虽然一条指令从取指开始到执行完毕，
  通常要经历若干总线周期，
  但对采用了流水线技术的CPU，
  指令周期并不简单等于这些总线周期的时间和。

- **CPI (clock cycles per second)**
  执行每条指令所需的时钟周期数的平均值
  通常根据不同指令出现的频度，乘上不同的系数，求得的统计平均值

- 一个程序的CPU时间
  = 一个程序的CPU时钟周期数 x 时钟周期时间
  = 指令数 x CPI x 时钟周期时间
  = 指令数 x CPI / 时钟频率

第2节　MSP430F6xx 时钟系统模块

Unified Clock System Module

一、时钟系统模块结构图及构成 *Diagram*

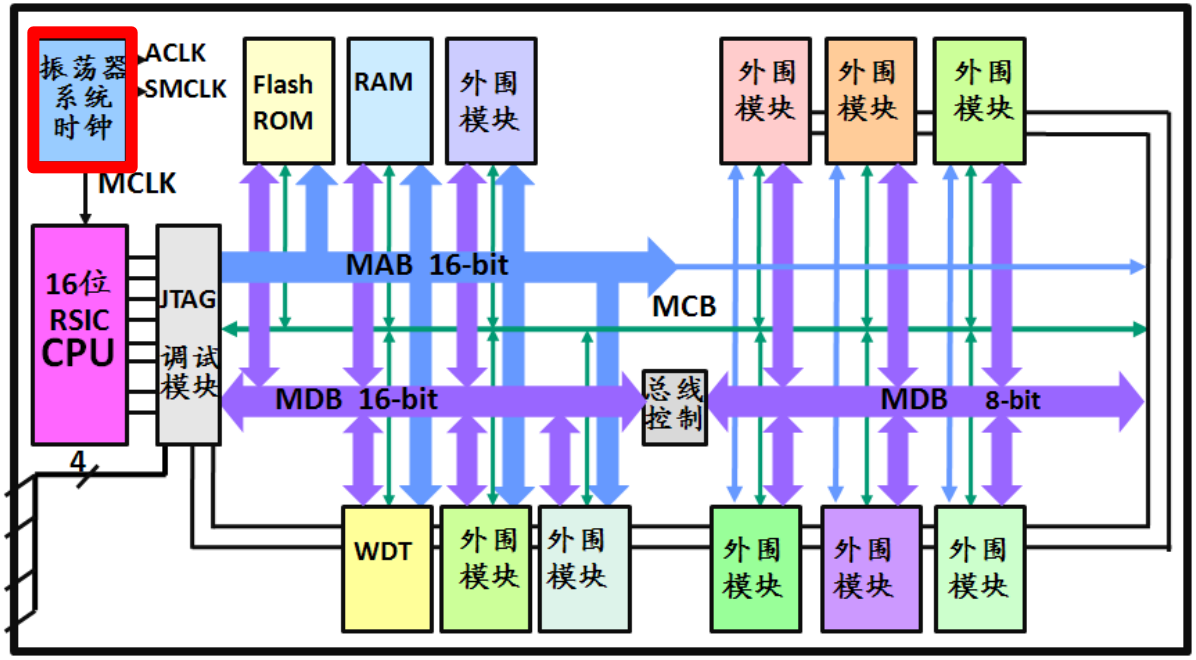
二、时钟系统模块相关寄存器 *Registers*

三、时钟系统模块输出引脚 *Pins*

四、振荡器失效检测 *Oscillator Fault Detection*

五、时钟系统设置举例 *Examples*

一、 时钟系统模块结构图及构成



- 时钟信号是定时操作的基本信号，
  在时钟的作用下，各部件可以有条不紊地自动工作
- 基本时钟模块由高速晶体振荡器、低速晶体振荡器、
  数字控制振荡器 DCO、锁频环FLL等部分构成
- 多种时钟有利于实时应用系统对低功耗和快速响应外部事件要求
- 不同系列单片机包含的时钟模块不完全相同

一、时钟系统模块结构图及构成

1. 结构图
2. 构成

振荡器

时钟源

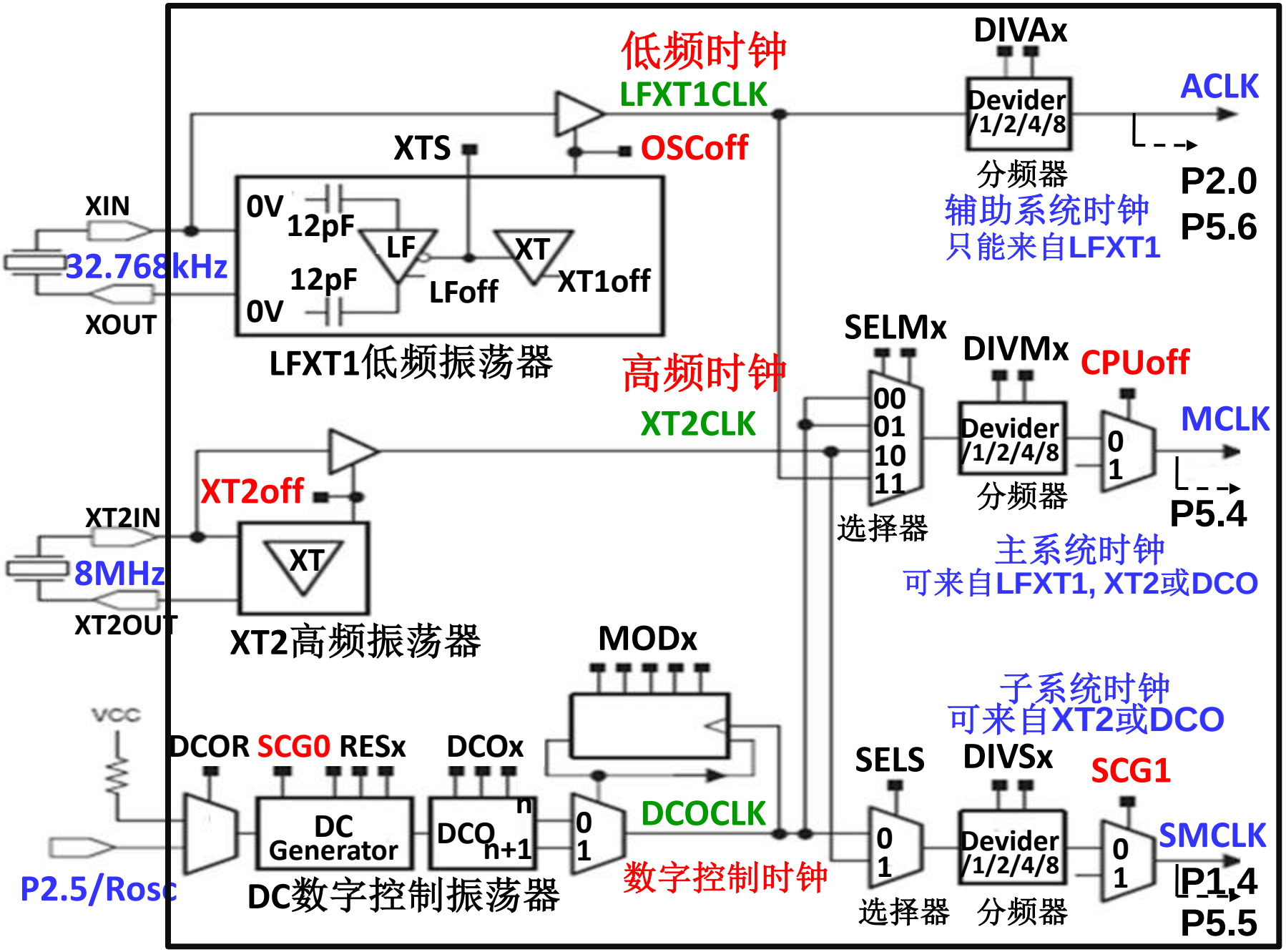
时钟信号

外设模块请求时钟系统

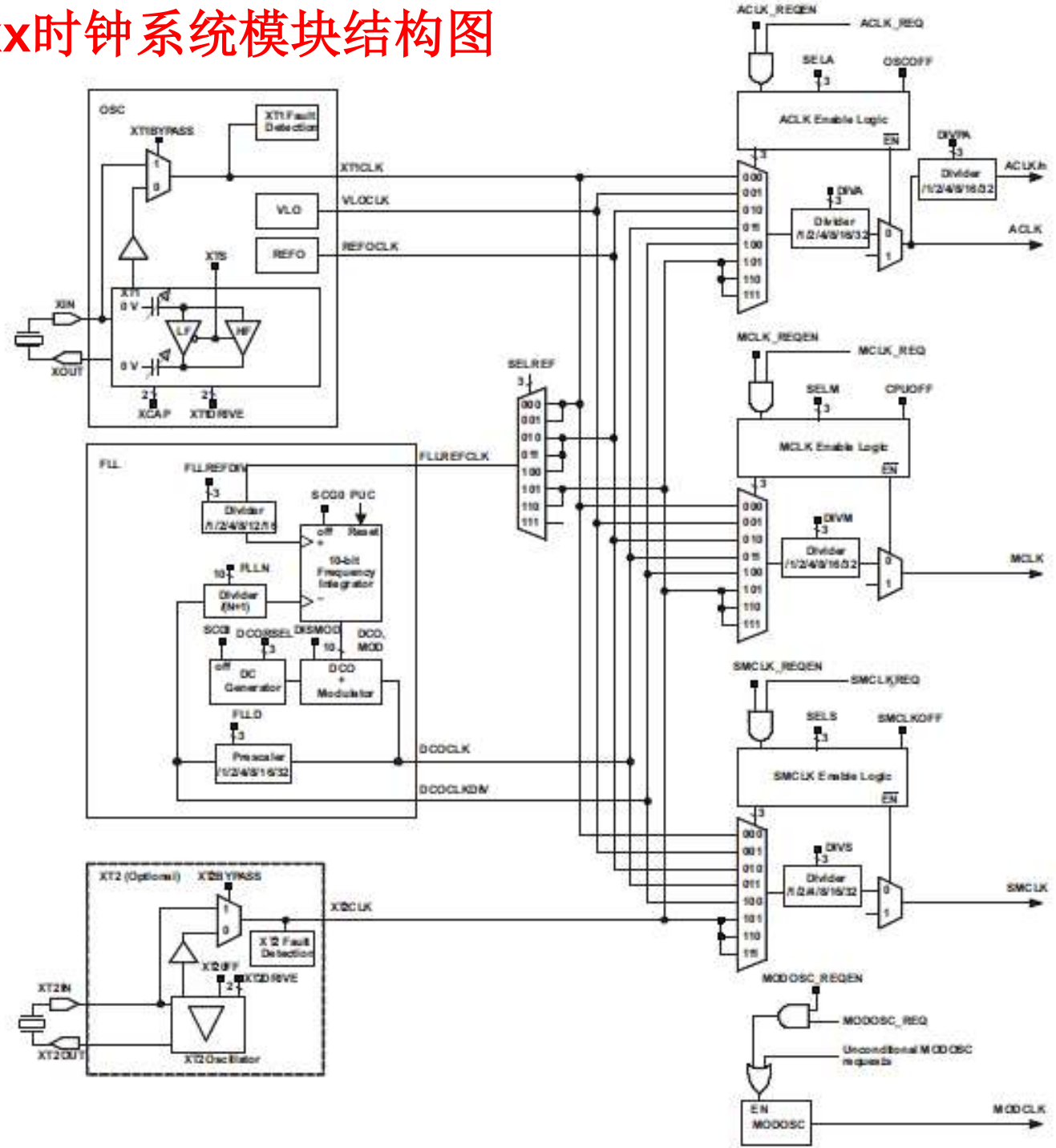
内部的振荡器MODOSC

1. 结构图

MSP430F1xx基本时钟模块结构图



MSP430F6xx时钟系统模块结构图

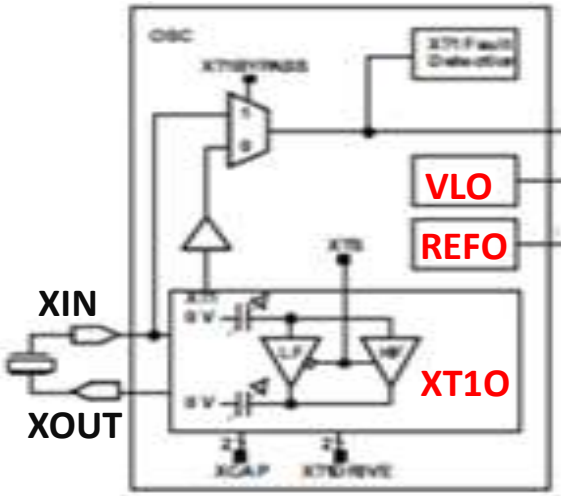


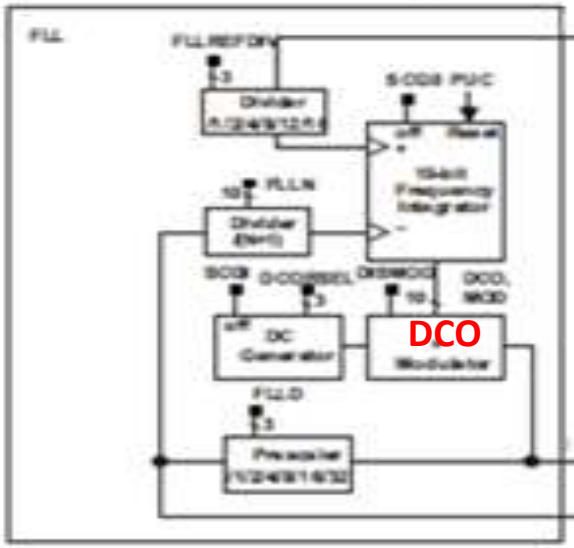
2、时钟系统模块构成

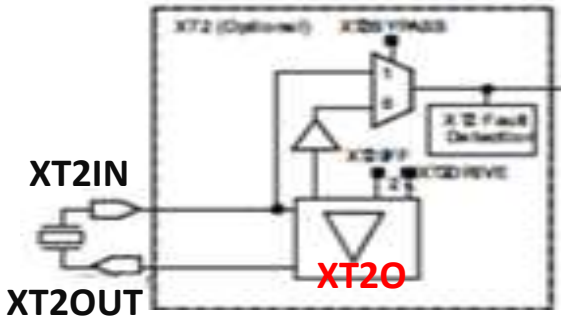
1). 时钟振荡器

包括五个时钟输入源振荡器

- XT1O 振荡器
- XT2O 振荡器
- VLO片内低功耗低频振荡器
- REFO片内已整形的低频参考振荡器
- DCO片内数字控制振荡器







■ XT1O 振荡器

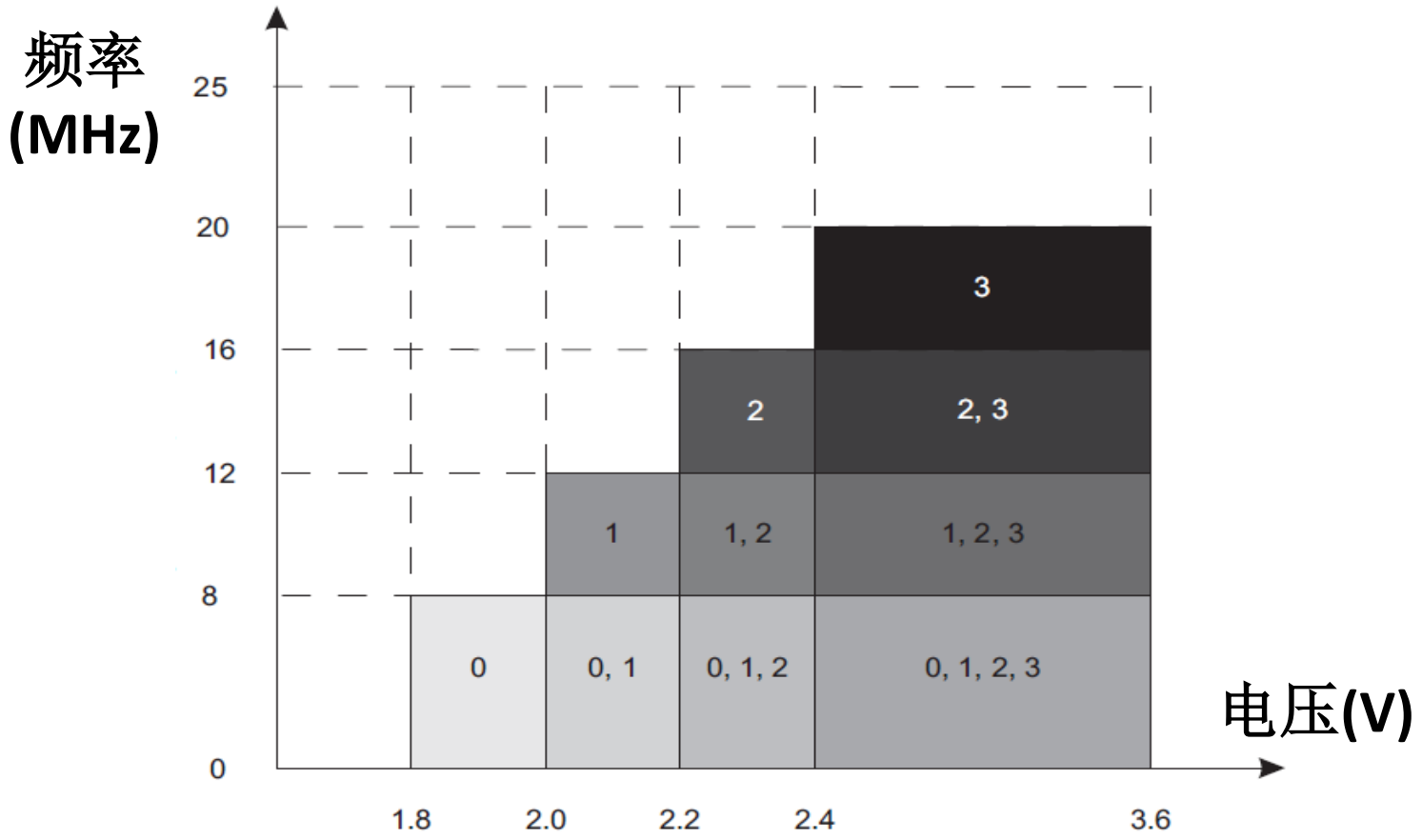
- ✓ 产生XT1CLK时钟信号.
- ✓ XTS=0时，XT1工作在LF低频模式时，提供支持32768Hz时钟的超低功耗模式。由于片内集成了保证工作稳定的电容，外部晶振只需经过XIN和XOUT两个引脚连接，不需要其他外部器件。片内电容可通过控制寄存器中的XCAP选择.
- ✓ XTS=1时，一些MCU中的XT1可以选择工作在HF高频模式，此时外部的高频晶振或谐振器连接到XIN和XOUT引脚，需要在两个端口配置电容.
- ✓ 当把XT1BYPASS=1，可XIN1引脚直接接入时钟，此时振荡器处于断电状态.

XT2O 振荡器

- ✓ 产生XT2CLK时钟信号
- ✓ XT2O工作特性与 XT1振荡器工作在高频模式时类似，与之有关的外部引脚为XT2IN和XT2OUT

系统频率和工作电压的关系

系统频率和系统的工作电压关系密切，见下图



(方格中的数字表示所需要的PMMCOREVx配置)

■ VLO片内低功耗低频振荡器

- ✓ 产生VLOCLK时钟，典型振荡频率为10kHz，不需要外接晶振。
- ✓ 对时钟精确要求不高的应用，可以使用VLO，达到低成本和超低功耗的目的。

■ REFO 片内已整形低频参考振荡器

- ✓ 可生REFOCLK时钟,典型值为32768Hz，

  一个比较稳定的频率，可用作FLLREFCLK.
- ✓ 用在没有外部晶振，对成本又比较敏感的场合.

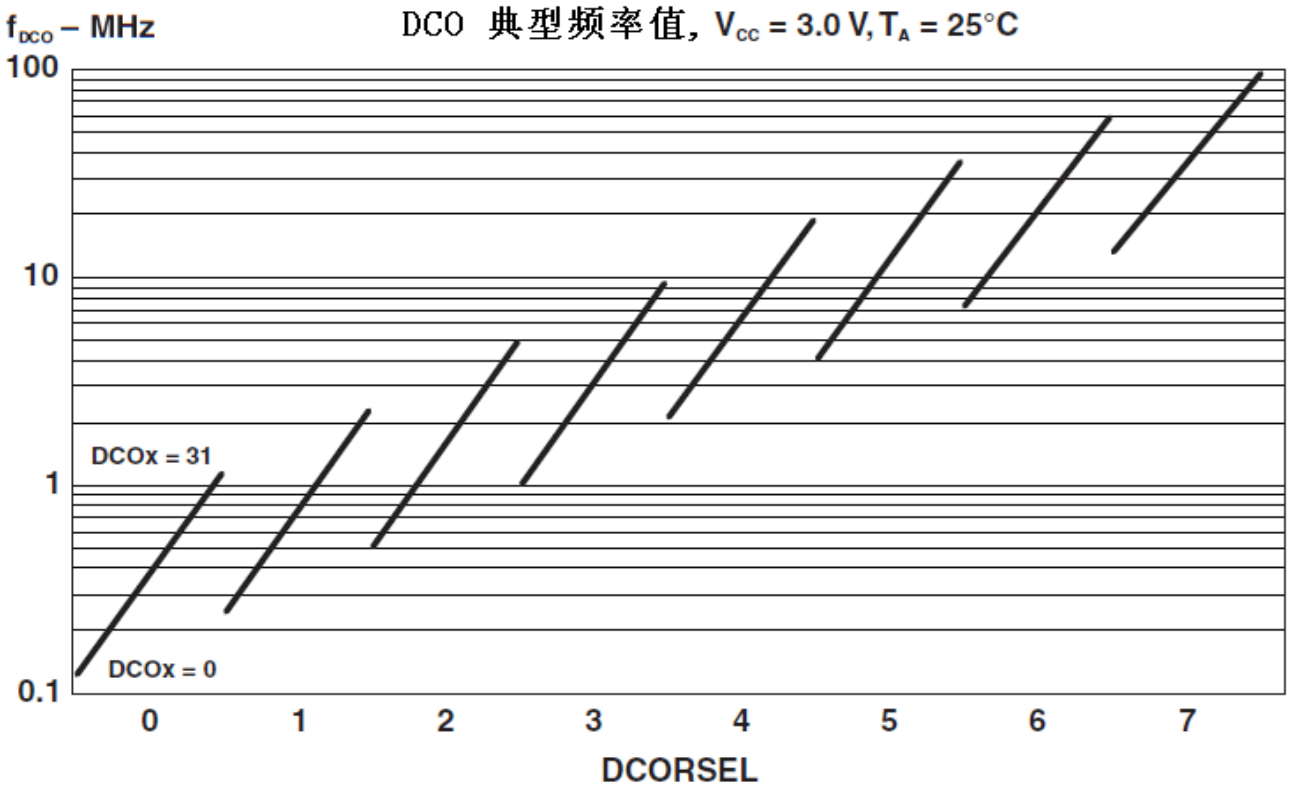
■ DCO片内数字控制振荡器

- ✓ 可数字控制的RC振荡器, 产生DCOCLK时钟.

  频率受供电电压、环境温度影响，稳定性不好.
- ✓ 可通过选择FLL锁相环, 增强DCO振荡频率的稳定性.

DCOCLK频率调整过程：

- 设置3位(0~7)的DCORSELx值，
  从8个DCO额定频率中选择一个频率.
- 设置5位的DCO (0~31),
  选择某DCORSEL下32个频率中的一个频率，
  相邻两个的频率相差约8%.

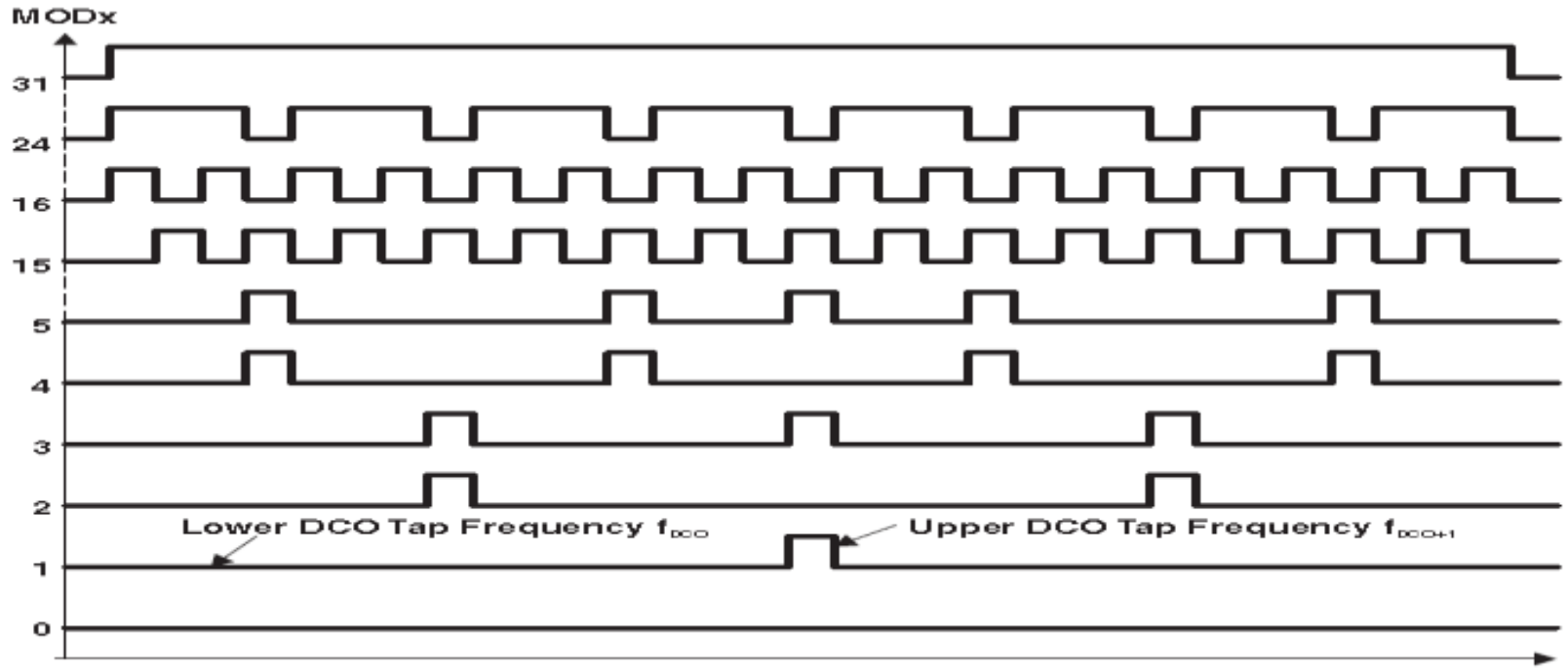


DCOCLK
频率的调节

DCO调制器 (DCO Mofulator)

设置5位的MOD， 控制在每32个周期DCO中的中，利用当前的相邻的两DCO频率进行微调，得到DCOCLK，调整后的32个DCOCLK时钟周期：

$$t = (32\text{-}MOD) \times t_{DCO} + MOD \times t_{DCO+1}$$



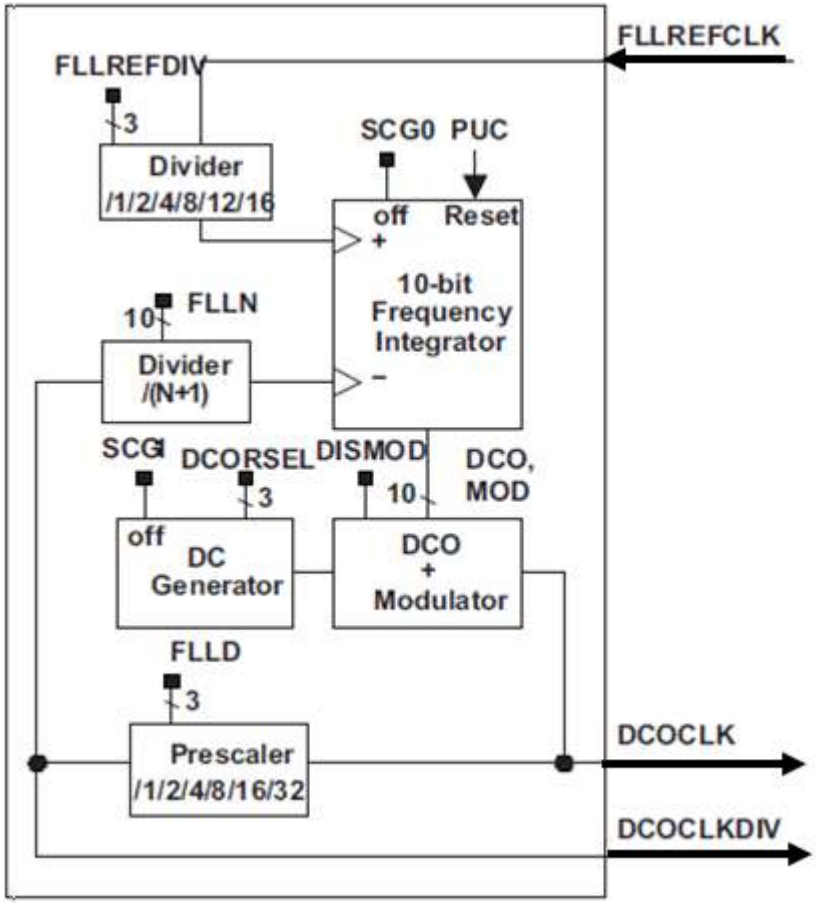
FLL锁频环(Frequency Locked Loop)

- FLL锁频环通过频率积分器和调制器的自动调节使DCOCLK的频率趋于稳定. 在锁频环工作时，DCO位和MOD位的值由硬件自动调节.

- 锁相环FLLREFCLK输入信号与两个输出信号DCOCLK和DCOCLKDIV的关系：

  DCOCLK= FLLD X (FLLN+1) X FLLREFCLK / FLLREFDIV

  DCOCLKDIV= DCOCLK / FLLD



2). MSP430F6xx时钟模块有 5 个时钟源：

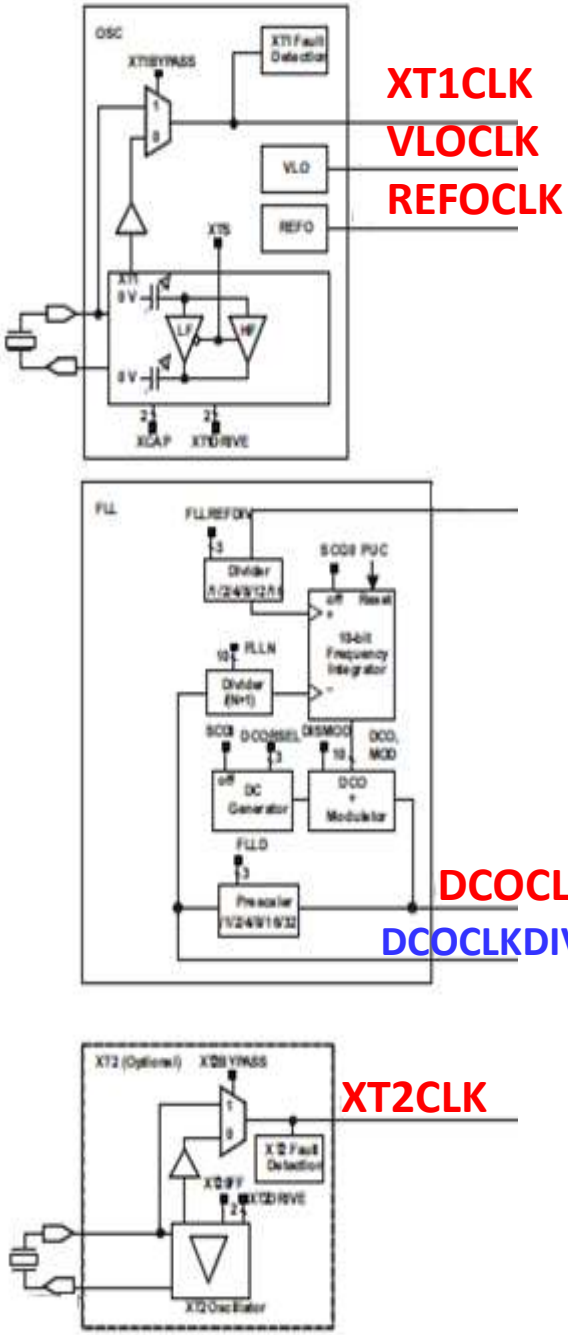
- **XT1CLK** 低频或高频时钟源
  可使用32768Hz的低频晶振、
  或外接时钟，频率范围4~32MHz，
  可作为FLL基准时钟源

- **VLOCLK** 片内低功耗低频时钟源
  典型值为10KHz.

- **REFOCLK** 片内已整形的低频时钟源
  典型值为32768Hz，可作为FLL基准时钟源.

- **DCOCLK** 片内数字控制时钟源
  可通过FLL模块来稳定.
  DCOCLKDIV是DCOCLK分频得到的时钟

- **XT2CLK** 高频时钟源
  可接标准晶振、振荡器或外部时钟，
  频率范围在4MHz～32MHz。



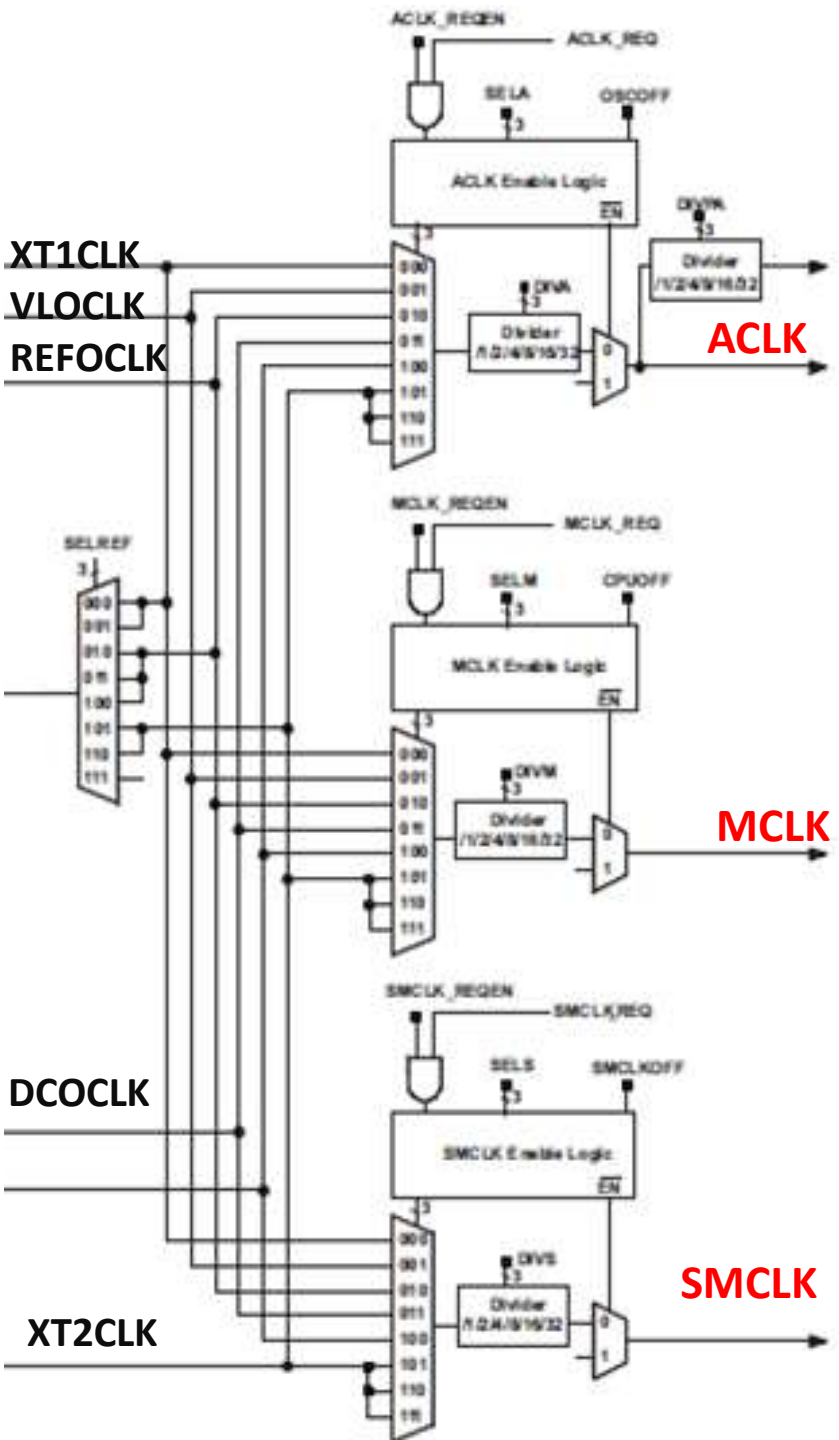
3)．时钟系统模块提供3种时钟信号

- **ACLK** 辅助时钟
  ACLK一般用于低速外设模块.

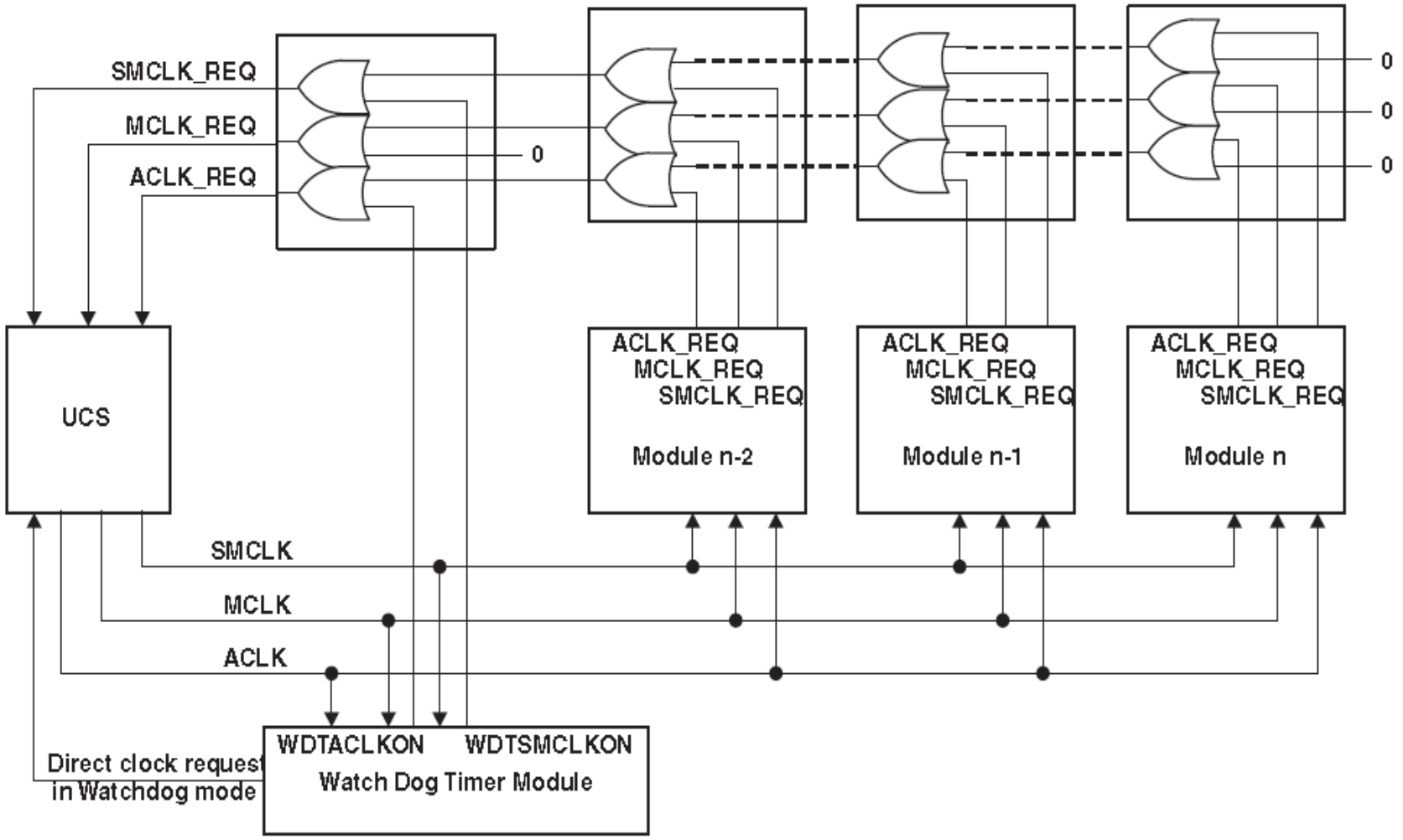
- **MCLK** 系统主时钟
  MCLK主要用于CPU和系统.

- **SMCLK** 子系统时钟
  主要用于高速外设模块.

ACLK、MCLK、SMCLK均可通过软件从XT1CLK、VLOCLK、REFOCLK、DCOCLK、DCOCLKDIV、XT2CLK这6个时钟源中选择，经1/2/4/8/16/32分频得到。



4). 外设模块请求时钟系统

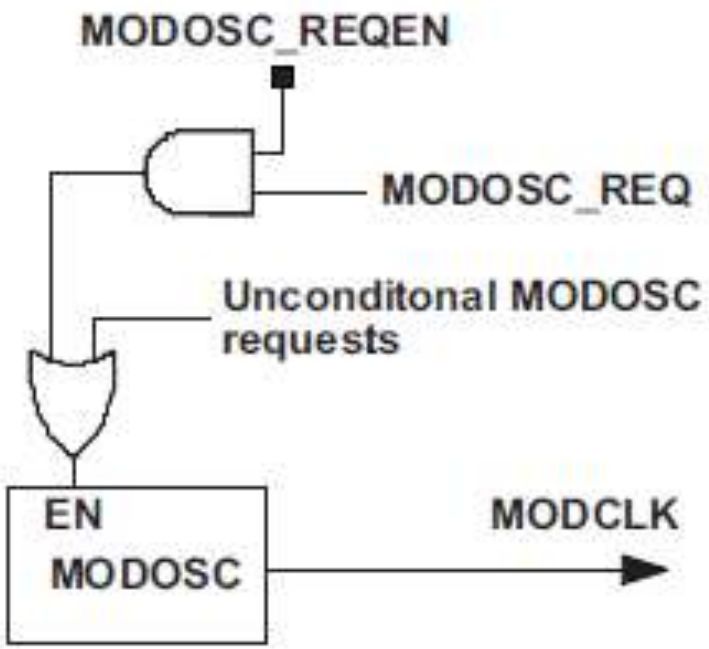


当进入MCU低功耗模式，某时钟被关闭，而外设选择了此时钟时，外设发出时钟请求，激活该时钟，不受低功耗影响

5). MODOSC 内部的振荡器

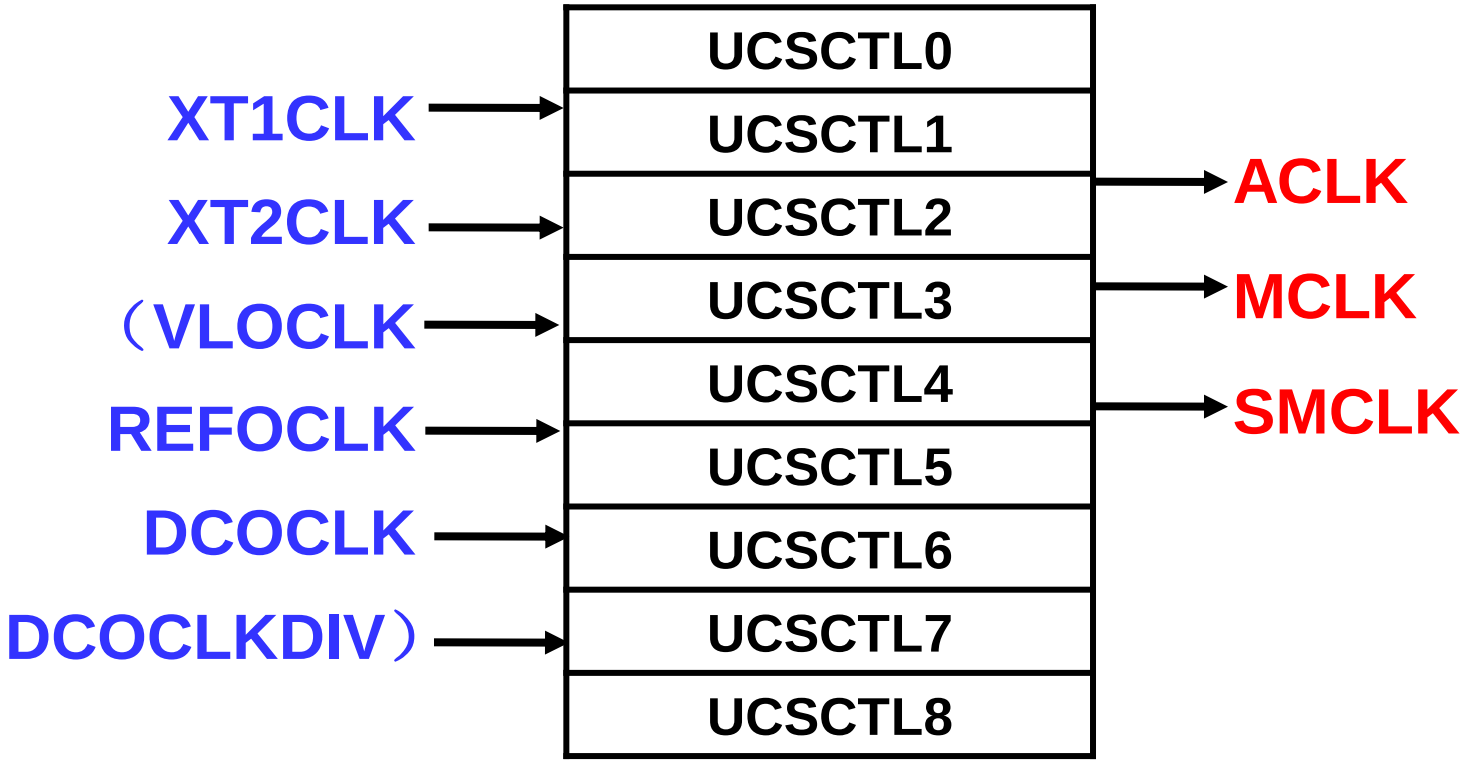
主要向**FLASH**模块控制器或其他模块提供时钟。

MODOSC产生时钟信号**MODCLK**。



二、时钟系统模块相关寄存器

- 各系列不同时钟模块均可输出辅助时钟**ACLK**、主系统时钟**MCLK**、子系统时钟**SMCLK.**
- 内部有九个控制寄存器，均为**16**位寄存器



MSP430F663x时钟模块编程结构图

Unified Clock System Control 0 Register (UCSCTL0)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved			DCO				
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
MOD					Reserved		
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	r0

Reserved	Bits 15-13	Reserved. Reads back as 0.
DCO	Bits 12-8	DCO tap selection. These bits select the DCO tap and are modified automatically during FLL operation.
MOD	Bits 7-3	Modulation bit counter. These bits select the modulation pattern. All MOD bits are modified automatically during FLL operation. The DCO register value is incremented when the modulation bit counter rolls over from 31 to 0. If the modulation bit counter decrements from 0 to the maximum count, the DCO register value is also decremented.
Reserved	Bits 2-0	Reserved. Reads back as 0.

Unified Clock System Control 1 Register (UCSCTL1)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved	DCORSEL			Reserved		Reserved	DISMOD
r0	rw-0	rw-1	rw-0	r0	r0	rw-0	rw-0

Reserved	Bits 15-8	Reserved. Reads back as 0.
Reserved	Bit 7	Reserved. Reads back as 0.
DCORSEL	Bits 6-4	DCO frequency range select. These bits select the DCO frequency range of operation defined in the device-specific datasheet.
Reserved	Bits 3-2	Reserved. Reads back as 0.
Reserved	Bit 1	Reserved. Reads back as 0.
DISMOD	Bit 0	Modulation. This bit enables/disables the modulation. 0 Modulation enabled 1 Modulation disabled

Unified Clock System Control 2 Register (UCSCTL2)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	FLLD			Reserved		FLLN	
r0	rw-0	rw-0	rw-1	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
FLLN							
rw-0	rw-0	rw-0	rw-1	rw-1	rw-1	rw-1	rw-1

Reserved	Bit 15	Reserved. Reads back as 0.
FLLD	Bits 14-12	FLL loop divider. These bits divide f_{DCOCLK} in the FLL feedback loop. This results in an additional multiplier for the multiplier bits. See also multiplier bits. 000 $f_{DCOCLK}/1$ 001 $f_{DCOCLK}/2$ 010 $f_{DCOCLK}/4$ 011 $f_{DCOCLK}/8$ 100 $f_{DCOCLK}/16$ 101 $f_{DCOCLK}/32$ 110 Reserved for future use. Defaults to $f_{DCOCLK}/32$. 111 Reserved for future use. Defaults to $f_{DCOCLK}/32$.
Reserved	Bits 11-10	Reserved. Reads back as 0.
FLLN	Bits 9-0	Multiplier bits. These bits set the multiplier value N of the DCO. N must be greater than 0. Writing zero to FLLN causes N to be set to 1.

Unified Clock System Control 3 Register (UCSCTL3)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved	SELREF			Reserved	FLLREFDIV		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

Reserved	Bits 15-8	Reserved. Reads back as 0.
Reserved	Bit 7	Reserved. Reads back as 0.
SELREF	Bits 6-4	FLL reference select. These bits select the FLL reference clock source. 000 XT1CLK 001 Reserved for future use. Defaults to XT1CLK. 010 REFOCLK 011 Reserved for future use. Defaults to REFOCLK. 100 Reserved for future use. Defaults to REFOCLK. 101 XT2CLK when available, otherwise REFOCLK. 110 Reserved for future use. XT2CLK when available, otherwise REFOCLK. 111 Reserved for future use. XT2CLK when available, otherwise REFOCLK.
Reserved	Bit 3	Reserved. Reads back as 0.
FLLREFDIV	Bits 2-0	FLL reference divider. These bits define the divide factor for $f_{FLLREFCLK}$. The divided frequency is used as the FLL reference frequency. 000 $f_{FLLREFCLK}/1$ 001 $f_{FLLREFCLK}/2$ 010 $f_{FLLREFCLK}/4$ 011 $f_{FLLREFCLK}/8$ 100 $f_{FLLREFCLK}/12$ 101 $f_{FLLREFCLK}/16$ 110 Reserved for future use. Defaults to $f_{FLLREFCLK}/16$. 111 Reserved for future use. Defaults to $f_{FLLREFCLK}/16$.

Unified Clock System Control 4 Register (UCSCTL4)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved					SELA		
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved	SELS			Reserved	SELM		
r0	rw-1	rw-0	rw-0	r0	rw-1	rw-0	rw-0

Reserved	Bits 15-11	Reserved. Reads back as 0.
SELA	Bits 10-8	Selects the ACLK source 000 XT1CLK 001 VLOCLK 010 REFOCLK 011 DCOCLK 100 DCOCLKDIV 101 XT2CLK when available, otherwise DCOCLKDIV 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
Reserved	Bit 7	Reserved. Reads back as 0.
SELS	Bits 6-4	Selects the SMCLK source 000 XT1CLK 001 VLOCLK 010 REFOCLK 011 DCOCLK 100 DCOCLKDIV 101 XT2CLK when available, otherwise DCOCLKDIV 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
Reserved	Bit 3	Reserved. Reads back as 0.
SELM	Bits 2-0	Selects the MCLK source 000 XT1CLK 001 VLOCLK 010 REFOCLK 011 DCOCLK 100 DCOCLKDIV 101 XT2CLK when available, otherwise DCOCLKDIV 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.

Unified Clock System Control 5 Register (UCSCTL5)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	DIVPA			Reserved	DIVA		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved	DIVS			Reserved	DIVM		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

Reserved Bit 15 Reserved. Reads back as 0.

DIVPA Bits 14-12 ACLK source divider available at external pin. Divides the frequency of ACLK and presents it to an external pin.

- 000 $f_{ACLK}/1$
- 001 $f_{ACLK}/2$
- 010 $f_{ACLK}/4$
- 011 $f_{ACLK}/8$
- 100 $f_{ACLK}/16$
- 101 $f_{ACLK}/32$
- 110 Reserved for future use. Defaults to $f_{ACLK}/32$.
- 111 Reserved for future use. Defaults to $f_{ACLK}/32$.

Reserved Bit 11 Reserved. Reads back as 0.

DIVA Bits 10-8 ACLK source divider. Divides the frequency of the ACLK clock source.

- 000 $f_{ACLK}/1$
- 001 $f_{ACLK}/2$
- 010 $f_{ACLK}/4$
- 011 $f_{ACLK}/8$
- 100 $f_{ACLK}/16$
- 101 $f_{ACLK}/32$
- 110 Reserved for future use. Defaults to $f_{ACLK}/32$.
- 111 Reserved for future use. Defaults to $f_{ACLK}/32$.

Reserved Bit 7 Reserved. Reads back as 0.

DIVS Bits 6-4 SMCLK source divider

- 000 $f_{SMCLK}/1$
- 001 $f_{SMCLK}/2$
- 010 $f_{SMCLK}/4$
- 011 $f_{SMCLK}/8$
- 100 $f_{SMCLK}/16$
- 101 $f_{SMCLK}/32$
- 110 Reserved for future use. Defaults to $f_{SMCLK}/32$.
- 111 Reserved for future use. Defaults to $f_{SMCLK}/32$.

Unified Clock System Control 6 Register (UCSCTL6)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
XT2DRIVE		Reserved	XT2BYPASS	Reserved			XT2OFF
rw-1	rw-1	r0	rw-0	r0	r0	r0	rw-1
7	6	5	4	3	2	1	0
XT1DRIVE		XTS	XT1BYPASS	XCAP		SMCLKOFF	XT1OFF
rw-1	rw-1	rw-0	rw-0	rw-1	rw-1	rw-0	rw-1

XT2DRIVE Bits 15-14 The XT2 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength.

- 00 Lowest current consumption. XT2 oscillator operating range is 4 MHz to 8 MHz.
- 01 Increased drive strength XT2 oscillator. XT2 oscillator operating range is 8 MHz to 16 MHz.
- 10 Increased drive capability XT2 oscillator. XT2 oscillator operating range is 16 MHz to 24 MHz.
- 11 Maximum drive capability and maximum current consumption for both XT2 oscillator. XT2 oscillator operating range is 24 MHz to 32 MHz.

Reserved Bit 13 Reserved. Reads back as 0.

XT2BYPASS Bit 12 XT2 bypass select

- 0 XT2 sourced internally
- 1 XT2 sourced externally from pin

Reserved Bits 11-9 Reserved. Reads back as 0.

XT2OFF Bit 8 Turns off the XT2 oscillator

- 0 XT2 is on if XT2 is selected via the port selection and XT2 is not in bypass mode of operation.
- 1 XT2 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation.

XT1DRIVE Bits 7-6 The XT1 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength.

- 00 Lowest current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 4 MHz to 8 MHz.
- 01 Increased drive strength for XT1 LF mode. XT1 oscillator operating range in HF mode is 8 MHz to 16 MHz.
- 10 Increased drive capability for XT1 LF mode. XT1 oscillator operating range in HF mode is 16 MHz to 24 MHz.
- 11 Maximum drive capability and maximum current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 24 MHz to 32 MHz.

XTS Bit 5 XT1 mode select

- 0 Low-frequency mode. XCAP bits define the capacitance at the XIN and XOUT pins.
- 1 High-frequency mode. XCAP bits are not used.

XT1BYPASS Bit 4 XT1 bypass select

- 0 XT1 sourced internally
- 1 XT1 sourced externally from pin

Unified Clock System Control 7 Register (UCSCTL7)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved		Reserved	Reserved	Reserved		Reserved	
r0	r0	rw-0	rw-(0)	rw-(1)	rw-(1)	r-1	r-1

7	6	5	4	3	2	1	0
Reserved			Reserved	XT2OFFG [1]	XT1HFOFFG [1]	XT1LFOFFG	DCOFFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(1)	rw-(1)

Reserved	Bits 15-14	Reserved. Reads back as 0.
Reserved	Bit 13	Reserved. This bit must always be written with 0.
Reserved	Bit 12	Reserved. This bit must always be written with 0.
Reserved	Bits 11-10	Reserved. The states of these bits should be ignored.
Reserved	Bits 9-8	Reserved. The states of these bits should be ignored.
Reserved	Bits 7-5	Reserved. Reads back as 0.
Reserved	Bit 4	Reserved. The state of this bit should be ignored.
XT2OFFG [1]	Bit 3	XT2 oscillator fault flag. If this bit is set, the OFIFG flag is also set. XT2OFFG is set if a XT2 fault condition exists. XT2OFFG can be cleared via software. If the XT2 fault condition still remains, XT2OFFG is set. 0 No fault condition occurred after the last reset. 1 XT2 fault. An XT2 fault occurred after the last reset.
XT1HFOFFG [1]	Bit 2	XT1 oscillator fault flag (HF mode). If this bit is set, the OFIFG flag is also set. XT1HFOFFG is set if a XT1 fault condition exists. XT1HFOFFG can be cleared via software. If the XT1 fault condition still remains, XT1HFOFFG is set. 0 No fault condition occurred after the last reset. 1 XT1 fault. An XT1 fault occurred after the last reset.
XT1LFOFFG	Bit 1	XT1 oscillator fault flag (LF mode). If this bit is set, the OFIFG flag is also set. XT1LFOFFG is set if a XT1 fault condition exists. XT1LFOFFG can be cleared via software. If the XT1 fault condition still remains, XT1LFOFFG is set. 0 No fault condition occurred after the last reset. 1 XT1 fault (LF mode). A XT1 fault occurred after the last reset.
DCOFFG	Bit 0	DCO fault flag. If this bit is set, the OFIFG flag is also set. The DCOFFG bit is set if DCO = {0} or DCO = {31}. DCOFFG can be cleared via software. If the DCO fault condition still remains, DCOFFG is set. 0 No fault condition occurred after the last reset. 1 DCO fault. A DCO fault occurred after the last reset.

Unified Clock System Control 8 Register (UCSCTL8)

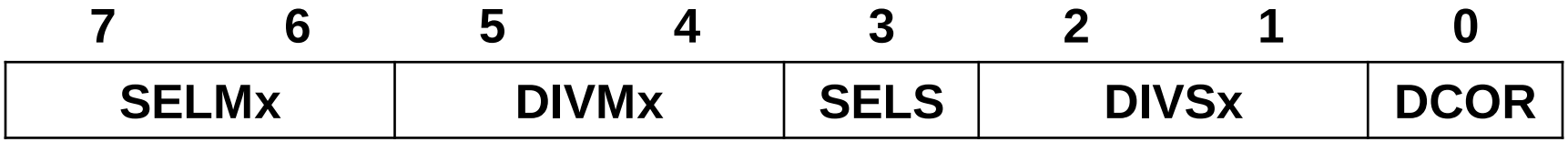
15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved					Reserved		
r0	r0	r0	r0	r0	rw-(1)	rw-(1)	rw-(1)
7	6	5	4	3	2	1	0
Reserved			Reserved	MODOSC REQEN	SMCLKREQEN	MCLKREQEN	ACLKREQEN
r0	r0	r0	rw-(0)	rw-(0)	rw-(1)	rw-(1)	rw-(1)

Reserved	Bits 15-11	Reserved. Reads back as 0.
Reserved	Bits 10-8	Reserved. Must always be written as 1.
Reserved	Bits 7-5	Reserved. Reads back as 0.
Reserved	Bit 4	Reserved. Must always be written as 0.
MODOSCREQEN	Bit 3	MODOSC clock request enable. Setting this enables conditional module requests for MODOSC. 0 MODOSC conditional requests are disabled. 1 MODOSC conditional requests are enabled.
SMCLKREQEN	Bit 2	SMCLK clock request enable. Setting this enables conditional module requests for SMCLK 0 SMCLK conditional requests are disabled. 1 SMCLK conditional requests are enabled.
MCLKREQEN	Bit 1	MCLK clock request enable. Setting this enables conditional module requests for MCLK 0 MCLK conditional requests are disabled. 1 MCLK conditional requests are enabled.
ACLKREQEN	Bit 0	ACLK clock request enable. Setting this enables conditional module requests for ACLK 0 ACLK conditional requests are disabled. 1 ACLK conditional requests are enabled.

BCSCTL2

基本时钟系统控制寄存器2

Basic Clock System Control Register 2



选择MCLK
select MCLK
00: DCOCLK
01: DCOCLK
10:有XT2,选XT2
　无XT2,选LFXT1CLK
11: LFXT1CLK

MCLK分频控制
00： /1
01： /2
10： /4
11： /8

选择SMCLK
0:DCOCLK
1 :XT2
　无XT2,
　选LFXT1CLK

SMCLK分频控制
00： /1
01： /2
10： /4
11： /8

选择DCO电阻
0: 片内电阻
1 :片外电阻

注意：
复位后，MCLK、SMCLK选择DCO为时钟，1分频

BCSCTL2

基本时钟系统控制寄存器2

Basic Clock System Control Register 2

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-0	rw-0	rw-0

io430x14x.h

```
__no_init volatile union
{  unsigned char BCSCTL2;  //Basic Clock System Control 2
   struct
   {   unsigned char DCOR  : 1;  //Enable External Resistor
       unsigned char DIVS0  : 1;  // SMCLK Divider 0
       unsigned char DIVS1   : 1;  //SMCLK Divider 1
       unsigned char SELS     : 1;  // SMCLK Source Select 0:DCO/ 1:XT2/LFXT
       unsigned char DIVM0  : 1;  // CLK Divider 0
       unsigned char DIVM1  : 1;  // CLK Divider 1
       unsigned char SELM0  : 1;  // MCLK Source Select 0
       unsigned char SELM1  : 1;  // MCLK Source Select 1
    } BCSCTL2_bit;
} @ 0x0058;
```

io430x14x.h

```
#define   DIVA_0        (0x00)  /* ACLK Divider 0: /1 */
#define   DIVA_1        (0x10)  /* ACLK Divider 1: /2 */
#define   DIVA_2        (0x20)  /* ACLK Divider 2: /4 */
#define   DIVA_3        (0x30)  /* ACLK Divider 3: /8 */

#define   DIVS_0        (0x00)  /* SMCLK Divider 0: /1 */
#define   DIVS_1        (0x02)  /* SMCLK Divider 1: /2 */
#define   DIVS_2        (0x04)  /* SMCLK Divider 2: /4 */
#define   DIVS_3        (0x06)  /* SMCLK Divider 3: /8 */

#define   DIVM_0        (0x00)  /* MCLK Divider 0: /1 */
#define   DIVM_1        (0x10)  /* MCLK Divider 1: /2 */
#define   DIVM_2        (0x20)  /* MCLK Divider 2: /4 */
#define   DIVM_3        (0x30)  /* MCLK Divider 3: /8 */

#define   SELM_0        (0x00)  /* MCLK Source Select 0: DCOCLK */
#define   SELM_1        (0x40)  /* MCLK Source Select 1: DCOCLK */
#define   SELM_2        (0x80)  /* MCLK Source Select 2: XT2CLK/LFXTCLK */
#define   SELM_3        (0xC0)  /* MCLK Source Select 3: LFXTCLK */
```

三、时钟系统模块输出管脚

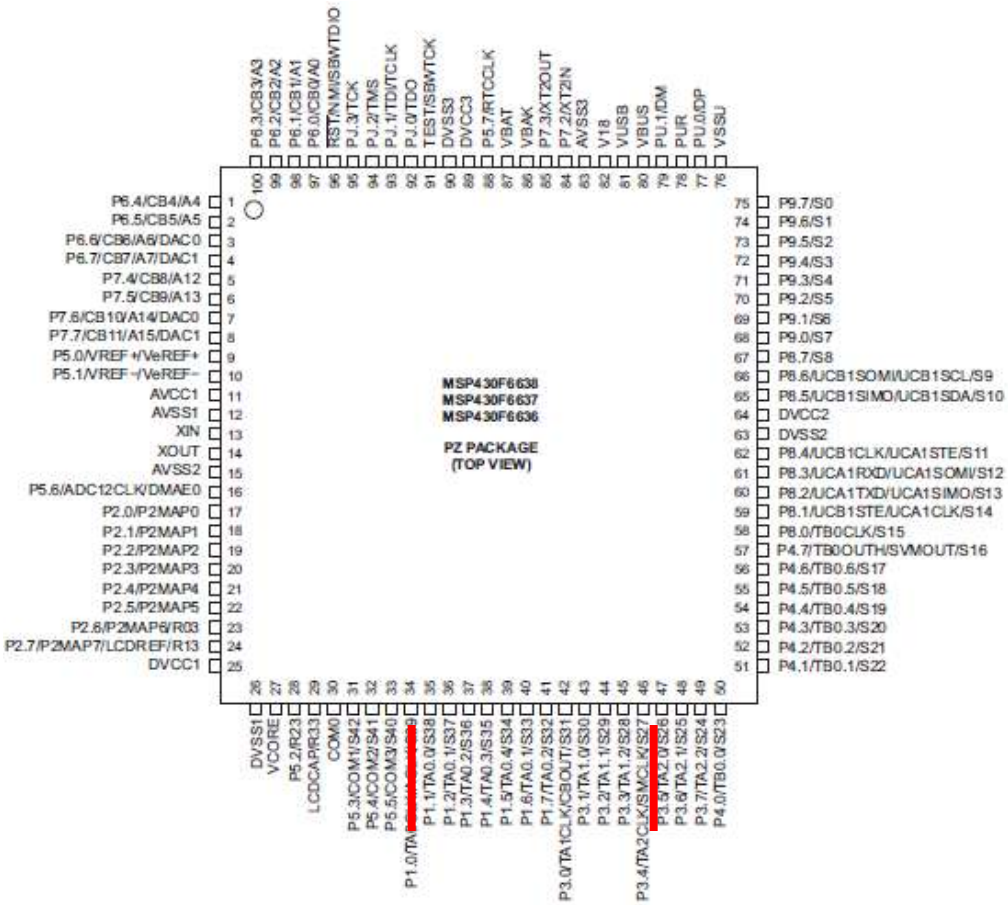
可通过MCU的相关管脚输出ACLK、SMCLK时钟信号

将管脚P1.0、P3.4的功能选择寄存器

SEL设置为1(模块功能)、方向寄存器DIR设为输出，

可通过这些管脚分别输出ACLK、SMCLK

```
//P1.0/TA0CLK/ACLK/S39
//P1.0 输出ACLK
P1SEL |= BIT0 ;
P1DIR  |= BIT0;
//P3.4/TA2CLK/SMCLK/S27
//P3.4 输出SMCLK
P3SEL |= BIT4 ;
P3DIR |= BIT4;
```



四、振荡器失效检测

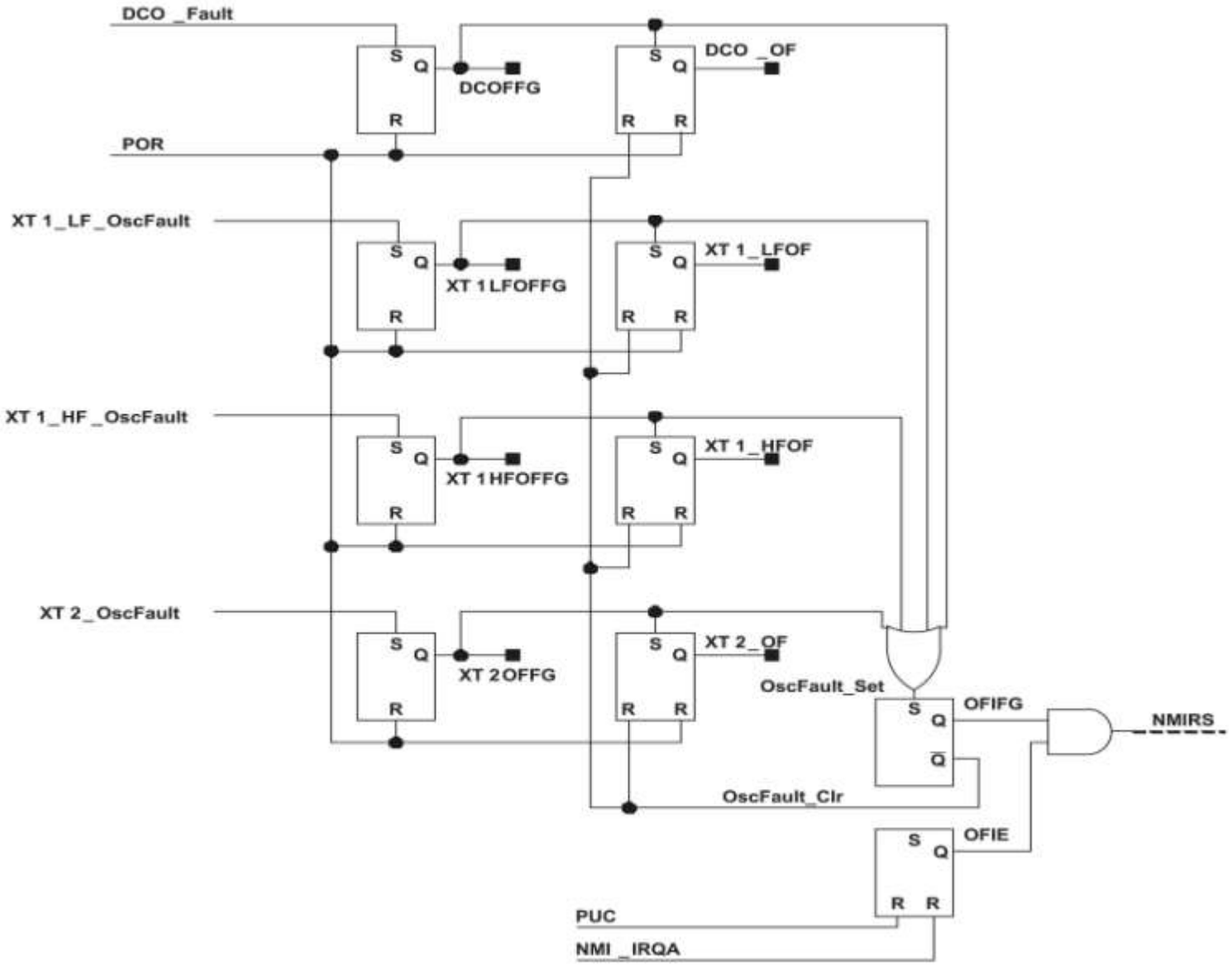
- 时钟系统模块包含有晶振失效保护的功能，可检测XT1、XT2、DCO的振荡器故障。
- 当晶体振荡器启用后，没有正常工作时，晶振失效保护工作，则将置相应的故障位：

  XT1LFOFFG：XT1的LF模式下晶振失效

  XT1HFOFFG：XT1的HF模式下晶振故障

  XT2OFFG：高频晶振故障

  DCOFFG：DCO故障标志


DCO _Fault
S
Q
R
DCOFFG
DCO _OF
POR
XT 1_LF _OscFault
XT 1 LFOFFG
XT 1_LFOF
XT 1_HF _OscFault
XT 1 HFOFFG
XT 1_HFOF
XT 2_OscFault
XT 2OFFG
XT 2_OF
OscFault_Set
OFIFG
NMIRS
OscFault_Clr
OFIE
PUC
NMI _IRQA

晶体振荡器失效逻辑图

- 如果检测出振荡器失效，置OFIFG=1 (Oscillator Fault)
  如果OFIFG标志对应的分中断控制位OFIE=1,
  则发出非屏蔽中断申请，CPU执行完当前指令，
  响应该中断，转去执行类型61的中断程序。

类型	中断源	中断标志	向量地址	优先级类型号
复位	上电,外部复位, 看门狗复位, FLASH密码错	WDTIFG, KEYV	0FFFEh	63 (highest)
…	…	…	…	…
用户 非屏蔽	NMI引脚, 振荡器失效, FCTlx访问错	NMIFG OFIFG ACCVIFG	0FFFAh	61
可 屏 蔽	…	…	…	…
	P1.0~P1.7的8个引脚	P1IFG.0~P1IFG.7	0FFDEh	47
	…	…	…	…
	P2.0~P2.7的8个引脚	P2IFG.0~P2IFG.7	0FFD8h	44

- 当检测出振荡器失效，系统会自动做出时钟切换，
  确保在振荡器失效的情况下，程序可以继续执行。

If a fault is detected for the oscillator sourcing MCLK, MCLK is automatically switched to the DCO for itsclock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If MCLK is sourced from XT1 in LFmode, an oscillator fault causes MCLK to be automatically switched to the REFO for its clock source(REFOCLK). This does not change the SELM bit settings. This condition must be handled by usersoftware.

If a fault is detected for the oscillator sourcing SMCLK, SMCLK is automatically switched to the DCO forits clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If SMCLK is sourced from XT1in LF mode, an oscillator fault causes SMCLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELS bit settings. This condition must be handled by user software.

If a fault is detected for the oscillator sourcing ACLK, ACLK is automatically switched to the DCO for itsclock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If ACLK is sourced from XT1 in LF mode, an oscillator fault causes ACLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELA bit settings. This condition must be handled by user software.

SFR特殊功能寄存器 special function register(16位寄存器）

SFRIFG1中断标志寄存器1 （Interrupt flag register 1)

15~8, 5, 2 这10位均为保留位。

7	6	5	4	3	2	1	0
JMBOUTIFG	JMBINIFG		NMIFG	VMAIFG		OFIFG	WDTIFG
rw-1	rw-0		rw-0			rw-1	rw-(0)

如果检测出振荡器失效
置OFIFG=1 (Oscillator Fault)

SFRIE1中断允许寄存器1 （Interrupt enble 1)

15~8, 2 这9位均为保留位。

7	6	5	4	3	2	1	0
JMBOUTIE	JMBINIE	ACCVIE	NMIE	VMAIE		OFIE	WDTIE
rw-0	rw-0	rw-0	rw-0			rw-0	rw-0

振荡器失效的分中断控制位

OFIE=1, 允许振荡器失效发出非屏蔽中断申请,
CPU执行完当前指令,转去执行类型61的中断程序

五、 时钟系统模块设置举例

```
void initClock( )
{   while(BAKCTL & LOCKIO) //解锁XT1引脚
    BAKCTL &= ~(LOCKIO);
   UCSCTL6 &= ~XT1OFF;                    //启动XT1
   P7SEL |= BIT2 + BIT3;                  //XT2引脚功能选择
   UCSCTL6 &= ~XT2OFF;                    //启动XT2
   while (SFRIFG1 & OFIFG) {              //等待XT1、XT2与DCO稳定
    UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
    SFRIFG1 &= ~OFIFG;
        }
  UCSCTL4 = SELA__XT1CLK + SELS__XT2CLK + SELM__XT2CLK; //避免DCO调整中跑飞
  UCSCTL1 = DCORSEL_5;       //6000kHz~23.7MHz
  UCSCTL2 = 20000000 / (4000000 / 16);   //XT2频率较高，分频后作为基准可获得更高的精度
  UCSCTL3 = SELREF__XT2CLK + FLLREFDIV__16; //XT2进行16分频后作为基准
  while (SFRIFG1 & OFIFG)      {         //等待XT1、XT2与DCO稳定
    UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
    SFRIFG1 &= ~OFIFG;
     }
  UCSCTL5 = DIVA__1 + DIVS__1 + DIVM__1; //设定几个CLK的分频
  UCSCTL4 = SELA__XT1CLK + SELS__DCOCLK + SELM__DCOCLK; //设定几个CLK的时钟源
}
```

第3节　MSP430 的低功耗模式
(Low Power Modes)

一、低功耗控制

二、MSP430工作模式

三、低功耗模式的进入与退出

四、低功耗模式编程举例

一、低功耗控制

MSP430 是专门为超低功耗应用而设计的，
有多种运行方式，不同的运行方式有不同的芯片功耗，
使用者可以根据应用要求来选择相应的运行方式。

- 工作频率越低，**MCU的功耗越小**
- 工作电压越低，**MCU的功耗越小**
- 工作模块越少，**MCU的功耗越小**

当系统时钟发生器的基本功能建立后，
CPU中状态寄存器SR中的SCG1、SCG0、OSCoff、CPUoff位
是重要的低功耗控制位

状态寄存器SR的低功耗控制位

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCOFF	CPUOFF	GIE	N	Z	C

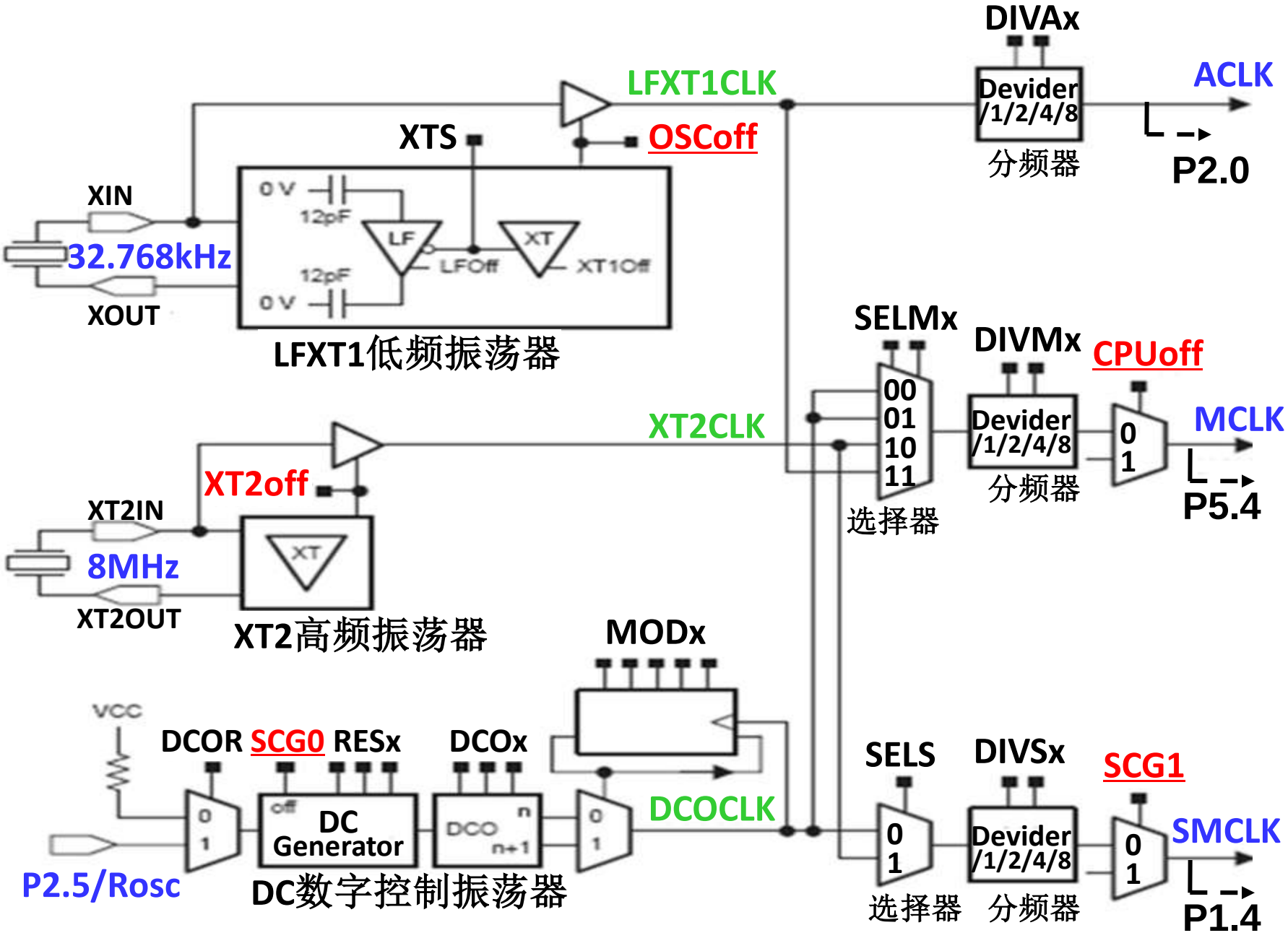
SCG1：系统时钟发生器控制位1 (System clock generator 1)
置位时关闭SMCLK

SCG0：系统时钟发生器控制位0 (System clock generator 0)
置位时关闭DC发生器

OSCoff: 晶振控制位(Oscillator Off Bit)
置位时关闭LFXT1振荡器

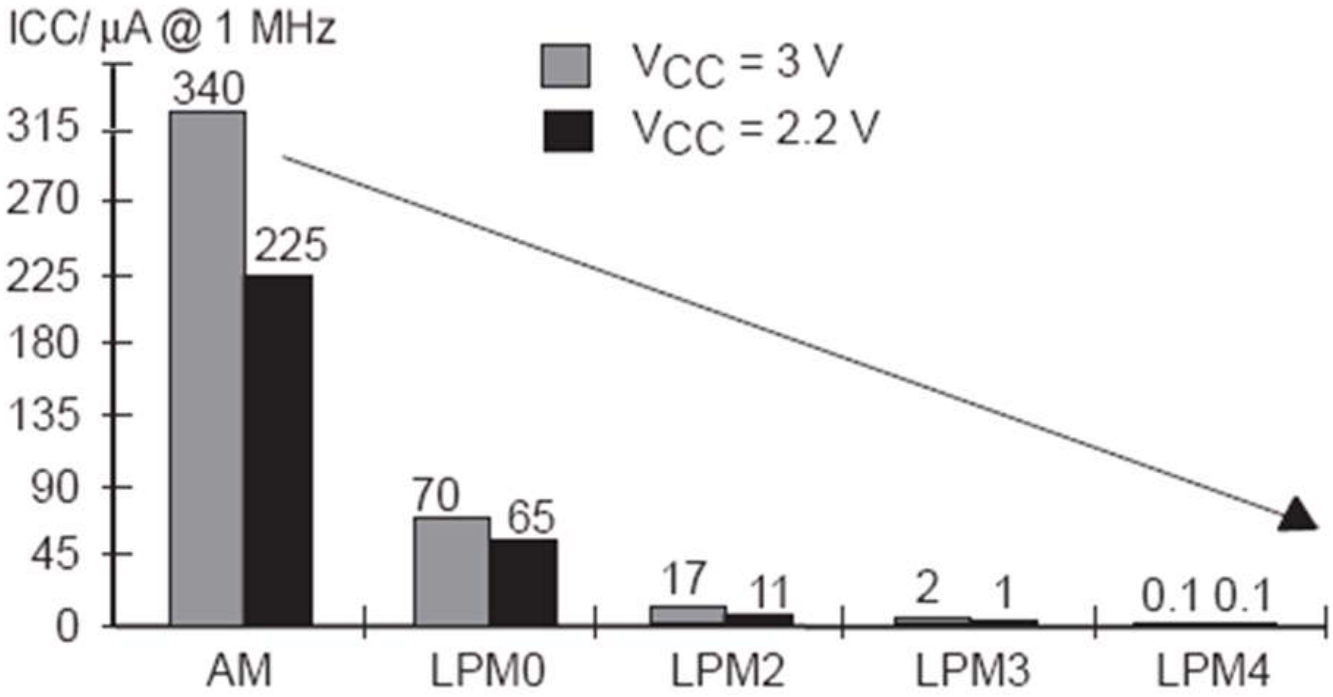
CPUoff: CPU 控制位(CPU Off Bit)
置位时关闭MCLK，即使CPU 进入关闭模式
此时除了RAM 内容、端口、寄存器保持外，
CPU 处于停止状态，由中断将CPU 从此状态唤醒

MSP430F1xx基本时钟模块结构图



二、MSP430工作模式

- MSP430F663x有7种不同的工作模式：1种活动模式AM和6种低功耗模式LPM0~LPM4(Low Power Mode)、LPM3.5、LPM4.5
- 通过设置SR中的控制位可从活动模式进入到低功耗模式，各种低功耗模式可通过中断回到活动模式



MSP430x14x不同工作模式下的典型工作电流

- MSP430内部各模块运行完全是独立的
  TA、TB、输入/输出端口、A/D、WDT等均可在CPU休眠的状态下独立运行;
  各片内模块也可通过禁止相应寄存器中控制位来关闭。

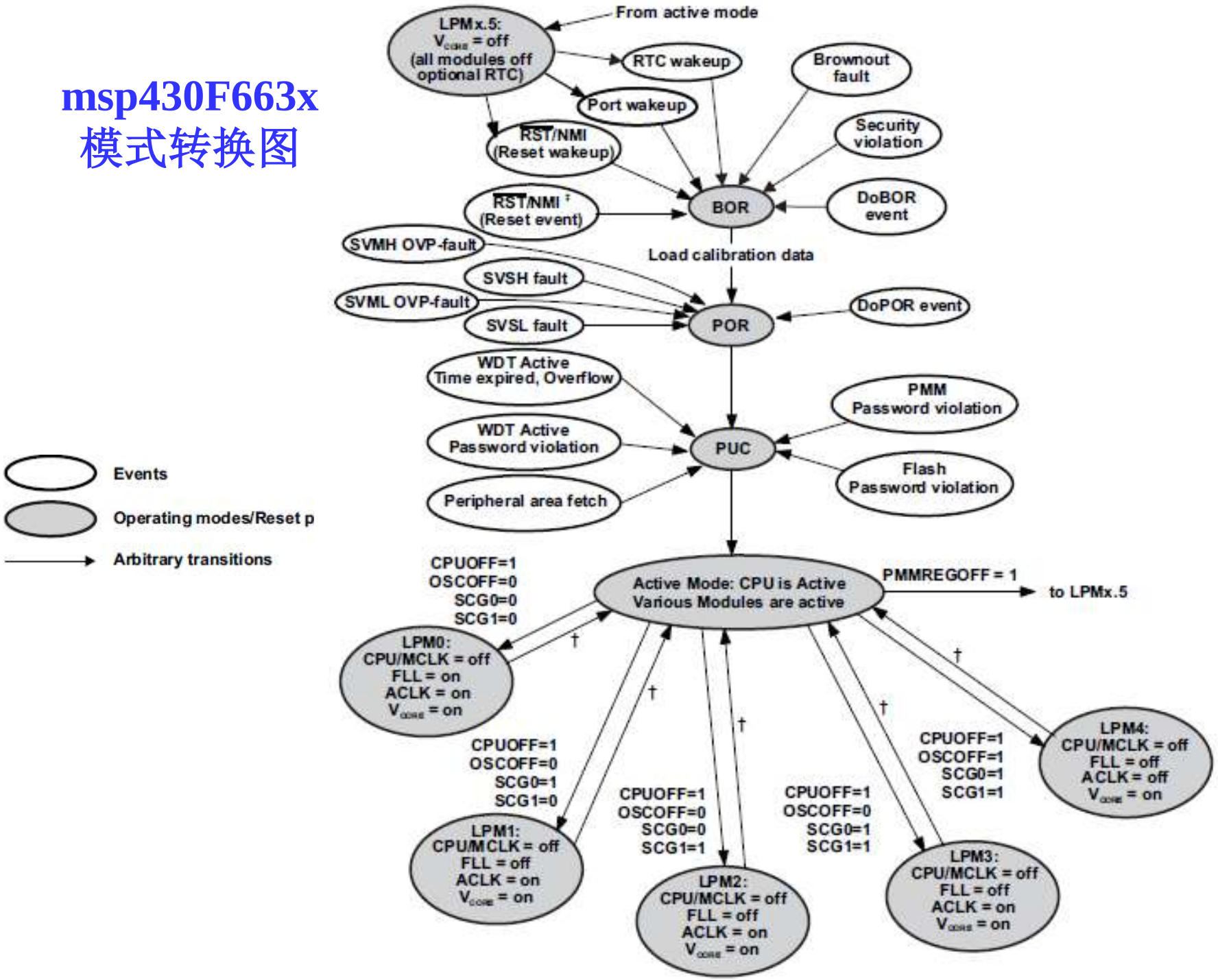
- 一旦改变了 SR 中的模式控制位，工作模式便立即改变;
  有关的模块也因为相应的时钟源被禁止而被关闭。

- 改变模式不影响所有的I/O引脚及RAM/寄存器的值。

各状态标志的含义

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCoff	CPUoff	GIE	N	Z	C

msp430F663x
模式转换图



三、低功耗模式的进入与退出

- 在活动模式下, 按低功耗模式设定SR 中的控制位,
  MSP430就进入设定的低功耗模式,
  此时CPU寄存器的值保持不变,
  有关的模块也因为相应的时钟源被禁止而被关闭。

- 任意中断均可唤醒处于低功耗模式的MSP430，
  使MSP430切换到AM活动模式，即退出低功耗模式。

低功耗退出过程(与中断响应过程一样)

处于低功耗模式下的MSP430,
当有N类型号的非屏蔽或可屏蔽中断源产生,
满足响应条件,
CPU硬件自动完成下面操作：

1. 入栈保护 当前PC;
2. 入栈保护当前SR
3. 清零SR

   (置GIE=0,屏蔽可屏蔽中断,

   清SCG1、SCG0、OSCoff、CPUoff , 结束低功耗方式,

   切换到活动模式)
4. 从中断向量表取中断向量至PC
5. 转去执行中断服务子程

中断程序执行完毕，执行到RETI 返回指令时：

1. 出栈恢复 SR和PC值
2. 由于恢复了进入中断前SR中的SCG1,SCG0,OSCoff,CPUoff值，使MSP430切换回到原来的低功耗模式；
3. 利用在MSP430F6638.h的定义，
   可在中断子程返回前，调用LPMx_EXIT，退出低功耗模式。

msp430F663x.h

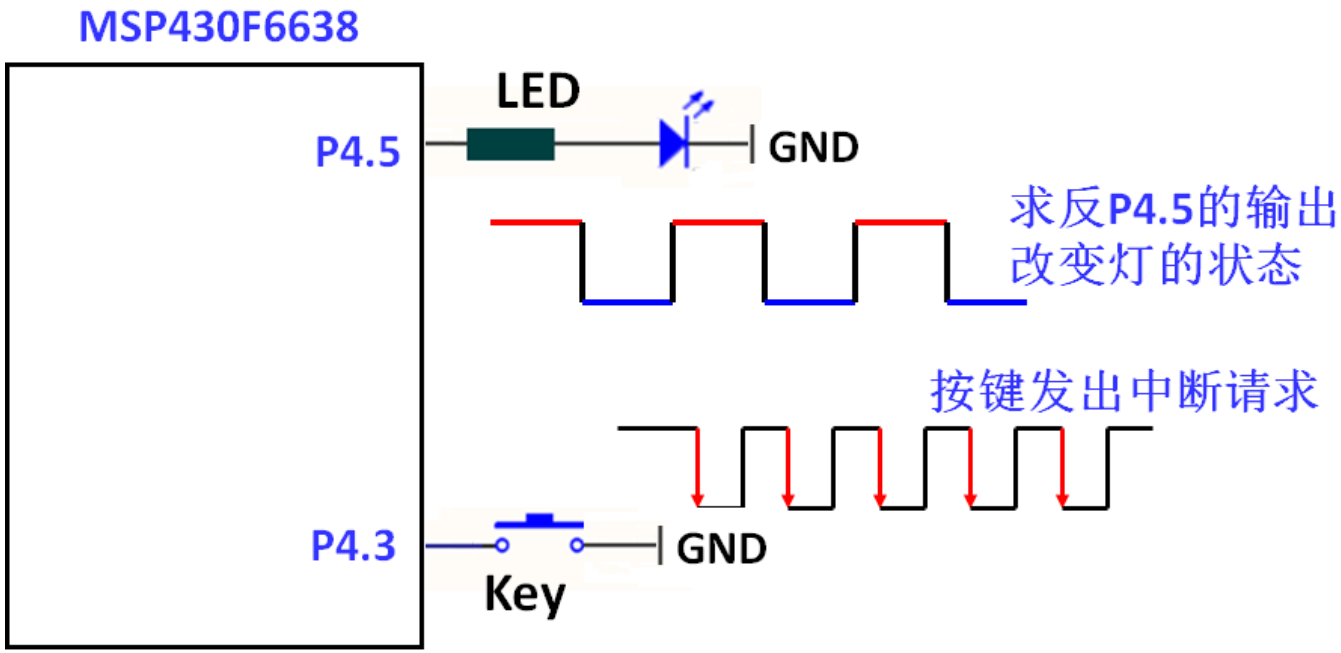
```
#define LPM0_bits         (CPUOFF)
#define LPM1_bits         (SCG0+CPUOFF)
#define LPM2_bits         (SCG1+CPUOFF)
#define LPM3_bits         (SCG1+SCG0+CPUOFF)
#define LPM4_bits         (SCG1+SCG0+OSCOFF+CPUOFF)

#include "in430.h"

#define LPM0       _bis_SR_register(LPM0_bits)       /* Enter Low Power Mode 0 */
#define LPM0_EXIT  _bic_SR_register_on_exit(LPM0_bits) /* Exit Low Power Mode 0 */
#define LPM1       _bis_SR_register(LPM1_bits)       /* Enter Low Power Mode 1 */
#define LPM1_EXIT  _bic_SR_register_on_exit(LPM1_bits) /* Exit Low Power Mode 1 */
#define LPM2       _bis_SR_register(LPM2_bits)       /* Enter Low Power Mode 2 */
#define LPM2_EXIT  _bic_SR_register_on_exit(LPM2_bits) /* Exit Low Power Mode 2 */
#define LPM3       _bis_SR_register(LPM3_bits)       /* Enter Low Power Mode 3 */
#define LPM3_EXIT  _bic_SR_register_on_exit(LPM3_bits) /* Exit Low Power Mode 3 */
#define LPM4       _bis_SR_register(LPM4_bits)       /* Enter Low Power Mode 4 */
#define LPM4_EXIT  _bic_SR_register_on_exit(LPM4_bits) /* Exit Low Power Mode 4 */
#endif /* End #defines for C */
```

四、低功耗模式举例

用低功耗方式改写第5章的中断例程，

在无限死循环while(1){ }前， 进入低功耗模式LPM0，

比较改写前后程序的执行流程有何不同？



Msp430F6638.h头文件

```
/* STATUS REGISTER BITS */
#define   C             (0x0001)
#define   Z             (0x0002)
#define   N             (0x0004)
#define   V             (0x0100)
#define   GIE           (0x0008)
#define   CPUOFF        (0x0010)
#define   OSCOFF        (0x0020)
#define   SCG0          (0x0040)
#define   SCG1          (0x0080)
```

状态寄存器SR (Status Register)

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCOFF	CPUOFF	GIE	N	Z	C

in430.h
声明可以操作状态寄存器SR的内敛函数

```
unsigned short _bic_SR_register(unsigned short mask);
unsigned short _bic_SR_register_on_exit(unsigned short mask);
unsigned short _bis_SR_register(unsigned short mask);
unsigned short _bis_SR_register_on_exit(unsigned short mask);
unsigned short _get_SR_register(void);
unsigned short _get_SR_register_on_exit(void);
```

状态寄存器SR (Status Register)

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCOFF	CPUOFF	GIE	N	Z	C

msp430F6638.h头文件

```
#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)

#include "in430.h"

#define LPM0       _bis_SR_register(LPM0_bits)       /* Enter Low Power Mode 0 */
#define LPM0_EXIT  _bic_SR_register_on_exit(LPM0_bits) /* Exit Low Power Mode 0 */
#define LPM1       _bis_SR_register(LPM1_bits)       /* Enter Low Power Mode 1 */
#define LPM1_EXIT  _bic_SR_register_on_exit(LPM1_bits) /* Exit Low Power Mode 1 */
#define LPM2       _bis_SR_register(LPM2_bits)       /* Enter Low Power Mode 2 */
#define LPM2_EXIT  _bic_SR_register_on_exit(LPM2_bits) /* Exit Low Power Mode 2 */
#define LPM3       _bis_SR_register(LPM3_bits)       /* Enter Low Power Mode 3 */
#define LPM3_EXIT  _bic_SR_register_on_exit(LPM3_bits) /* Exit Low Power Mode 3 */
#define LPM4       _bis_SR_register(LPM4_bits)       /* Enter Low Power Mode 4 */
#define LPM4_EXIT  _bic_SR_register_on_exit(LPM4_bits) /* Exit Low Power Mode 4 */
```

进入低功耗模式编程举例

例 进入低功耗模式LPM0

```
_bis_SR_register(LPM0_bits);
```

或 `LPM0;`

例 从中断返回后，退出低功耗模式LPM0

```
#pragma  vector=xxxx                    //置P1中断向量
__interrupt void port_int(void)         //中断子程
{       ……
        LPM0_EXIT;
}
```

例1 进入低功耗模式例程序清单

```
#include        “msp430F6638.h"
int main( void )
{       WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
        _DINT();                           //禁止可屏蔽中断 GIE=0
        P4DIR |= BIT5;                     //设置P4.5口方向为输出
        P4DIR &= ~BIT3;  P4REN |= BIT3;    //使能P4.3上拉电阻
        P4OUT |= BIT3;                     //P4.3口置高电平
        P4IES |= BIT3;                     //中断沿设置（下降沿触发）
        P4IFG &= ~BIT3;                    //清P4.3中断标志
        P4IE |= BIT3;                      //使能P4.3口中断
        _EINT();                           //开中断
        LPM0;
        while(1){ };                       //无限循环
}

#pragma vector=PORT4_VECTOR                // P4中断函数
__interrupt void Port_4(void)
{       P4OUT ^= BIT5;                     //改变LED5灯状态
        P4IFG &= ~BIT3;                    //清P4.3中断标志位
}
```

例　中断子程中利用LPM0_EXIT，使执行完中断程序后，退出LPM0低功耗。

```
#include          "io430.h"
#include          "in430.h"
int main( void )
{         WDTCTL = WDTPW + WDTHOLD;   //关闭看门狗
          _DINT();                                  //禁止可屏蔽中断 GIE=0
          P1IE_bit.P0=0;              // 关闭P1.0中断允许
          P3SEL_bit.P6=0;             //设置P3.6为基本I/O功能
          P3DIR_bit.P6=1;                       //设置P3.6为输出
          P3OUT_bit.P6=0;                       //置P3.6输出初值为0
          P1SEL_bit.P0=0;                       //置P1.0作为基本I/O端口
          P1DIR_bit.P0=0;                       //置P1.0为输入
          P1IES_bit.P0=1;                       //置P1.0下降沿作中断源
          P1IFG_bit.P0=0;             //清P1.0中断标志
          P1IE_bit.P0=1;              //打开P1.0中断允许
          _EINT();                              //允许可屏蔽中断 GIE=1
//        _BIS_SR(LPM0_bits);         //进入LPM0(编程方法1)
          LPM0;                       //进入LPM0(编程方法1)
          while(1) {  };              //主程循环
}

#pragma  vector=PORT1_VECTOR                    //置P1中断向量
__interrupt void port_int(void)                 //中断子程
{    if  (P1IFG_bit.P0==1)              //判断是否是P1IFG.0中断标志
     {       P3OUT_bit.P6=~P3OUT_bit.P6;      //对P3.6取反
             P1IFG_bit.P0=0;                                  //清P1.0中断标志
      }
          LPM0_EXIT;
}
```