第二讲

实验与创新实践教育中心
哈尔滨工业大学（深圳）

本节内容

- 第1讲回顾
- 大功率LED灯控制
- MSP430时钟系统概述
- MSP430定时器A

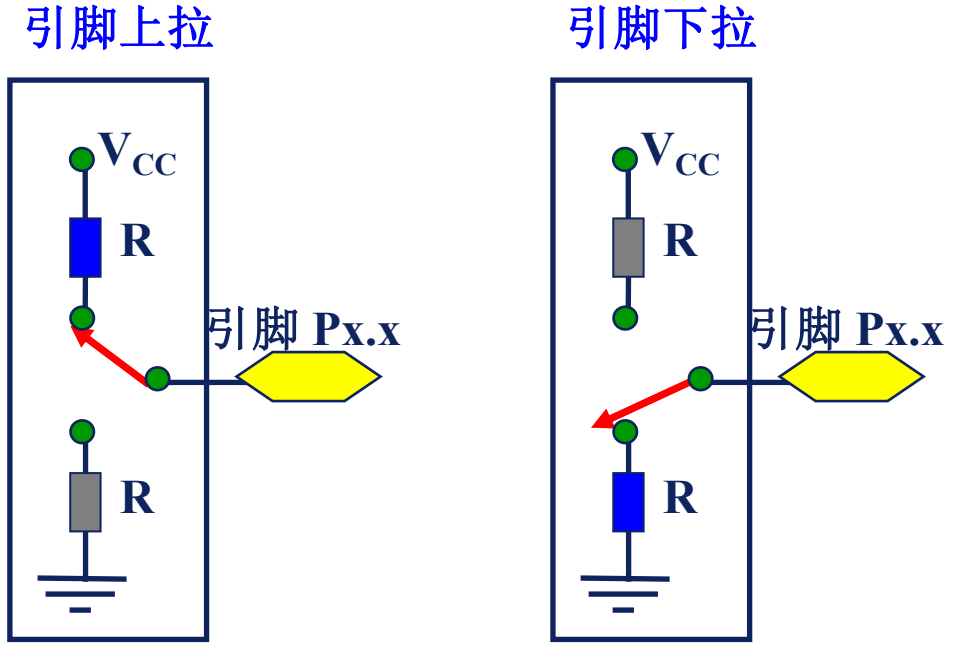
GPIO 寄存器

名称	缩写	BIT=1	BIT=0
方向寄存器	PxDIR	输出模式	输入模式
输入寄存器	PxIN	输入高电平	输入低电平
输出寄存器	PxOUT	输出高电平	输出低电平
上下拉电阻使能寄存器	PxREN	使能	禁用
功能选择寄存器	PxSEL	外设功能	IO端口
驱动强度寄存器	PxDS	高强度	低强度
中断使能寄存器	PxIE	允许中断	禁止中断
中断触发沿寄存器	PxIES	下降沿置位	上升沿置位
中断标志寄存器	PxIFG	有中断请求	无中断请求

通用控制寄存器

中断控制寄存器

上下拉电阻？



PxOUT

- **Bit = 1: 引脚选择上拉;**

上拉电阻简单来说就是把电平拉高，通常用4.7~10K的电阻接到Vcc电源。

- **Bit = 0: 引脚选择下拉;**

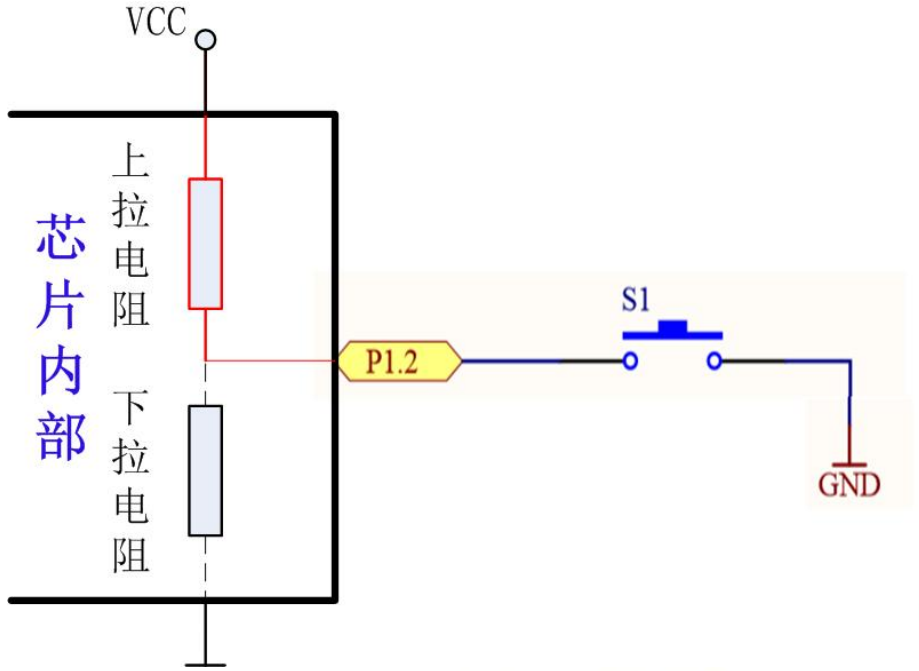
下拉电阻则是把电平拉低，电阻接到GND地线上。

P1.2的上下拉电阻使能寄存器：

P1REN置1,代码编写为 P1REN |= BIT2;

P1.2的上拉或者下拉电阻选择寄存器（输出寄存器）：

P1OUT置1,代码编写为 P1OUT |= BIT2;



寄存器位操作

与（&）	0 & 0 = 0	1 & 0 = 0	0 & 1 = 0	1 & 1 = 1
或（|）	0 | 0 = 0	1 | 0 = 1	0 | 1 = 1	1 | 1 = 1
异或（^）	0 ^ 0 = 0	1 ^ 0 = 1	0 ^ 1 = 1	1 ^ 1 = 0

每一位代表一个IO引脚

```
P1OUT |= BIT0;        //P1.0置1
P1OUT &= ~BIT0;       //P1.0清零
P1OUT ^= BIT0;        //P1.0取反
P1IN&BIT0             //读取P1.0的值
```

字节操作
与位操作
的区别？

```
P1OUT = BIT0;         //P1.0置1, P1.1~P1.7清零
P1OUT |= BIT0;        //P1.0置1
```

中断函数（中断服务程序）

中断向量，名称参考示例程序或头文件

```
#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
  IE1 &= ~WDTIE;                    // disable interrupt
  IFG1 &= ~WDTIFG;                  // clear interrupt flag
  WDTCTL = WDTPW + WDTHOLD;         // put WDT back in hold state
  BUTTON_IE |= BUTTON;              // Debouncing complete
}
```

用户自定义的函数名

中断服务子程序

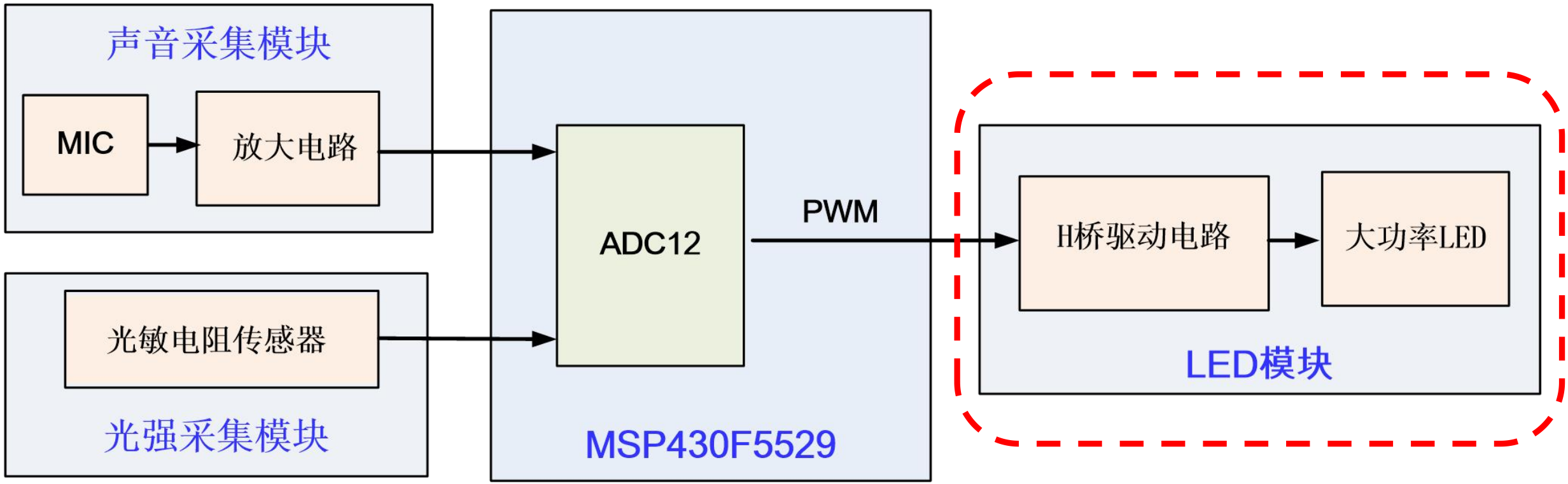
```
#include <msp430.h>
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD;
        P8DIR|=BIT1;  //p8.1 output
        P8OUT&=~BIT1;

        P1DIR&=~BIT2;  //p1.2 input,pull up
        P1REN|=BIT2;
        P1OUT|=BIT2;

        P1IE|=BIT2; //enable P1.2 interrupt
        P1IES|=BIT2; //high-low transition
        P1IFG&=~BIT2;

        _enable_interrupt();

        return 0;
}
```

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    if(P1IFG&BIT2){
        P8OUT^=BIT1;
        P1IFG&=~BIT2;
    }

    if(P1IFG&BIT3){
        ; //?
        ; //?
    }
}

#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{


}
```

本节内容

- 第1讲回顾
- 大功率LED灯控制
- MSP430时钟系统概述
- MSP430定时器A

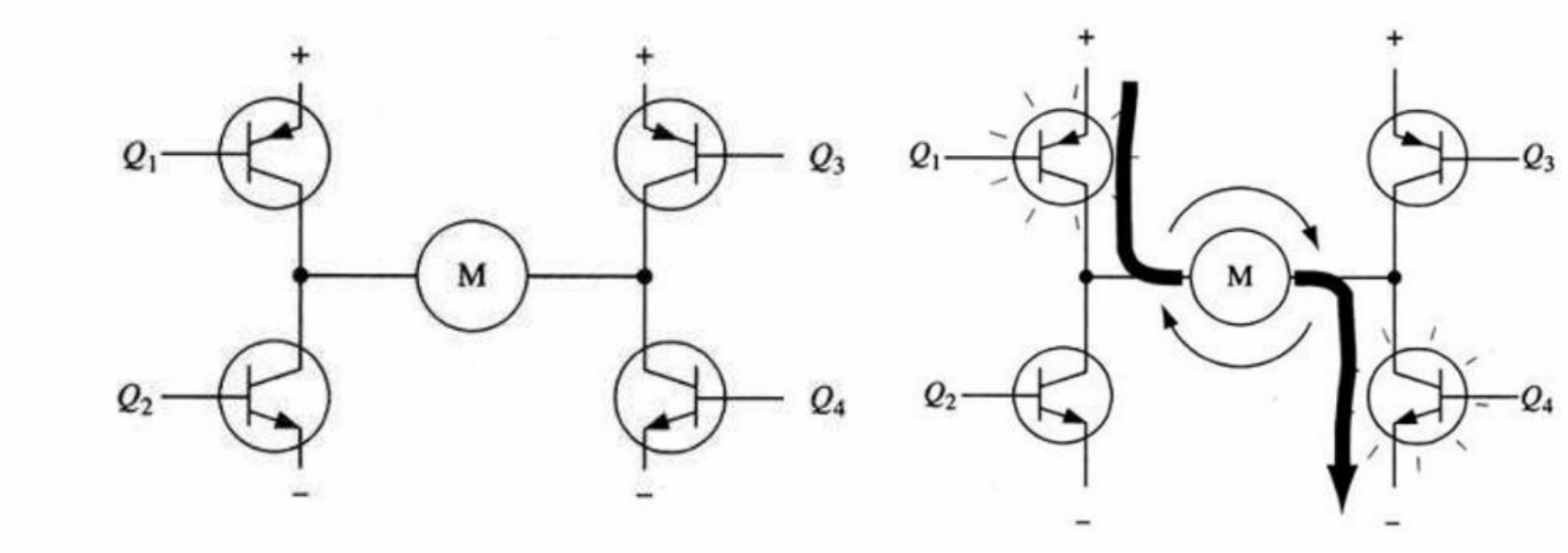
展馆灯光控制系统功能需求分析-大功率LED灯点灯灭灯



大功率LED驱动控制

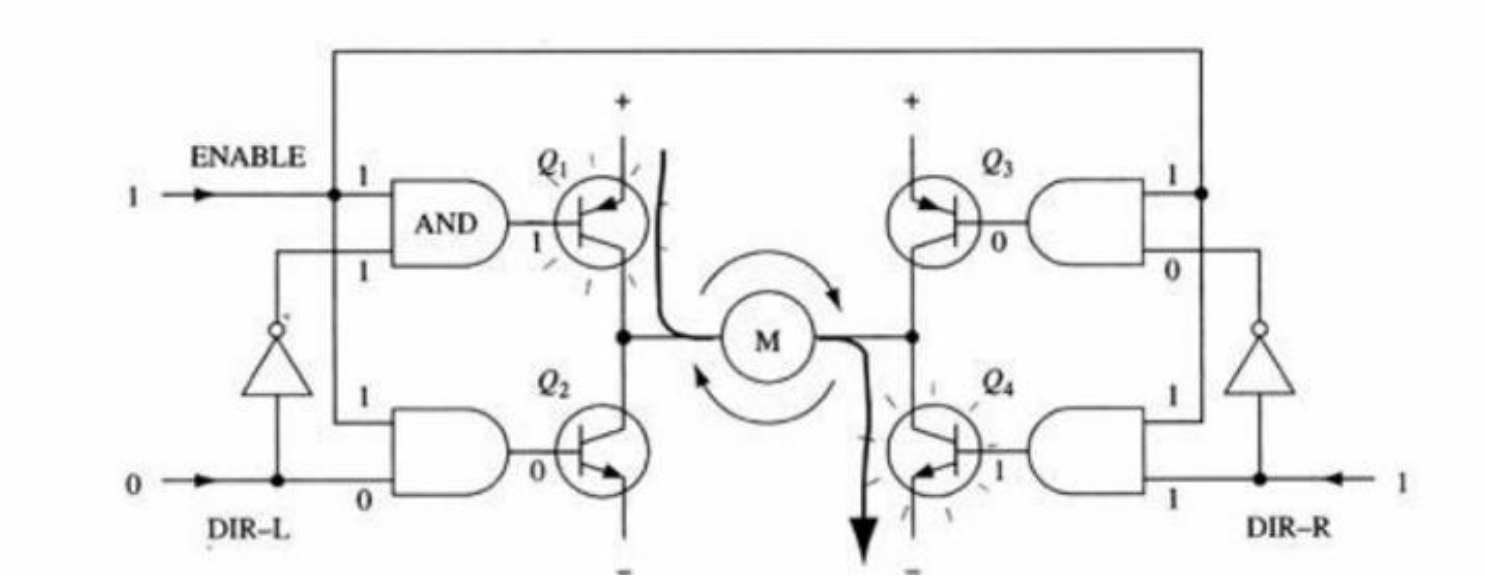
在单片机运用中，对于需要较大电流的元件，往往使用三极管来驱动，如数码管、蜂鸣器等。

在MSP430F5529 POCKET KIT中，典型的驱动电路最就是 H 桥驱动电路。



大功率LED驱动控制

保证 H 桥上两个同侧的三极管不会同时导通非常重要。如果三极管 Q1、Q2 同时导通，那么电流就会从正极穿过两个三极管直接回到负极。此时，电路中除了三极管外没有其他任何负载，因此电路上的电流就可能达到最大值（该电流仅受电源性能限制），甚至烧坏三极管。

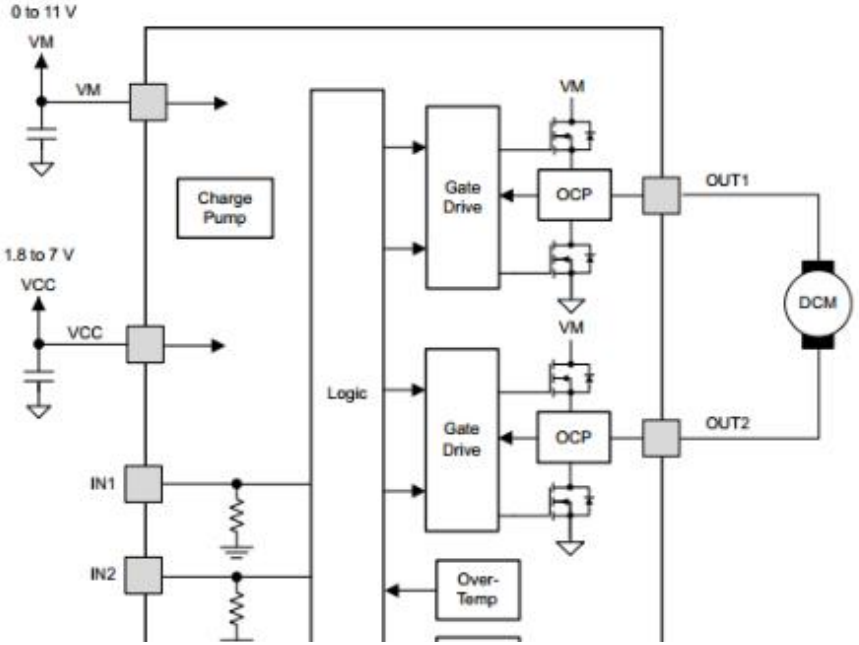


大功率LED驱动控制

口袋板上使用了TI DRV8837低电压电机驱动芯片。

芯片工作电压范围VCC：1.8 V—7 V

独立的电机电源VM ： 0—11V



drv8837.pdf

Table 1. DRV8837 Device Logic

nSLEEP	IN1	IN2	OUT1	OUT2	FUNCTION (DC MOTOR)
0	X	X	Z	Z	Coast
1	0	0	Z	Z	Coast
1	0	1	L	H	Reverse
1	1	0	H	L	Forward
1	1	1	L	L	Brake

大功率LED驱动控制

跳线帽短接

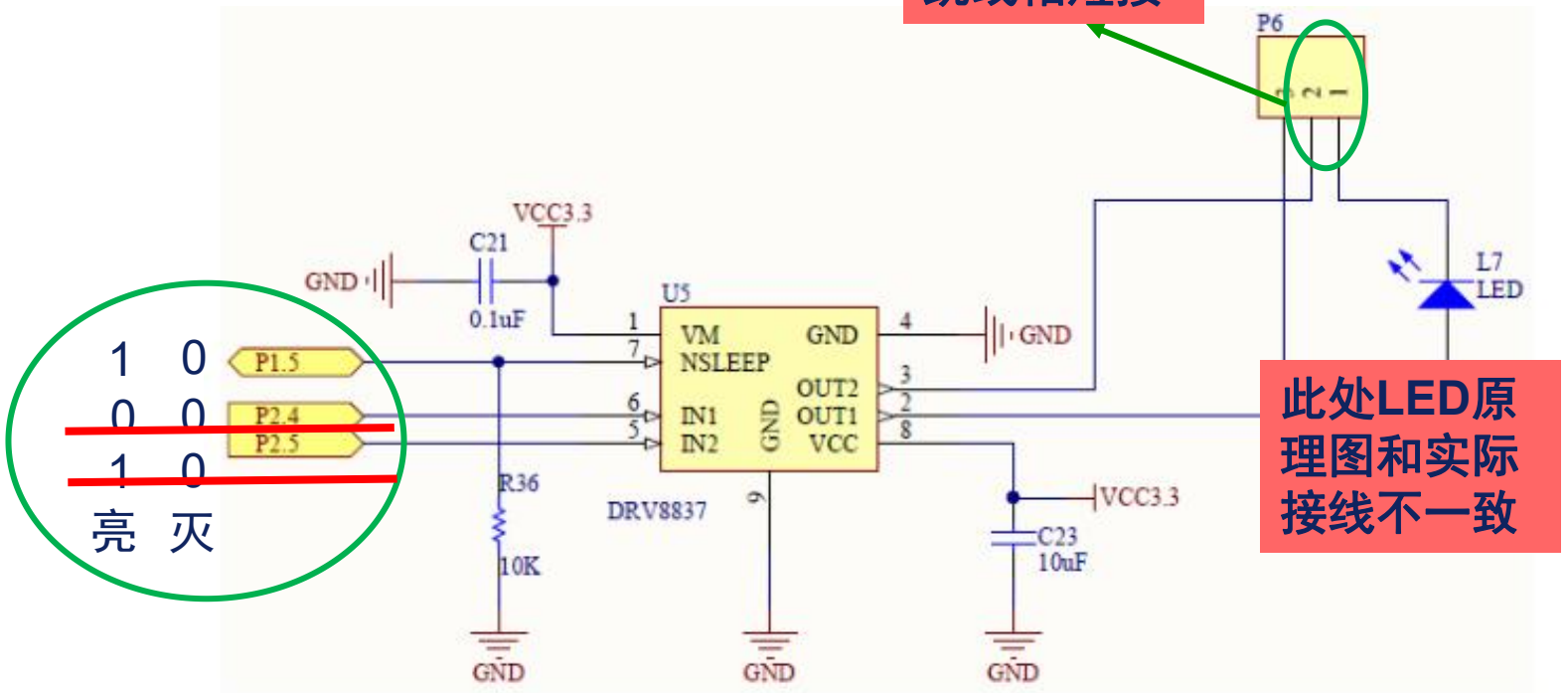
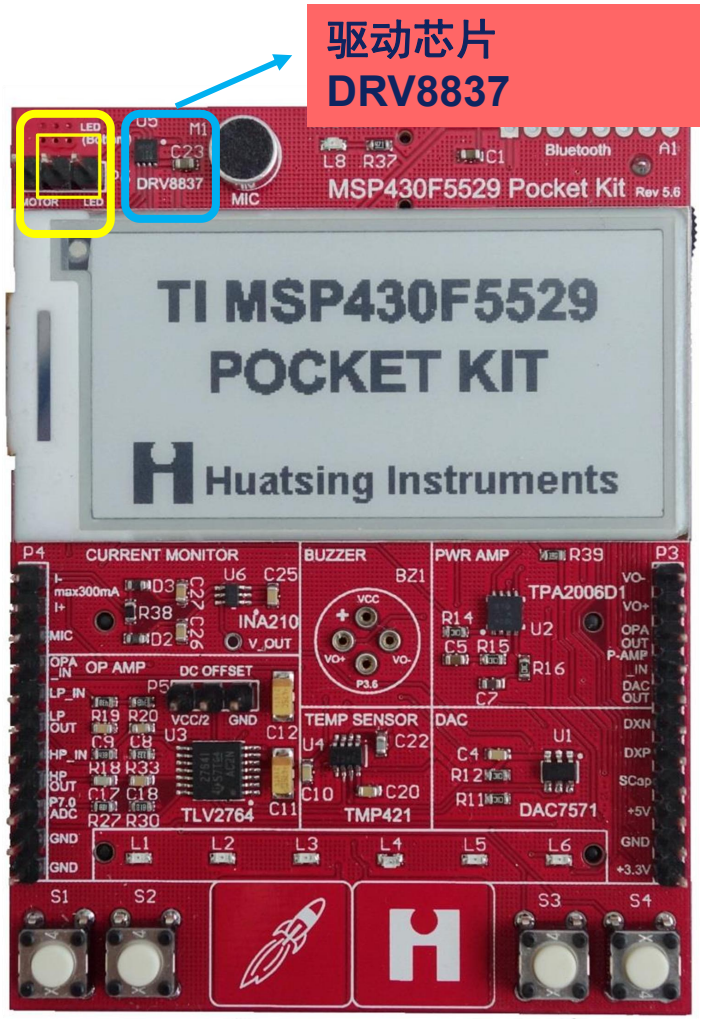


Table 1. DRV8837 Device Logic

nSLEEP	IN1	IN2	OUT1	OUT2	FUNCTION (DC MOTOR)
0	X	X	Z	Z	Coast
1	0	0	Z	Z	Coast
1	0	1	L	H	Reverse
1	1	0	H	L	Forward
1	1	1	L	L	Brake

灭

电路连接：用跳线帽短接图上插针



课堂实验2.1

- 请实现大功率LED灯的点灯、灭灯功能
- 方法不限（可用学过的查询IO引脚高低电平状态的方法，或者使用GPIO中断的方法）
- 实现逻辑不限制（可自动点灯灭灯循环，可手动控制点灯灭灯）

练一练

```
#include <msp430.h>
int main(void)
{
        // stop watchdog timer
        WDTCTL = WDTPW | WDTHOLD;
        // 大功率LED灯初始化
        **************;     //p1.5 output
        **************;     //p2.4、p2.5 output

        //按键初始化
        P1DIR  &=~BIT2;  //p1.2 input,pull up
        P1REN |= (BIT2+BIT3);
        P1OUT |= (BIT2+BIT3);

        //按键的中断初始化
        P1IE|=BIT2+BIT3;    //enable P1.2 interrupt
        P1IES|=BIT2+BIT3; //high-low transition
        P1IFG&=~(BIT2+BIT3);

        _enable_interrupt();   //  GIE开启
        return 0;
}
```

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    if(P1IFG&BIT2){
        **************.
        **************;
        **************.
        **************;
        P1IFG&=~BIT2;
    }

    if(P1IFG&BIT3){
        **************.
        **************;
        **************.
        **************;
        P1IFG&=~BIT3;
    }

}
```

点灯

灭灯

参考答案

```
#include <msp430.h>
int main(void)
{
        // stop watchdog timer
        WDTCTL = WDTPW | WDTHOLD;
        // 大功率LED灯初始化
        P1DIR|=BIT5;                  //p1.5 output
        P2DIR|= BIT4+BIT5;     //p2.4、p2.5 output

        //按键中断初始化
        P1DIR&=~BIT2;  //p1.2 input,pull up
        P1REN|=(BIT2+BIT3);
        P1OUT|=(BIT2+BIT3);

        P1IE|=BIT2+BIT3;    //enable P1.2 interrupt
        P1IES|=BIT2+BIT3; //high-low transition
        P1IFG&=~(BIT2+BIT3);

        _enable_interrupt();
        return 0;
}
```

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    if(P1IFG&BIT2){
        P1OUT|=BIT5;
        P2OUT&= ~BIT4;
        P2OUT|=BIT5;
        P1IFG&=~BIT2;
    }

    if(P1IFG&BIT3){
        P1OUT&=~BIT5;
        P2OUT&= ~BIT4;
        P2OUT&=~BIT5;
        P1IFG&=~BIT3;
    }

}
```

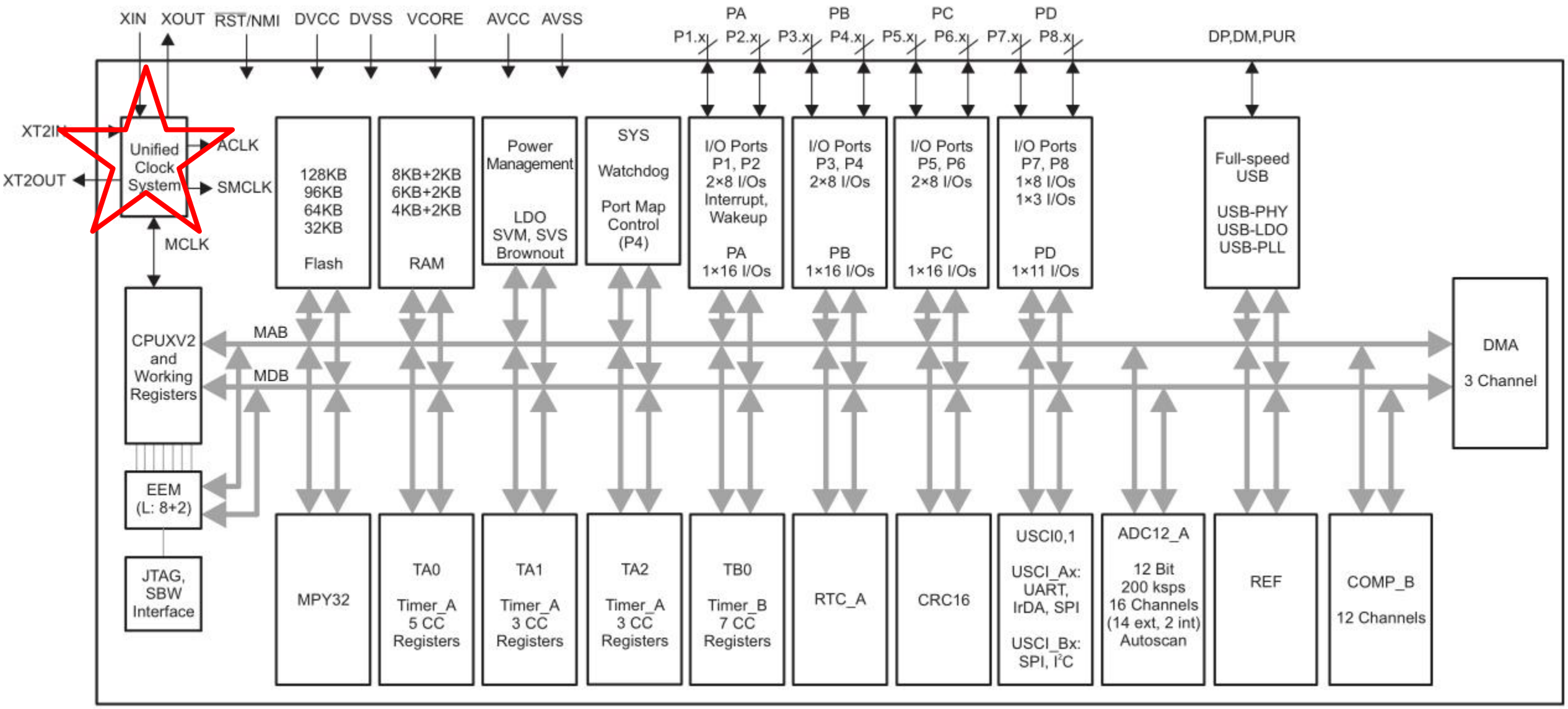
点灯

灭灯

本节内容

- 第1讲回顾
- 大功率LED灯控制
- MSP430时钟系统概述
- MSP430定时器A

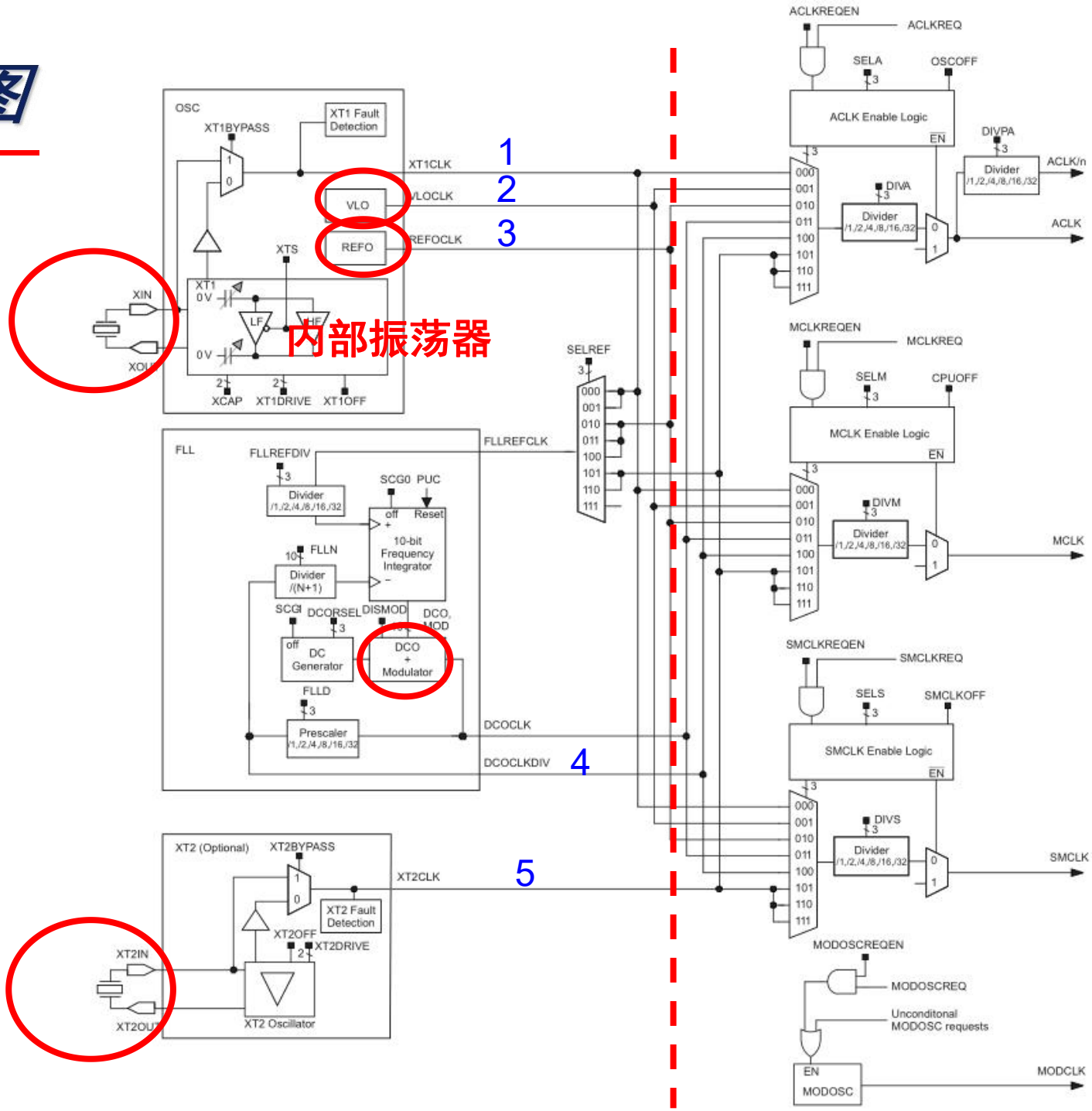
5529功能框图


XIN
XOUT
RST/NMI
DVCC
DVSS
VCORE
AVCC
AVSS
PA
PB
PC
PD
P1.x
P2.x
P3.x
P4.x
P5.x
P6.x
P7.x
P8.x
DP,DM,PUR
XT2IN
XT2OUT
Unified Clock System
ACLK
SMCLK
MCLK
128KB
96KB
64KB
32KB
Flash
8KB+2KB
6KB+2KB
4KB+2KB
RAM
Power Management
LDO
SVM, SVS
Brownout
SYS
Watchdog
Port Map Control
(P4)
I/O Ports
P1, P2
2×8 I/Os
Interrupt, Wakeup
PA
1×16 I/Os
I/O Ports
P3, P4
2×8 I/Os
PB
1×16 I/Os
I/O Ports
P5, P6
2×8 I/Os
PC
1×16 I/Os
I/O Ports
P7, P8
1×8 I/Os
1×3 I/Os
PD
1×11 I/Os
Full-speed USB
USB-PHY
USB-LDO
USB-PLL
CPUXV2 and Working Registers
MAB
MDB
DMA
3 Channel
EEM
(L: 8+2)
JTAG, SBW Interface
MPY32
TA0
Timer_A
5 CC Registers
TA1
Timer_A
3 CC Registers
TA2
Timer_A
3 CC Registers
TB0
Timer_B
7 CC Registers
RTC_A
CRC16
USCI0,1
USCI_Ax: UART, IrDA, SPI
USCI_Bx: SPI, I²C
ADC12_A
12 Bit
200 ksps
16 Channels
(14 ext, 2 int)
Autoscan
REF
COMP_B
12 Channels

时钟系统框图

外部振荡器XT1

外部振荡器2

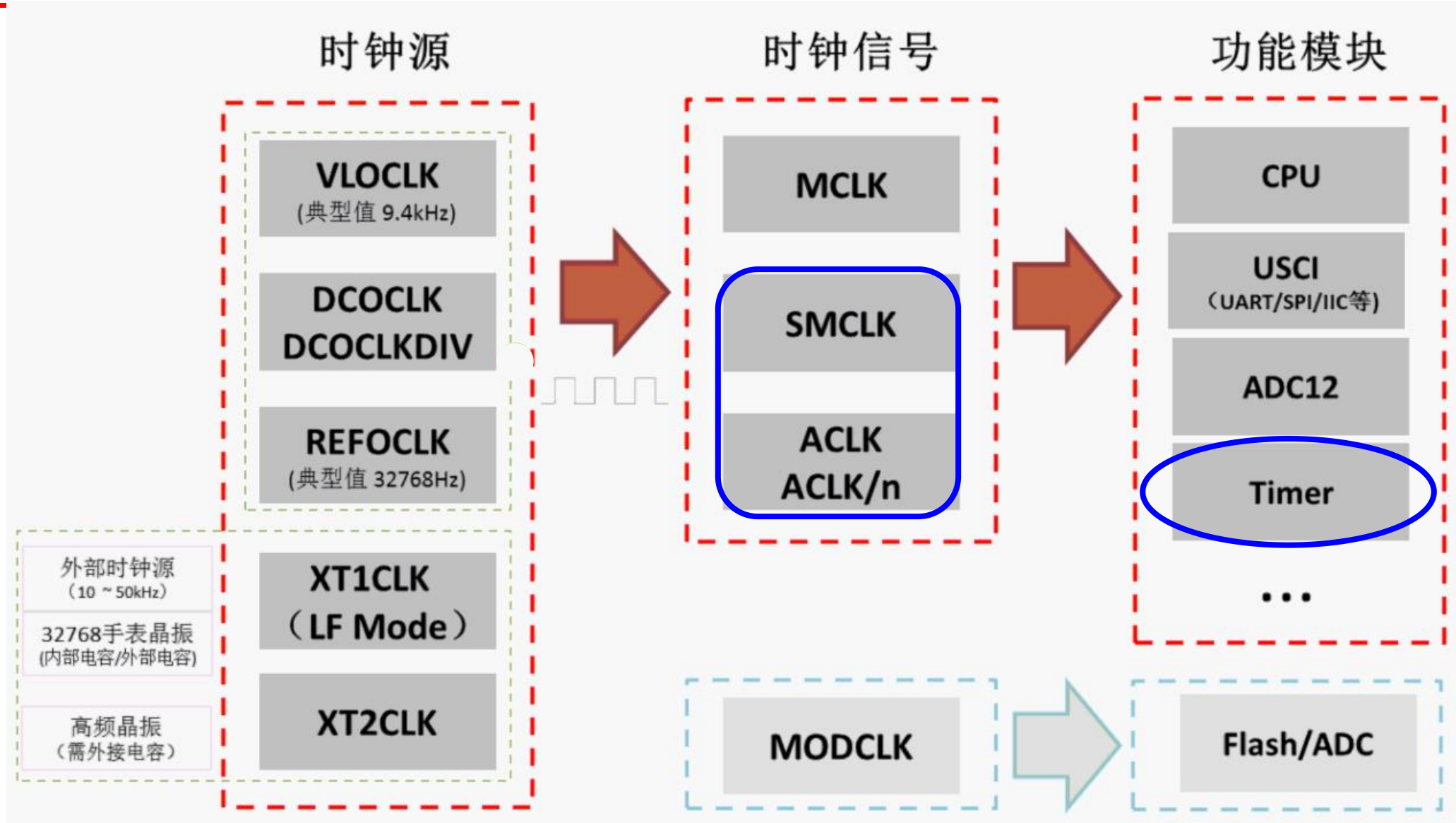


1 ACLK

2 MCLK

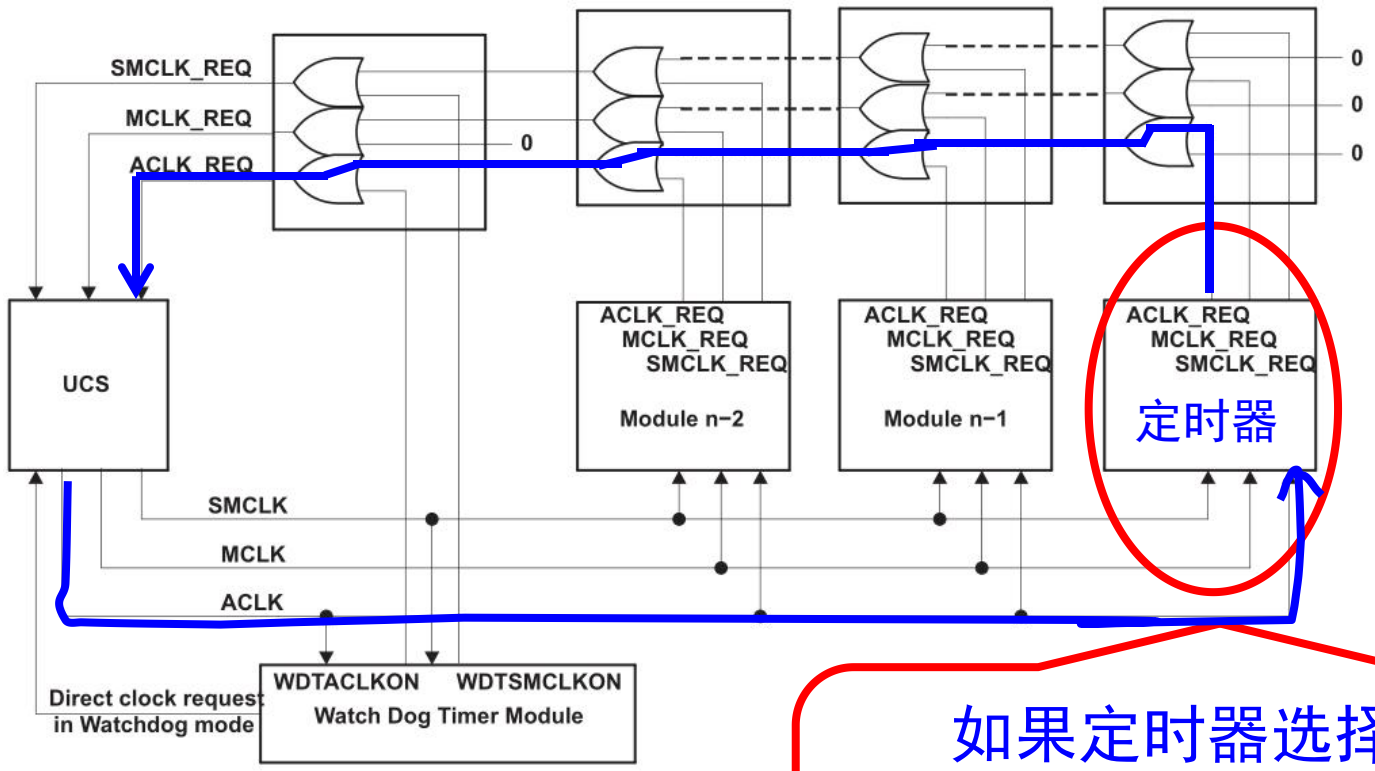
3 SMCLK

5529时钟系统简图



定时器A的时钟请求

外设模块如何获得时钟？



- 通过3个时钟请求信号来获得时钟：ACLK_REQ、MCLK_REQ、SMCLK_REQ。
- 在任何模式下，外设模块的正常操作都可以从时钟系统（UCS）请求时钟信号。

如果定时器选择了**ACLK**作为时钟源，只要定时器允许，**ACLK_REQ**信号就一直有效并向UCS申请时钟，而UCS则不管当前是在什么工作模式，都会输出**ACLK**信号。

时钟系统上电默认状态

- XT1处于LF模式，作为时钟源XT1CLK，ACLK选择XT1CLK（32768Hz）
- MCLK选择DCOCLKDIV（约1MHz）
- SMCLK选择DCOCLKDIV（约1MHz）
- 启用FLL，并将XT1CLK作为FLL参考时钟，即FLLREFCLK来自于XT1CLK
- XIN, XOUT引脚作为通用IO，XT1保持禁用
- XT2IN, XT2OUT引脚作为通用IO，XT2保持禁用

上电默认： MCLK和SMCLK的频率为1.048576MHz（约1MHz）

上电默认：ACLK的频率为32768Hz

```
__delay_cycles(  unsigned long cycles  )
```

演示实验2.1

默认时钟配置下，使用__delay_cycles(x)延时，实现主板LED1灯定时约1s翻转状态。

```
void main(void)
{
        WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
        P1DIR   |=  BIT0;           // configure P1.0 as output

        volatile unsigned int i;    // volatile to prevent optimization

        while(1)
        {
            P1OUT   ^=  BIT0;           // toggle P1.0
            __delay_cycles(1000000);    // delay
        }
}
```

课堂实验2.2

在调用右侧时钟设置接口后，使用__delay_cycles(x)延时，实现主板LED1灯定时约1s翻转状态。

```
void ClockInit()
{//最终MCLK:16MHZ,SMCLK:8MHZ,ACLK:32KHZ
      UCSCTL6 &= ~XT1OFF;         //启动XT1
      P5SEL |= BIT2 + BIT3;       //XT2引脚功能选择
      UCSCTL6 &= ~XT2OFF;         //打开XT2
      __bis_SR_register(SCG0);
      UCSCTL0 = DCO0+DCO1+DCO2+DCO3+DCO4;
      UCSCTL1 = DCORSEL_4;  //DCO频率范围在28.2MHZ以下
      UCSCTL2 = FLLD_5 + 1;         //D=16, N=1

      //n=8,FLLREFCLK时钟源为XT2CLK;
      //DCOCLK=D*(N+1)*(FLLREFCLK/n);
      //DCOCLKDIV=(N+1)*(FLLREFCLK/n);
      UCSCTL3 = SELREF_5 + FLLREFDIV_3;

      //ACLK的时钟源为DCOCLKDIV,
      //MCLK\SMCLK的时钟源为DCOCLK
      UCSCTL4 = SELA_4 + SELS_3 +SELM_3;

      //ACLK由DCOCLKDIV的32分频得到,
      //SMCLK由DCOCLK的2分频得到
      UCSCTL5 = DIVA_5 +DIVS_1;
}
```

本节内容

- 第1讲回顾
- 大功率LED灯控制
- MSP430时钟系统概述
- MSP430定时器A

定时器A主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

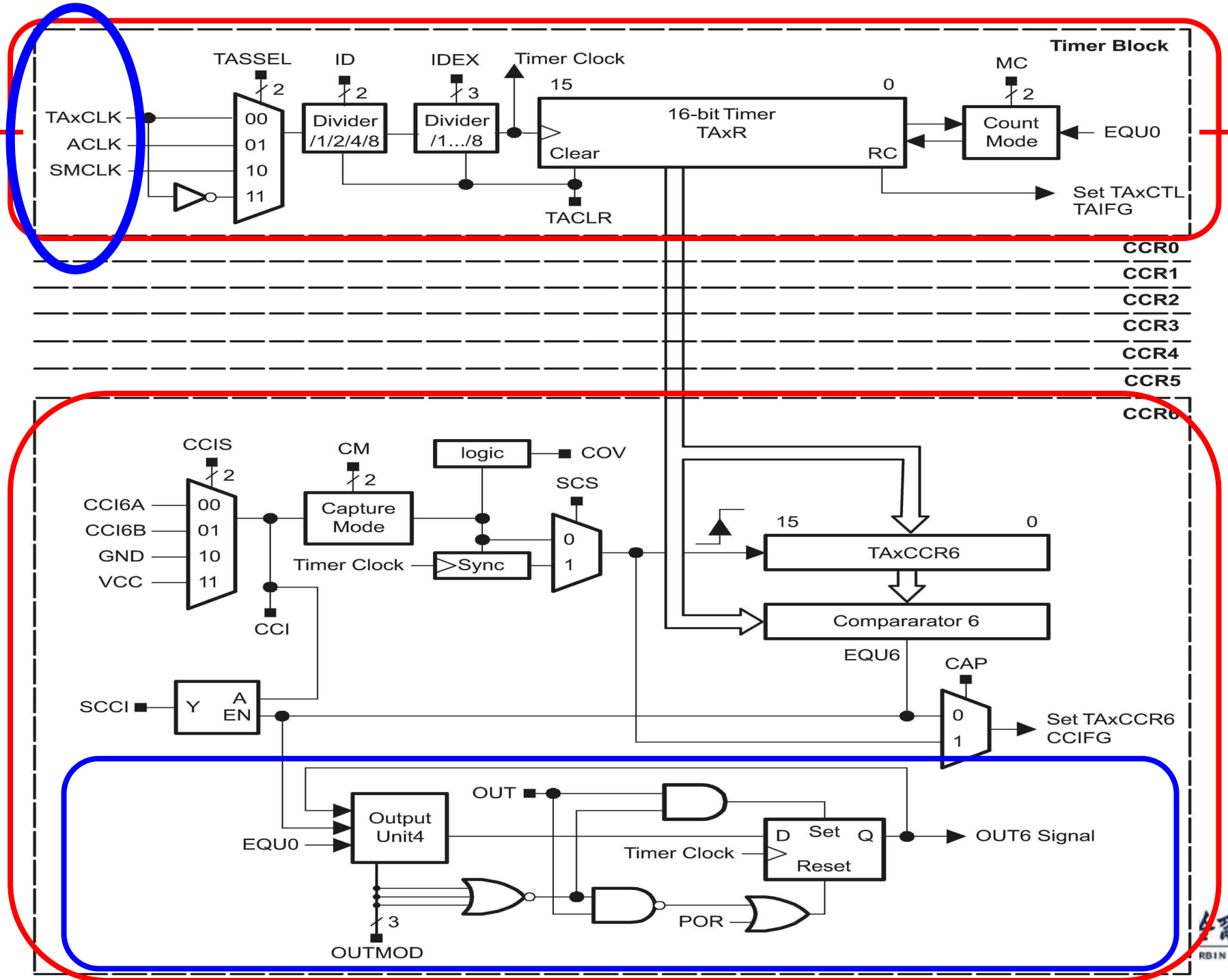
特性

- 定时器 Ax由一个16位定时器和多路捕获/比较通道组成。
- MSP430X5XX / 6XX系列单片机的Timer _A有以下特性：
  - 带有 4 种工作模式的异步16位定时/计数器;
  - 多种时钟源可选;
  - 可配置捕获/比较寄存器数多达 7 个;
  - 可配置的PWM（脉宽调制）输出;
  - 异步输入和同步锁存。不仅能捕获外部事件发生的时间
    还可锁定其发生时的高低电平;
  - 完善的中断服务功能。快速响应Timer_A中断的中断向量寄存器;
  - 8种输出方式选择。

定时器A主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

结构框图



计数器

捕获/比较器

输出单元

主要内容

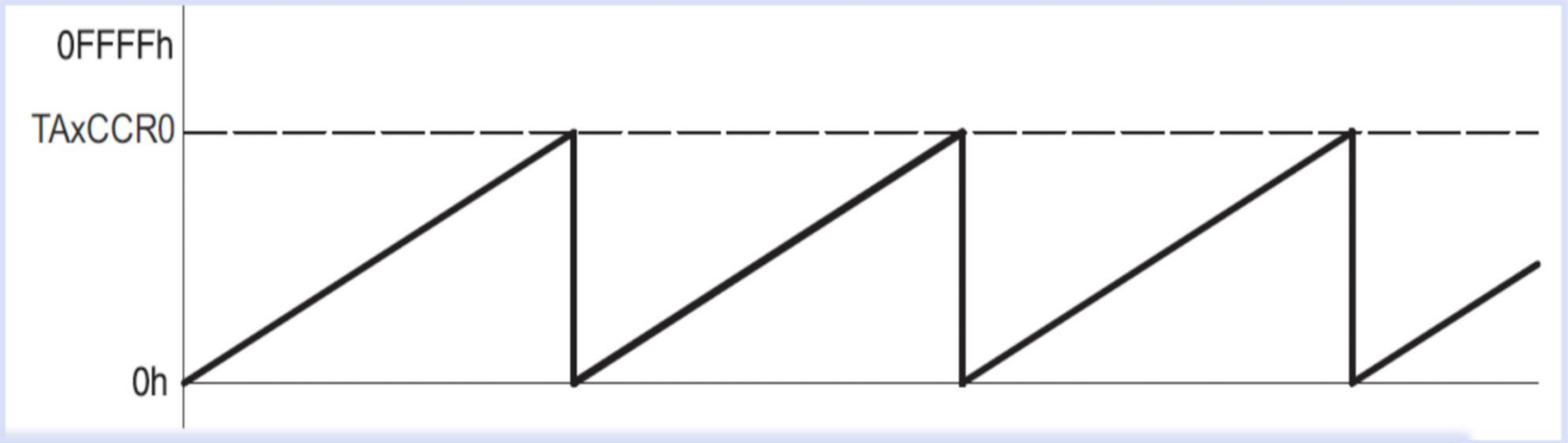
- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

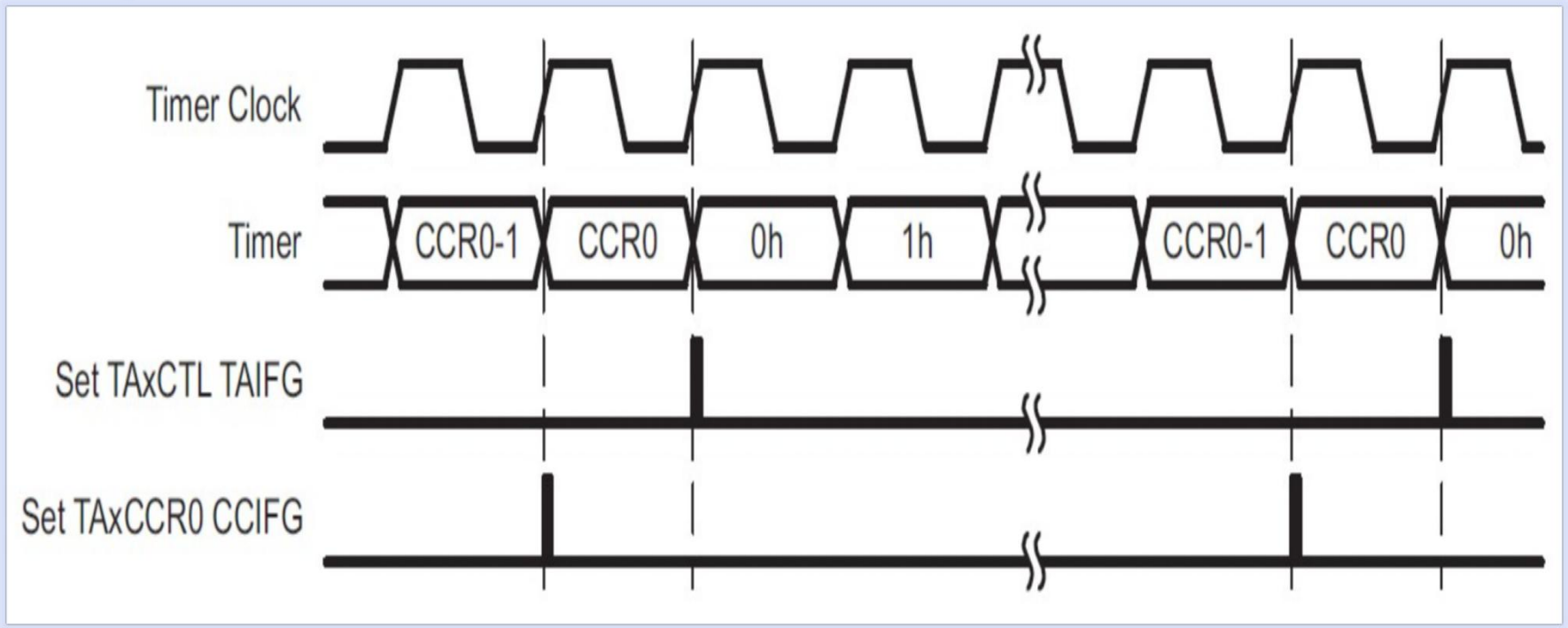
工作模式

MSP430X5XX / 6XX系列单片机的通用定时器共有4种计数模式，以定时器A为例，（TAxCTL 寄存器中的 MCx 位）

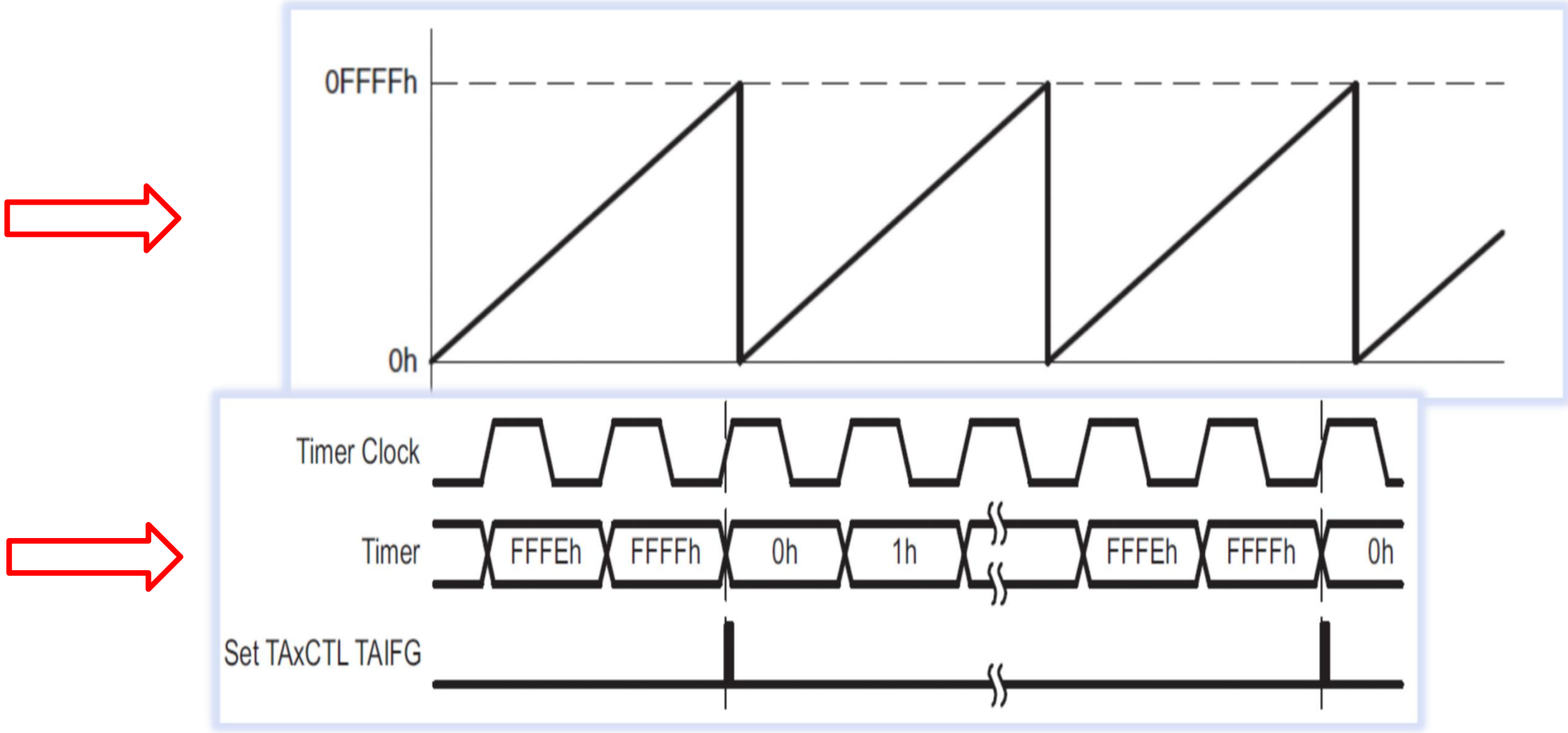
MCx	模式	说明
00	停止模式	定时器停止
01	增计数模式	定时器重复从 0 计数到 TAxCCR0
10	连续计数模式	定时器重复从 0 计数到 0FFFFh
11	增/减计数模式	定时器重复从 0 增计数到 TAxCCR0 再减计数到 0

增计数模式

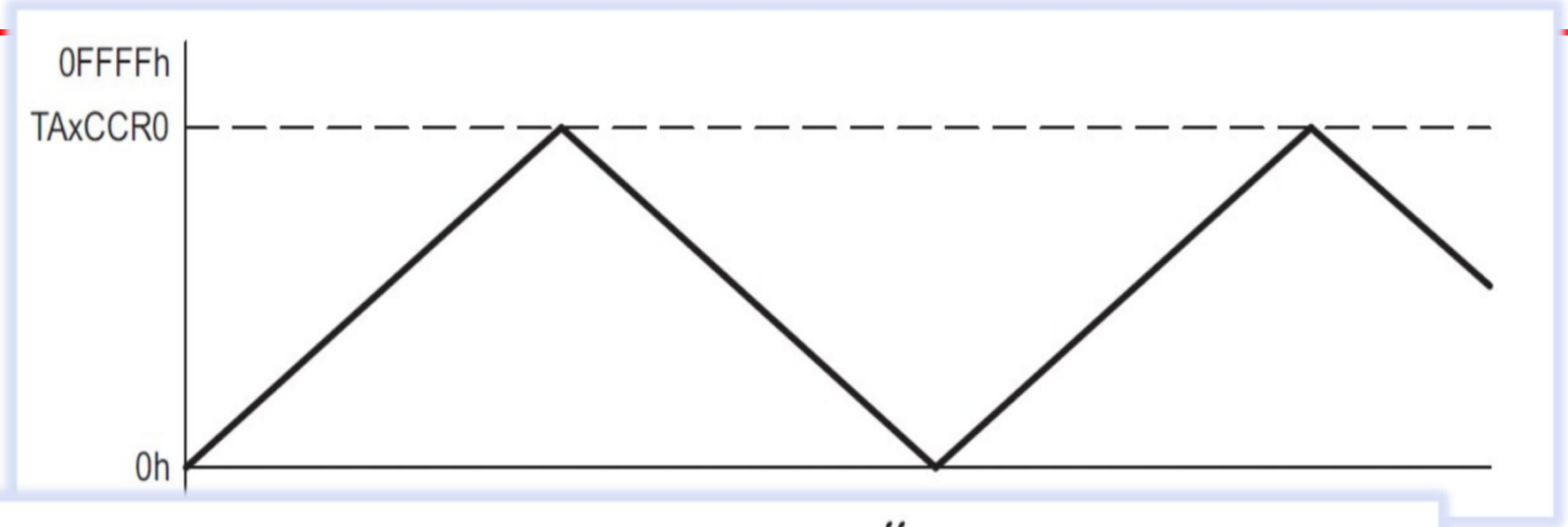

0FFFFh
TAxCCR0
0h

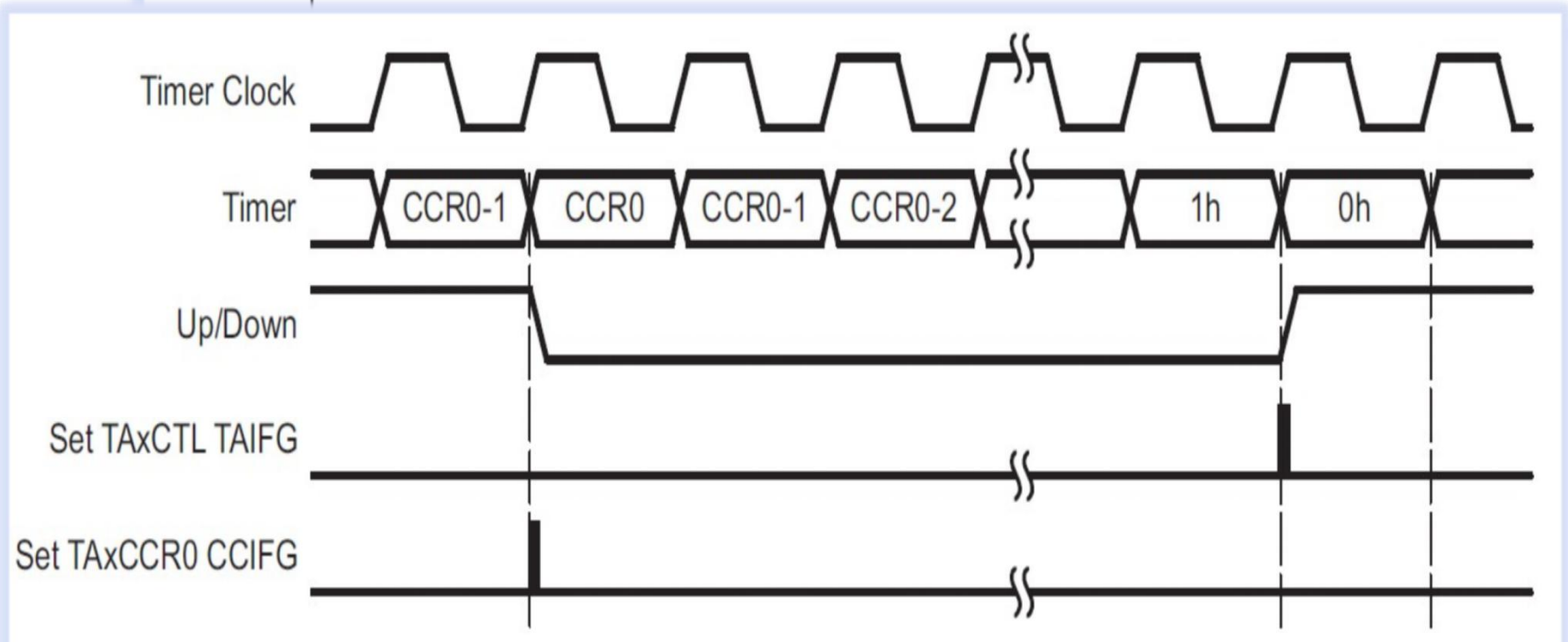

Timer Clock
Timer
CCR0-1
CCR0
0h
1h
CCR0-1
CCR0
0h
Set TAxCTL TAIFG
Set TAxCCR0 CCIFG

连续计数模式



增减计数模式

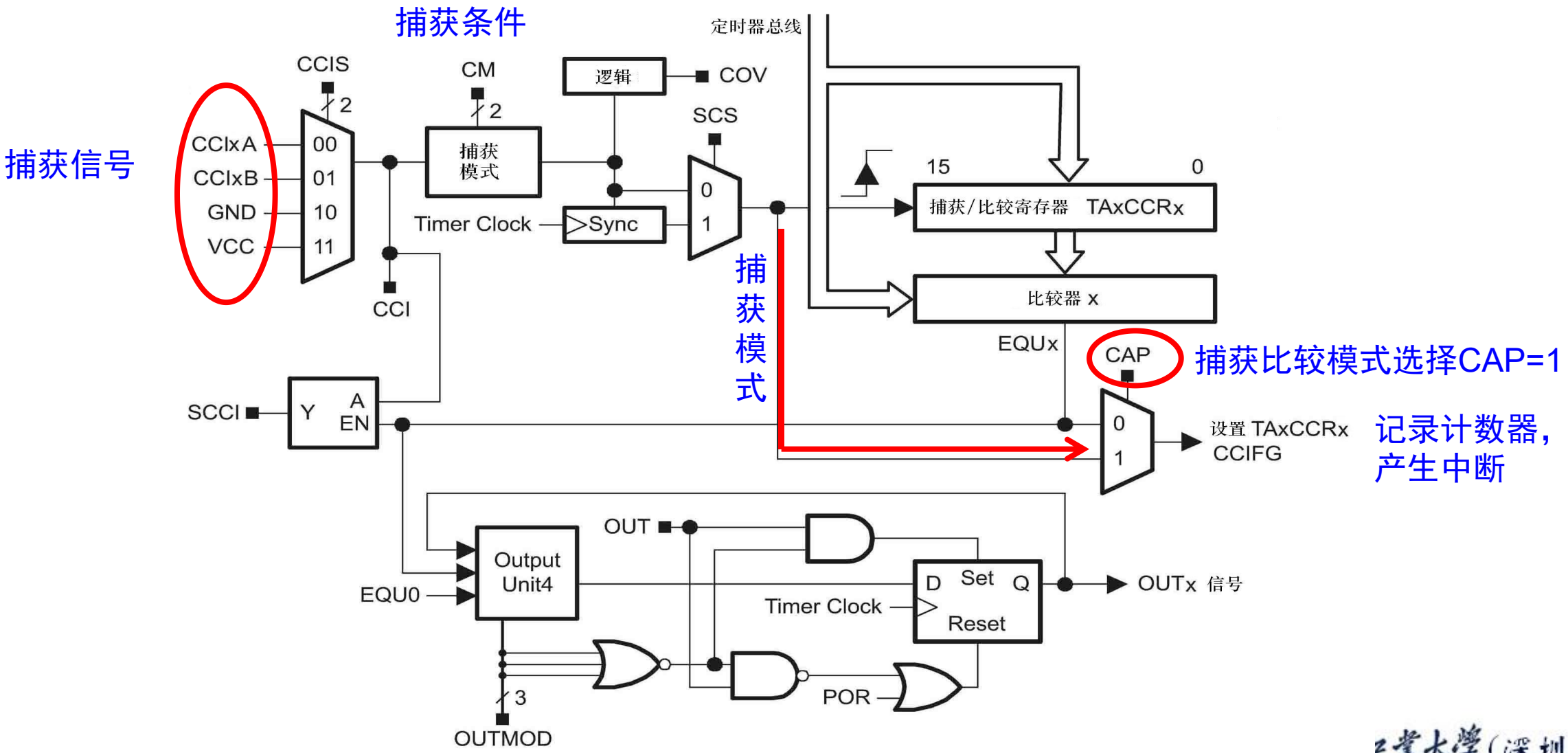

0FFFFh
TAxCCR0
0h


Timer Clock
Timer
CCR0-1
CCR0
CCR0-1
CCR0-2
1h
0h
Up/Down
Set TAxCTL TAIFG
Set TAxCCR0 CCIFG

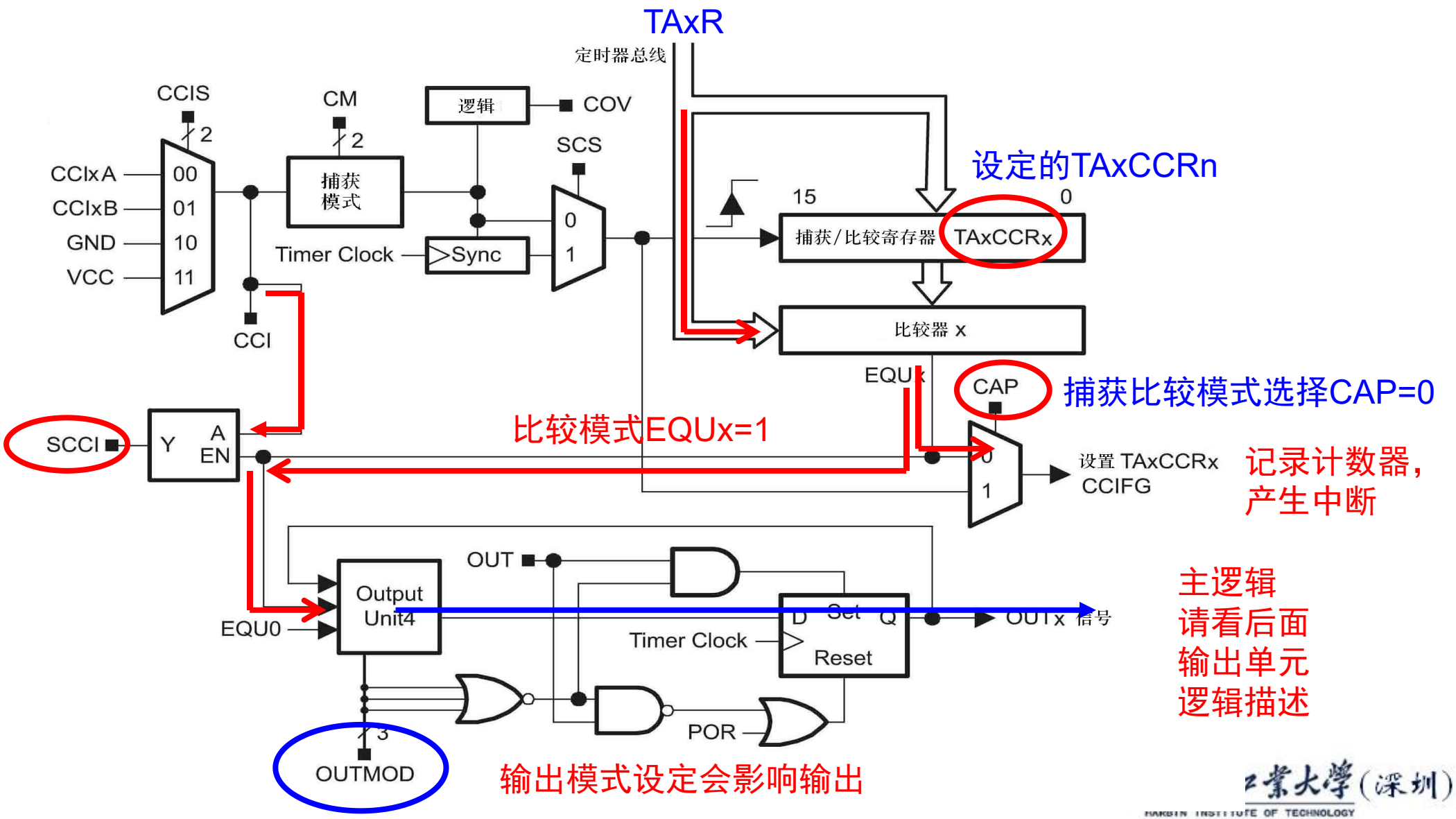
主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

捕获/比较模块


捕获信号
捕获条件
定时器总线
CCIS
2
CM
2
逻辑
COV
SCS
CCIxA
CCIxB
GND
VCC
00
01
10
11
捕获
模式
Timer Clock
Sync
0
1
CCI
15
0
捕获/比较寄存器 TAxCCRx
比较器 x
EQUx
CAP
捕获比较模式选择CAP=1
捕获模式
SCCI
Y
A
EN
0
1
设置 TAxCCRx
CCIFG
记录计数器，
产生中断
OUT
Output
Unit4
EQU0
D
Set
Q
Timer Clock
Reset
OUTx 信号
3
OUTMOD
POR

捕获/比较模块


TAxR
定时器总线
CCIS
2
CM
2
逻辑
COV
SCS
CCIxA
CCIxB
GND
VCC
00
01
10
11
捕获
模式
Timer Clock
Sync
0
1
CCI
设定的TAxCCRn
15
0
捕获/比较寄存器
TAxCCRx
比较器 x
EQUx
CAP
捕获比较模式选择CAP=0
比较模式EQUx=1
SCCI
Y
A
EN
设置 TAxCCRx
CCIFG
记录计数器，
产生中断
OUT
Output
Unit4
EQU0
D
Set
Q
Timer Clock
Reset
OUTx 信号
主逻辑
请看后面
输出单元
逻辑描述
POR
3
OUTMOD
输出模式设定会影响输出

主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

OUTMODx	模式	说明
000	输出模式0：输出	输出信号取决与寄存器 TACCTLx 中的 OUT位。当 OUT位更新时，输出信号立即更新。
001	输出模式1：置位	输出信号在TAxR等于TAxCCRn时置位，并保持置位到定时器复位或选择另一种输出模式为止。
010	输出模式2：翻转/复位	输出在TAxR的值等于TAxCCRn时翻转，当TAxR的值等于TAxCCR0时复位。
011	输出模式3：置位/复位	输出在TAxR的值等于TAxCCRn时置位，当TAxR的值等于TAxCCR0时复位。
100	输出模式4：翻转	输出电平在TAxR的值等于TAxCCRn时翻转，输出周期是定时器周期的2倍。
101	输出模式5：复位	输出在TAxR的值等于TAxCCRn时复位，并保持低电平直到选择另一种输出模式。
110	输出模式6：翻转/置位	输出电平在TAxR的值等于TAxCCRn时翻转，当TAxR值等于TAxCCR0时置位。
111	输出模式7：复位/置位	输出电平在TAxR的值等于TAxCCRn时复位，当TAxR的值等于TAxCCR0时置位。

◆ 定时器在增计数模式的输出实例

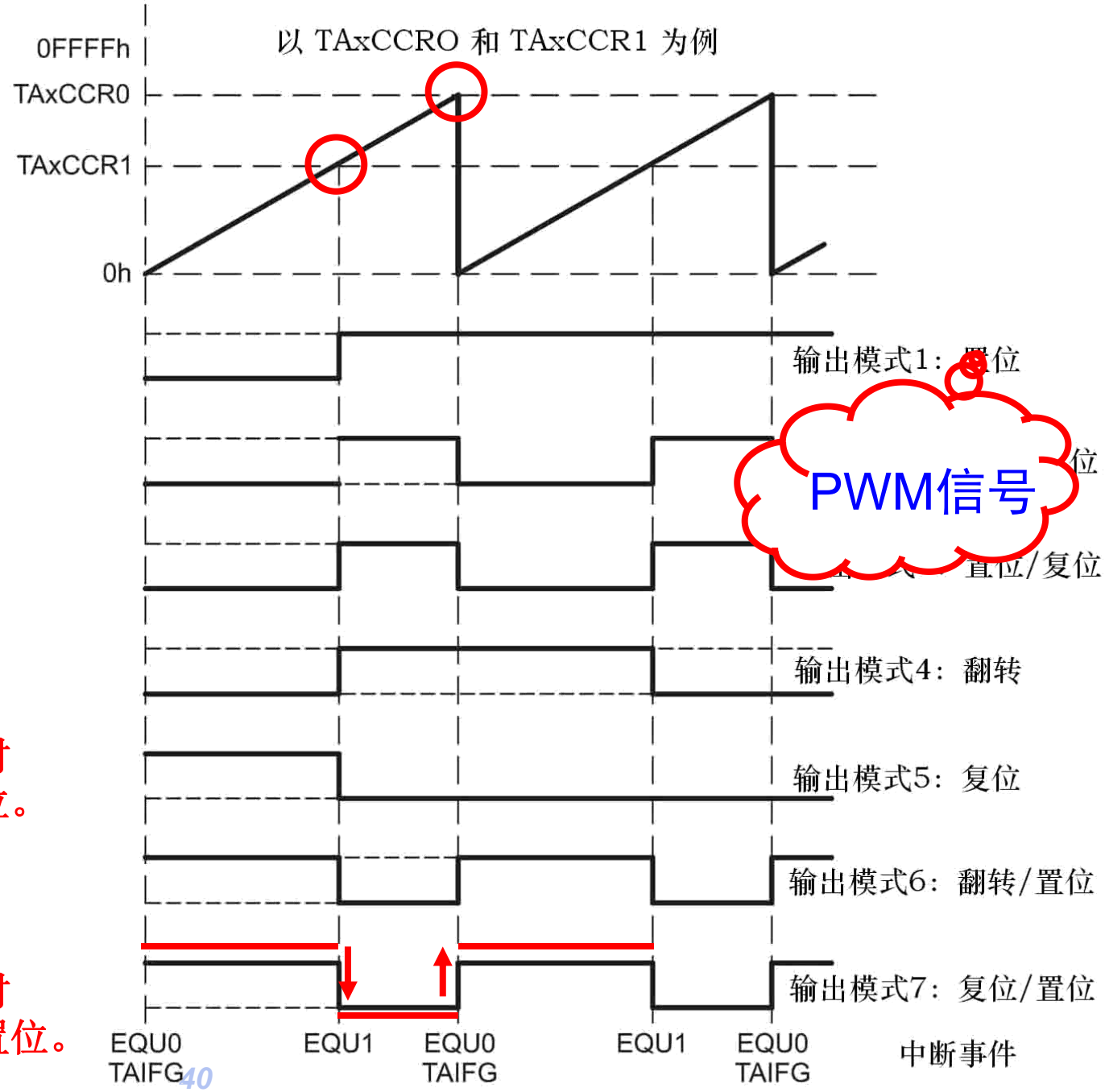
在增计数模式下，当计数器TAxR增加到TAxCCRx或从TAxCCR0计数到0时，OUTn信号按选择的输出模式发生变化。实例如右图所示。

输出模式6：
翻转/置位

输出电平在TAxR的值等于TAxCCRn时翻转，当TAxR值等于TAxCCR0时置位。

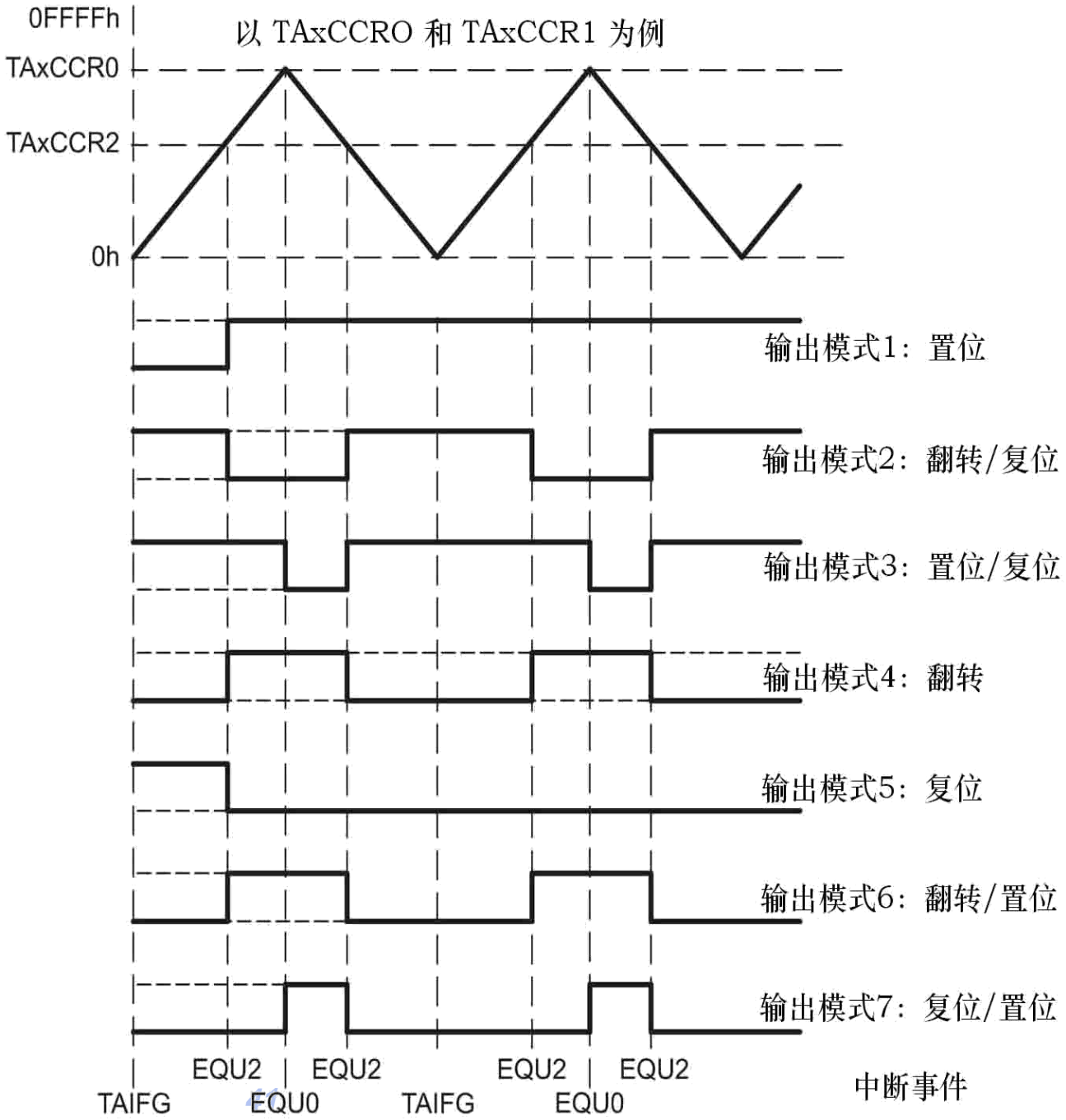
输出模式7：
复位/置位

输出电平在TAxR的值等于TAxCCRn时复位，当TAxR的值等于TAxCCR0时置位。



◆ 定时器在增/减计数模式的输出实例

在增/减计数模式下的输出实例，如右图所示。这时的各种输出波形与定时器增计数模式或连续计数模式不同。当定时器在任意计数方向上等于TAxCCRx时，OUTn信号都按选择的输出模式发生改变。

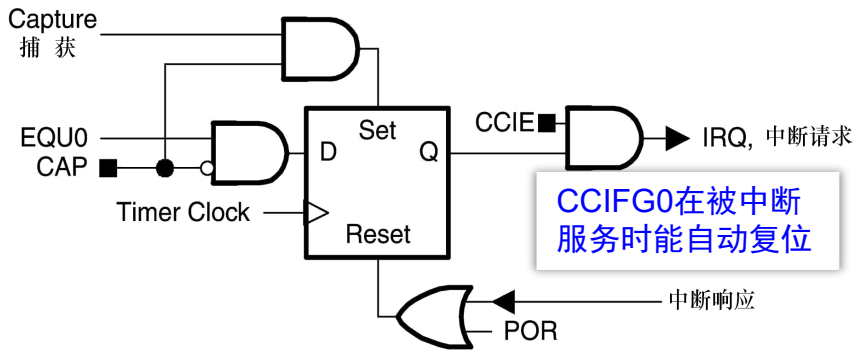


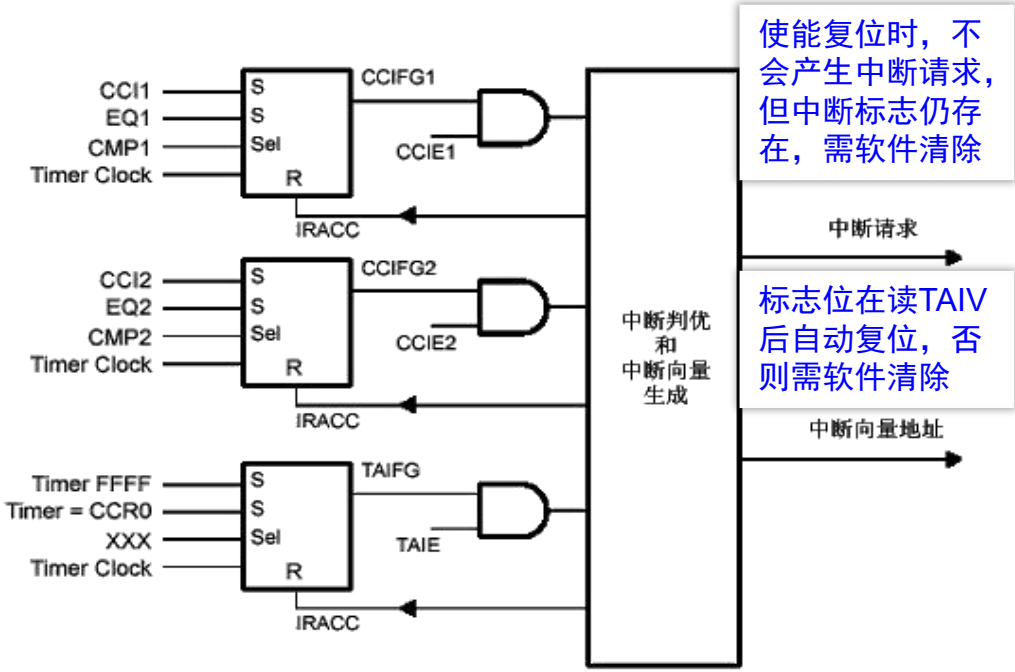
主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

中断

- Timer_A中断可由计数器溢出引起，也可以来自捕获/比较寄存器。每个捕获/比较模块可独立编程，由捕获/比较外部信号以产生中断。
- Timer_A模块使用两个中断向量：
  - 一个单独分配给捕获/比较寄存器TAxCCR0；
  - 另一个作为共用中断向量用于定时器和其他的捕获/比较寄存器（TAIV）。
- TAxCCR1～TAxCCRx和定时器按照优先次序结合共用一个中断向量，属于多源中断。中断向量寄存器（TAIV）用于确定哪个标志请求中断。（用法可参见用户手册）
- TAxCCR1～TAxCCRx中断，如右图所示：





主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

定时器A寄存器

控制寄存器

捕获比较控制寄存器

计数器

捕获比较寄存器

中断向量寄存器

Table 17-3. Timer_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TAxCTL	Timer_Ax Control	Read/write	Word	0000h	Section 17.3.1
02h	TAxCCTL0	Timer_Ax Capture/Compare Control 0	Read/write	Word	0000h	Section 17.3.3
04h	TAxCCTL1	Timer_Ax Capture/Compare Control 1	Read/write	Word	0000h	Section 17.3.3
06h	TAxCCTL2	Timer_Ax Capture/Compare Control 2	Read/write	Word	0000h	Section 17.3.3
08h	TAxCCTL3	Timer_Ax Capture/Compare Control 3	Read/write	Word	0000h	Section 17.3.3
0Ah	TAxCCTL4	Timer_Ax Capture/Compare Control 4	Read/write	Word	0000h	Section 17.3.3
0Ch	TAxCCTL5	Timer_Ax Capture/Compare Control 5	Read/write	Word	0000h	Section 17.3.3
0Eh	TAxCCTL6	Timer_Ax Capture/Compare Control 6	Read/write	Word	0000h	Section 17.3.3
10h	TAxR	Timer_Ax Counter	Read/write	Word	0000h	Section 17.3.2
12h	TAxCCR0	Timer_Ax Capture/Compare 0	Read/write	Word	0000h	Section 17.3.4
14h	TAxCCR1	Timer_Ax Capture/Compare 1	Read/write	Word	0000h	Section 17.3.4
16h	TAxCCR2	Timer_Ax Capture/Compare 2	Read/write	Word	0000h	Section 17.3.4
18h	TAxCCR3	Timer_Ax Capture/Compare 3	Read/write	Word	0000h	Section 17.3.4
1Ah	TAxCCR4	Timer_Ax Capture/Compare 4	Read/write	Word	0000h	Section 17.3.4
1Ch	TAxCCR5	Timer_Ax Capture/Compare 5	Read/write	Word	0000h	Section 17.3.4
1Eh	TAxCCR6	Timer_Ax Capture/Compare 6	Read/write	Word	0000h	Section 17.3.4
2Eh	TAxIV	Timer_Ax Interrupt Vector	Read only	Word	0000h	Section 17.3.5
20h	TAxEX0	Timer_Ax Expansion 0	Read/write	Word	0000h	Section 17.3.6

定时器A控制寄存器

Timer_Ax Control Register

Figure 17-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TA count direction. The TACLR bit is autom
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

时钟源选择

分频选择

工作模式选择

清零计数器TAR

TimerA溢出中断使能

定时器溢出中断标志

定时器A计数器

Timer_Ax Counter Register

Figure 17-17. TAxR Register

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

Table 17-5. TAxR Register Description

Bit	Field	Type	Reset	Description
15-0	TAxR	RW	0h	Timer_A register. The TAxR register is the count of Timer_A.

定时器A捕获/比较控制寄存器

Timer_Ax Capture/Compare Control n Register

Figure 17-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling
13-12	CCIS	RW	0h	Capture/compare input select. These bit the device-specific data sheet for specifi 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is u signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. Th with the EQUx signal and can be read vi
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

输出模式选择

捕获/比较中断使能

比

捕获/比较中断标志

定时器A捕获/比较寄存器

17.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

Figure 17-19. TAxCCRn Register

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

Table 17-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed.

定时器A中断向量寄存器

17.3.5 TAxIV Register

Timer_Ax Interrupt Vector Register

Figure 17-20. TAxIV Register

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

Table 17-8. TAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	TAIV	R	0h	Timer_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest 04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest

定时器A库函数

- 定时器配置和初使化
- 定时器输出
- 定时器中断处理

MSP430F5xx_6xx_DriverLib_Users_Guide.pdf → 章节37

主要内容

- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

演示实验2.2

- **课堂实验演示：**

利用定时器A的定时中断功能，控制一个LED灯开启和关断。在满足接口条件下，开启LED，同时定时器开启计时（如果查询一直满足接口条件则定时器在每次查询时均重新开始计时）；不满足接口条件下，定时器计时检查，到达5s则引起中断，中断服务程序完成关断LED灯的操作。

- **提示：**

（1）接口条件可用一个接口函数完成，可以是按键的操作，
或者任何可判断的条件

（2）Timer_A：增计数模式，时钟源选择SMCLK，初始化操作

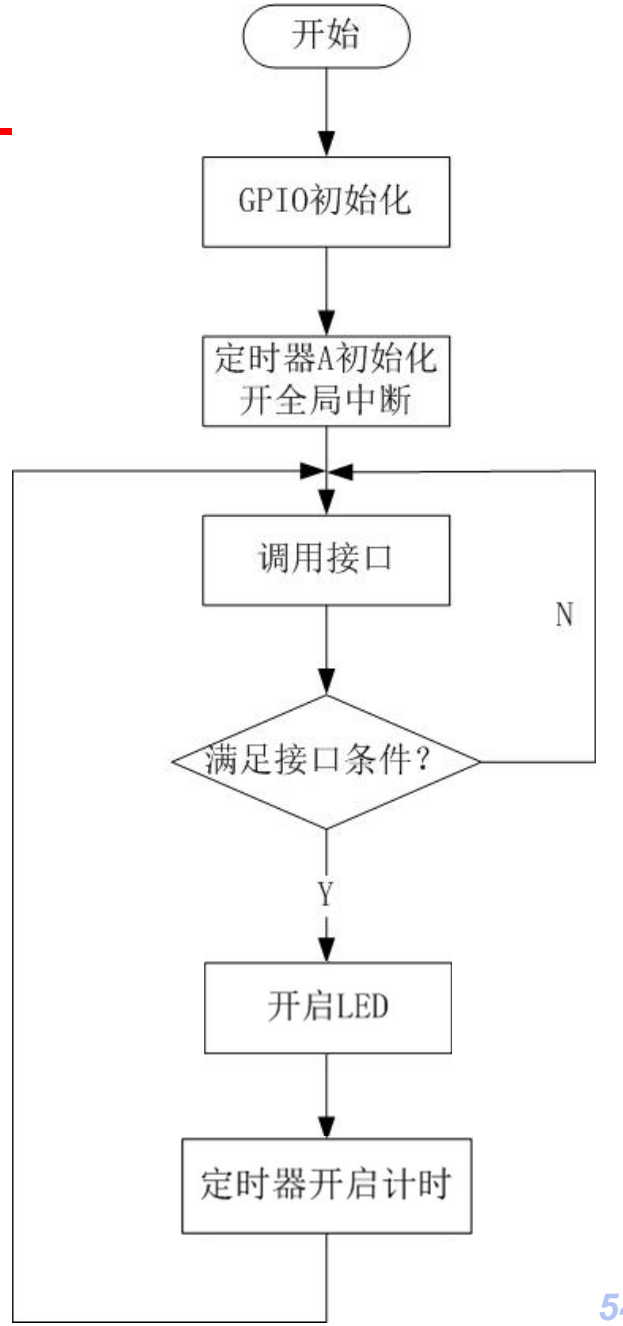
（3）使能捕获比较中断，设定比较值（计时时长）

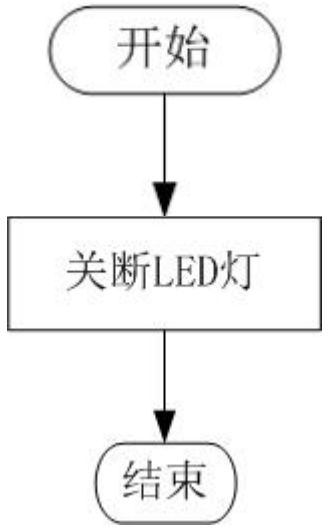
实验现象

流程示例

演示实验2.2

主循环参考流程图





中断服务程序参考流程图

演示实验2.2

练一练

```c
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdo
    IO_Init();
    TA0CTL |= MC_1 + TASSEL_2 + TACLR;        //时
    TA0CCTL0 = CCIE;                          //比
    TA0CCR0  = 50000;                         //比
    __enable_interrupt();

    while(1)
    {
        if(apInterface_Key()==1)
        {
            P8OUT |= BIT1;
            P3OUT |= BIT7;
            TA0CTL |= MC_1 + TASSEL_2 + TACLR;
            icnt=0;
        }
    }

}
```

```c
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    icnt++;
    if(icnt==100)
    {//5s关灯
        P8OUT &= ~BIT1;
        P3OUT &= ~BIT7;
        icnt=0;
    }
}
```

伪代码-详见Q群
代码编号

课堂实验2.3

- **实验要求：**

利用定时器A的定时中断功能，控制一个LED灯开启和关断。在满足接口条件下，开启LED，同时定时器开启计时（如果查询一直满足接口条件则定时器在每次查询时均重新开始计时）；不满足接口条件下，定时器计时检查，到达10s则引起中断，中断服务程序完成关断LED灯的操作。

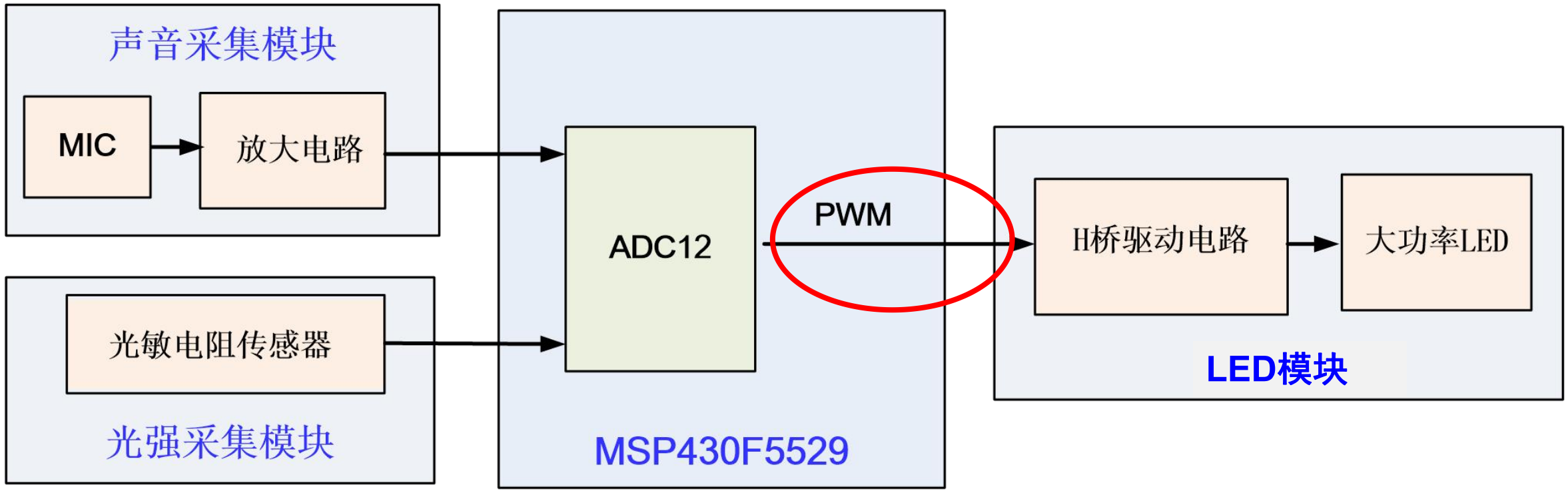
- **提示：**

根据演示实验2.2，修改代码。

主要内容

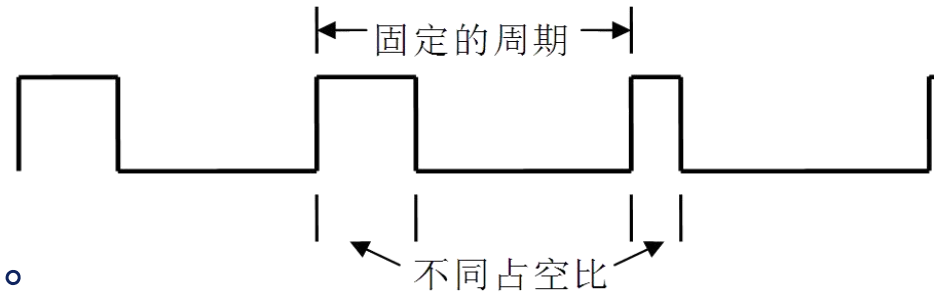
- 定时器A的特性
- 定时器A的结构
- 定时器A的工作原理
  - 定时器工作模式
  - 捕获/比较模块
  - 输出单元
  - Timer_A中断
- 定时器A的寄存器
- 定时器A中断实验
- 定时器A的典型应用—PWM

展馆灯光控制系统功能需求分析-总体设计



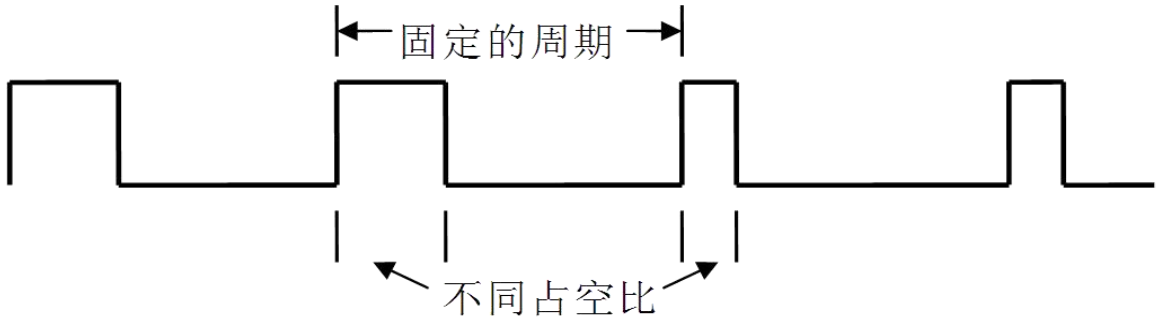
PWM（Pulse-Width Modulation）：脉宽调制的缩写

PWM信号是一种具有固定周期不定占空比的数字信号。

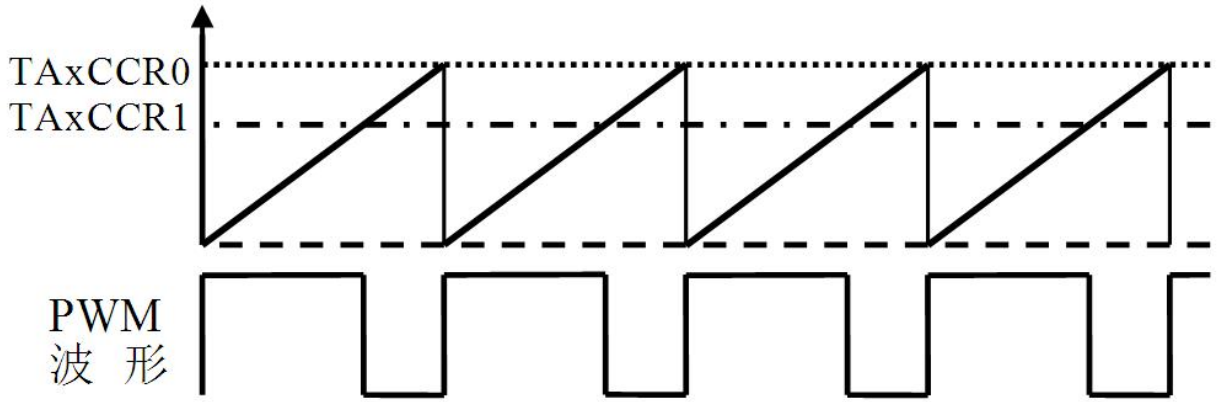


PWM信号用定时器A实现

◆ PWM（Pulse Width Modulation）信号是一种具有固定周期不定占空比的数字信号，如下图所示：



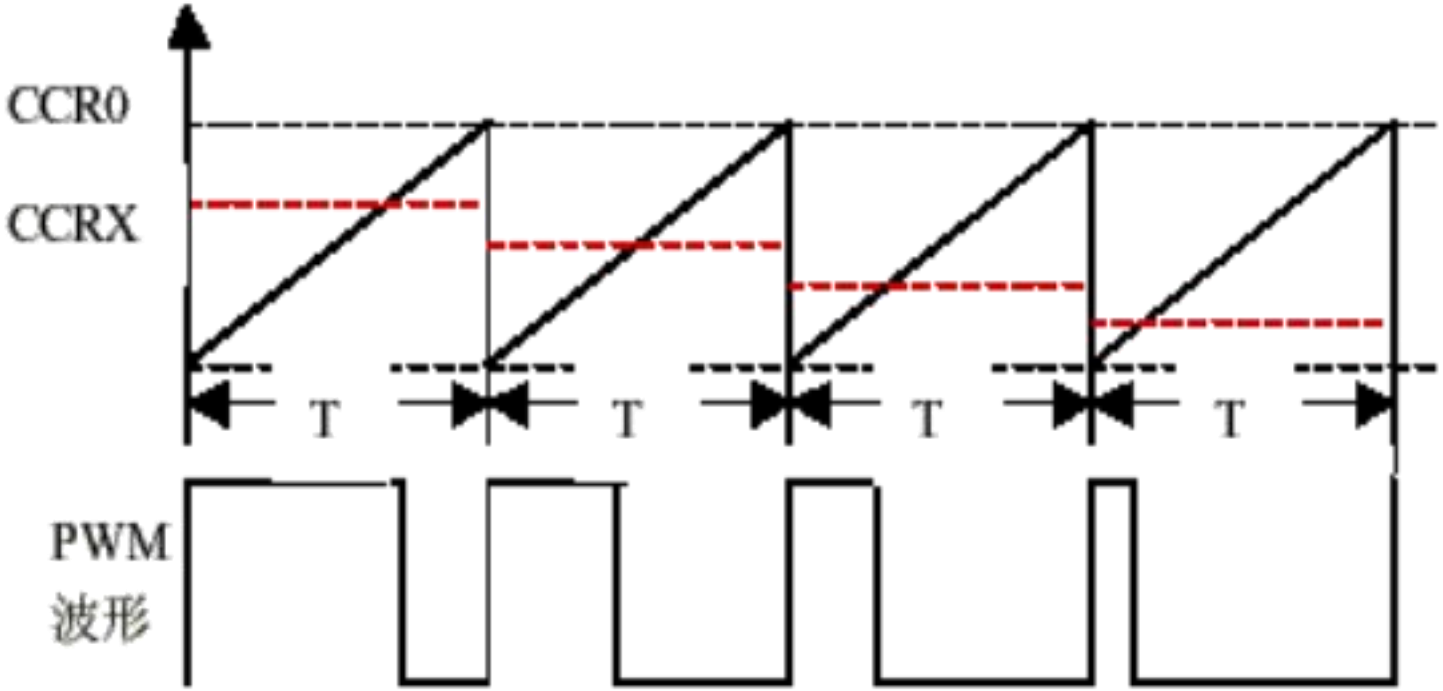
◆ 如果Timer_A定时器的计数器工作在增计数方式，输出采用输出模式7（复位/置位模式），则可利用寄存器TAxCCR0控制PWM波形的周期，用某个寄存器TAxCCRx控制占空比。这样Timer_A就可以产生出任意占空比的PWM波形。如下图所示：



PWM信号用定时器A实现

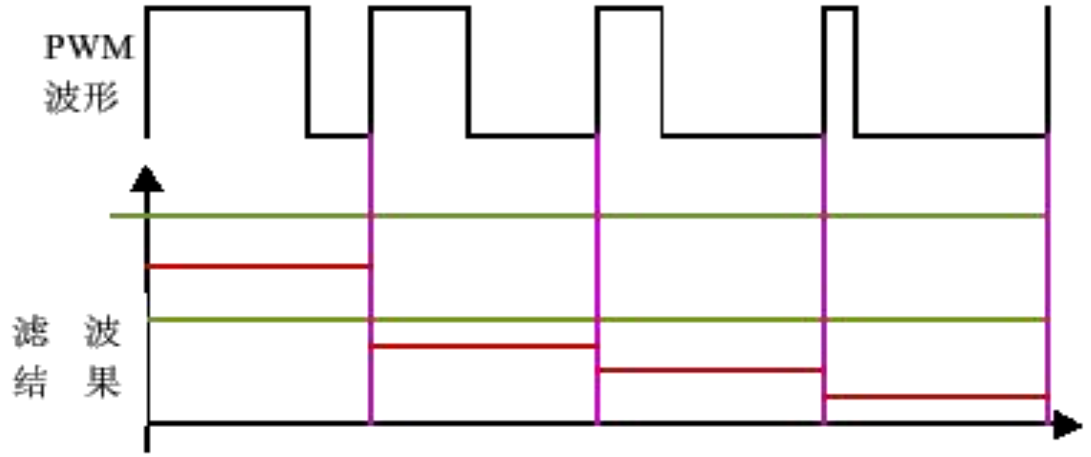
- 可以随时间变化任意改变PWM信号的占空比，具体做法：
  - 保持CCR0值（周期不变）；
  - 改变CCRx值（改变占空比）。

如下图所示：



PWM信号用定时器A实现

◆ 如果PWM信号占空比随时间变化，那么经过滤波之后的输出信号就是幅度变化的模拟信号，因此通过控制PWM信号的占空比，就可以产生不同的模拟信号，实现D/A转换。如下图所示：



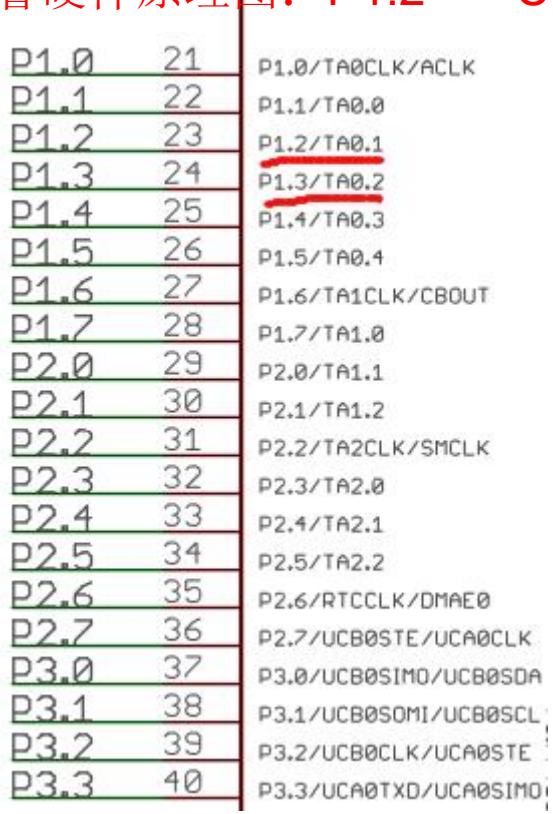
◆ PWM不需要修改占空比和时间时，CPU在做完Timer_A初始化工作之后，Timer_A就能自动输出PWM，而不需利用中断维持PWM输出，此时CPU就可以进入低功耗状态。

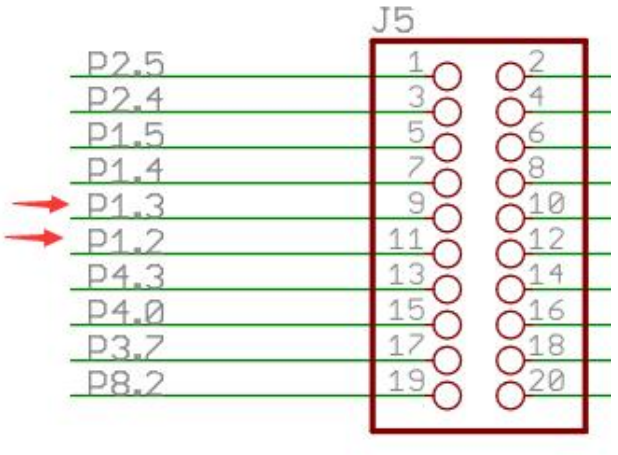
PWM信号用定时器A实现-演示实验2.3

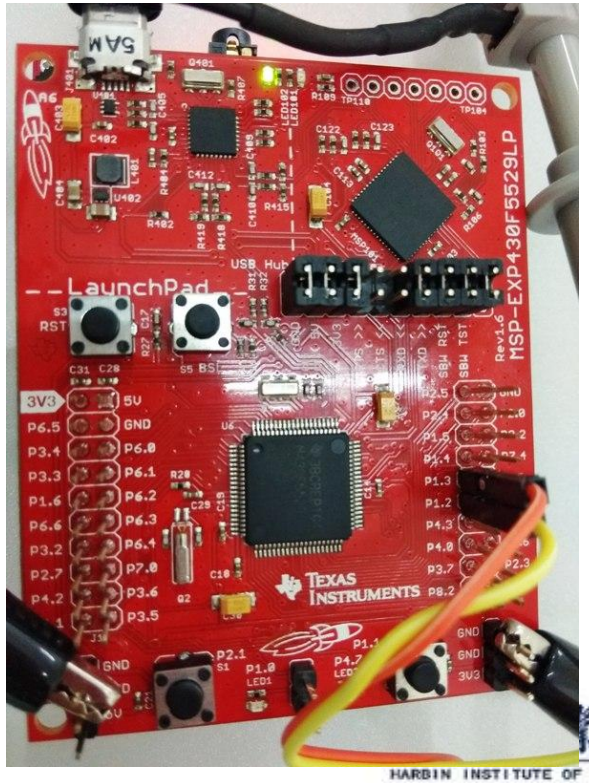
◆ 实验要求（指导书2.4.4）：

设ACLK = TACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCOCLK = 32×ACLK = 1.048576MHz，利用Timer_A输出周期为1s、占空比分别为90%和10%的PWM矩形波。请用示波器或指示灯验证现象。

查看硬件原理图：P1.2—> CCR1 - 90% PWM，P1.3—> CCR2 - 10% PWM。周期1s。

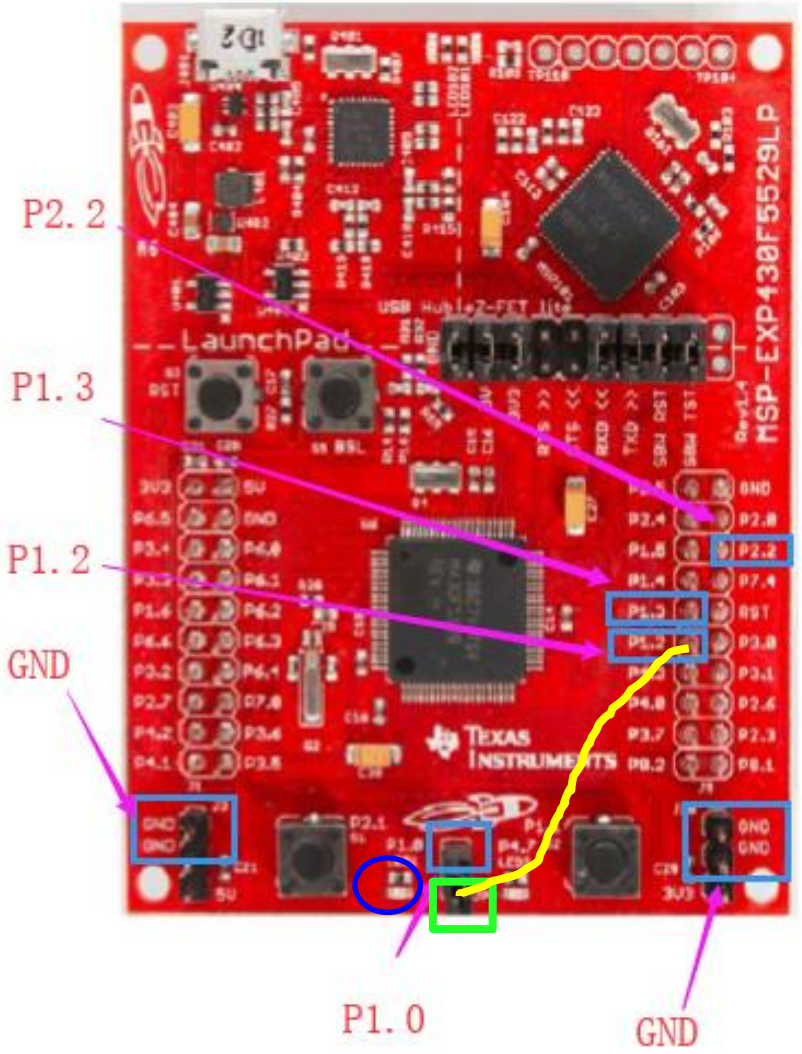




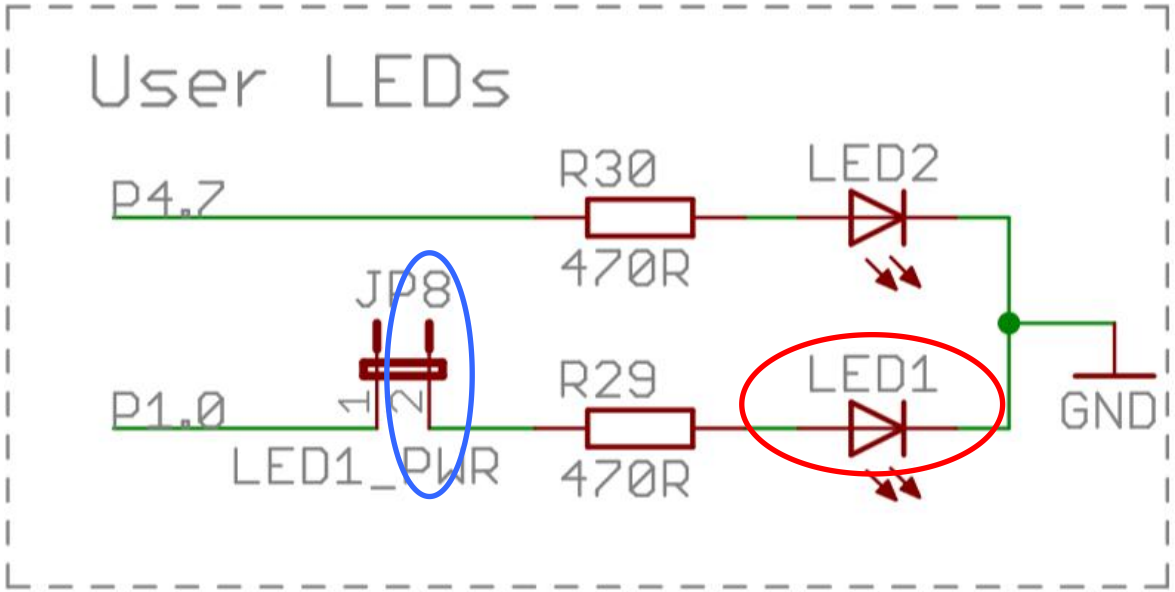


PWM信号用定时器A实现-演示实验2.3

接线部分引脚



杜邦线



PWM信号用定时器A实现-演示实验2.3

```
int main(void)
{
// stop watchdog timer
WDTCTL = WDTPW | WDTHOLD;

// ACLK, 清除 TAR，增计数
TA0CTL = TASSEL_1 + TACLR + MC0;
          (0x100)   (0x04)   (0x10)
TA0CCR0 = 32768;  // PWM周期

TA0CCTL1 = OUTMOD_7 ;// 输出模式7
TA0CCR1 = 29491;  //占空比90%

TA0CCTL2 = OUTMOD_7; // 输出模式7;
TA0CCR2 = 3277;  //占空比10%

P1DIR |= BIT2;    // P1.2 方向为输出
P1SEL |= BIT2;    // P1.2端口为外设，定时器TA0
P1DIR |= BIT3;    // P1.3 方向为输出
P1SEL |= BIT3;    // P1.3端口为外设，定时器TA0
}
```

Timer_Ax Control Register

Figure 17-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TA count direction. The TACLR bit is autom
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

时钟源选择

分频选择

工作模式选择

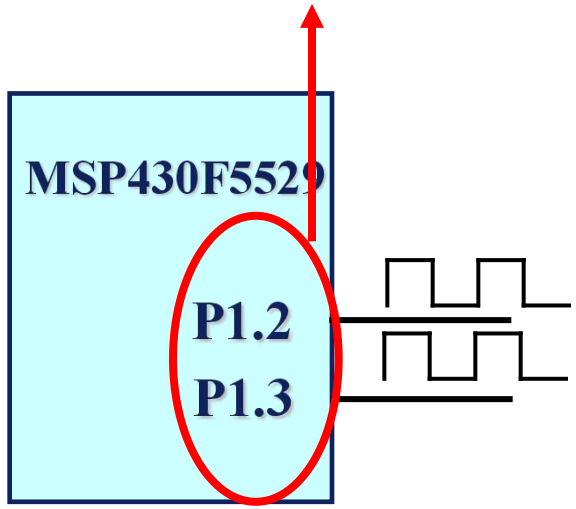
清零计数器TAR

TimerA溢出中断使能

定时器溢出中断标志

PWM信号用定时器A实现-演示实验2.3

此实验不可同时操作口袋板按键S1\S2！烧毁芯片引脚



哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

课堂实验2.4

◆ 实验要求：

设ACLK = TACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCOCLK = 32×ACLK = 1.048576MHz，利用Timer_A输出周期为512 /1MHz≈0.5ms、占空比分别为80%和10%的PWM矩形波。请用指示灯LED1验证现象。

查看硬件原理图：P1.4—> CCR1 - 80% PWM，P1.5—> CCR2 - 20% PWM。周期1s。

◆ 思考：与示例程序的区别：CCR0（周期由1s改为0.5ms，TimerA时钟选为SMCLK）

CCR3、 CCR4如何选择？

与示例现象的区别？