

**分析时钟树、配置GPIO**

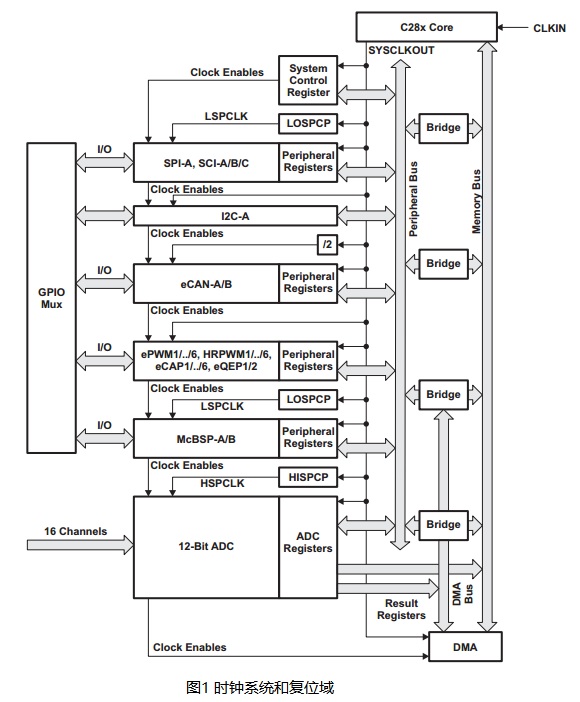
|  |  |
| --- | --- |
| **姓 名** | **冯浩然** |
| **学 号** | **202100800378** |
| **专 业** | **电子科学与技术** |
| **班 级** | **2021级1班** |
| **学 院** | **机电与信息工程学院** |
| **指导教师** | **王小利** |
| **课程名称** | **DSP原理与应用** |
| **课 程 号** | **28033160** |
| **学年学期** | **2023–2024学年第2学期** |
| **作业日期** | **2024年5月30日** |

# 一 系统时钟树

TMS320F28335通过外部时钟信号、OSC和PLL产生倍频时钟信号CLKIN后，CLKIN经过CPU后产生时钟SYSCLKOUT(CLKIN和SYSCLKOUT频率是一样的)，SYSCLKOUT给各个片内外设提供时钟信号。为了实现低功耗和提供高低频率时钟信号，需要把SYSCLKOUT进一步分频，下面主要讲解了对SYSCLKOUT分频产生低频时钟信号和高频时钟信号，完成外设时钟初始化的过程。

## 1 时钟系统

时钟系统如下图所示

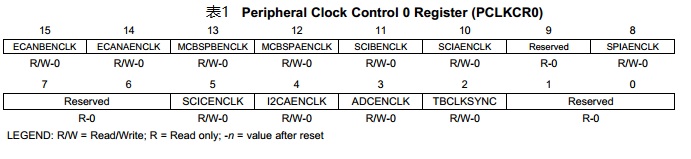


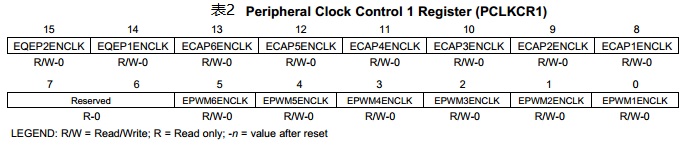
从图1可以看出，时钟CLKIN经过CPU后产生时钟SYSCLKOUT(CLKIN和SYSCLKOUT频率是一样的)，SYSCLKOUT给各个片内外设提供时钟；除了SPI、SCI、McBSP模块使用低频时钟，ADC使用高频时钟外，其他外设模块都是采用SYSCLKOUT时钟。为了实现低功耗，必须对每个片内外设时钟进行开关控制；为了实现高低频率时钟，必须对SYSCLKOUT进行不同的分频处理，因此与外设时钟配置相关的寄存器主要有两类：外设时钟控制寄存器PCLKCR和高低频外设时钟分频寄存器SPCP。

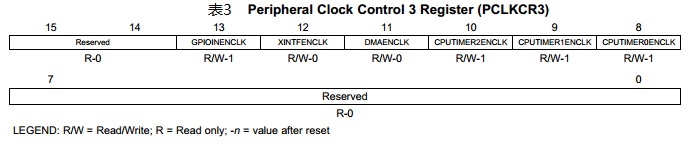
|  |  |  |  |
| --- | --- | --- | --- |
| 寄存器类型 | 位数 | 名字 | 作用 |
| 第一类寄存器（外设时钟控制寄存器PCLKCR） | 16位 | PCLKCR0 | 控制使能  禁用外设时钟 |
| 16位 | PCLKCR1 |
| 16位 | PCLKCR3 |
| 第二类寄存器（高低频外设时钟分频寄存器） | 16位 | HISPCP（High-Speed Peripheral Clock Prescaler Register） | 对外设时钟进行分频  调整高频和低频外设的时钟频率 |
| 16位 | LOSPCP（Low-Speed Peripheral Clock Prescaler Register） |

## 2 与外设时钟配置相关的寄存器

控制片内外设时钟开关的外设时钟控制寄存器PCLKCR0、PCLKCR1、PCLKCR3如表1、表2、表3所示。







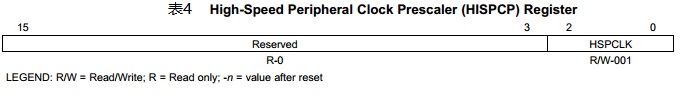
从表1、表2、表3可以看出，默认情况下，除了3个CPU定时器和GPIO口输入采样时钟使能外，其他所有外设的时钟是禁用的。

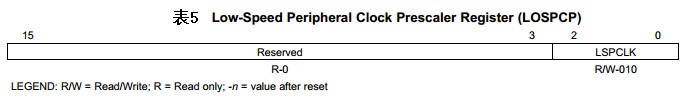
***注：***

***- PCLKCR0、PCLKCR1、PCLKCR3均受ELLOW保护。***

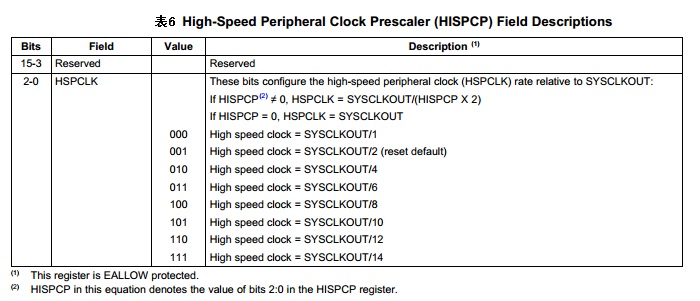
***- 不使用某外设模块时，可以禁用该时钟模块的时钟以省电。***

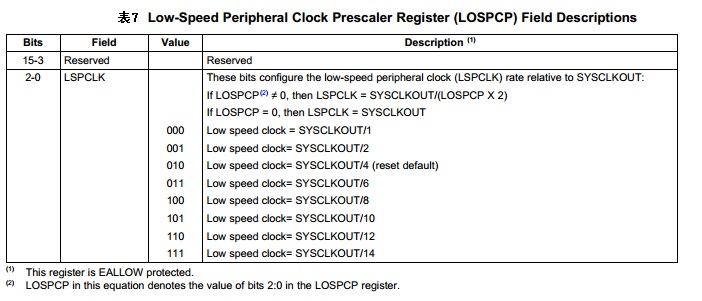
产生高低频外设时钟的分频寄存器HISPCP和LOSPCP分别如表4和表5所示。





两个寄存器具体位域描述分别如表6和表7所示。



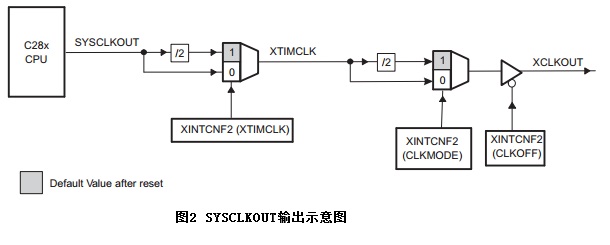


从表4到表7可以看出，16位的高频外设时钟分频寄存器HISPCP和低频外设时钟分频寄存器LOSPCP都只用了低3位；两个寄存器都受ELLOW保护；

分频计算方法：当分频系数为0，表示时钟等于SYSCLKOUT/1，当分频系数不为0时， 。

## 3 时钟输出

SYSCLKOUT可以按1、2、4分频从TMS320F28335的XCLKOUT引脚输出，SYSCLKOUT输出示意图如2所示。



从图2可以看出，上电或复位默认情况下，SYSCLK2分频产生XTIMCLK，XTIMCLK再通过2分频产生XCLKOUT时钟，该时钟信号通过引脚XCLKOUT输出，即默认情况下调试时可以观察该引脚的信号以判断设备是否在正确的时钟下工作。

**注：**

**XCLKOUT引脚上电或者复位默认情况下是激活状态的。**

**XCLKOUT引脚没有上拉或者下拉电阻。**

**如果XCLKOUT引脚不使用时，可以通过XINTCNF2[CLKOFF]=1关闭。**

**默认情况下，，它是外扩模块（外扩FLASH、SRAM等）的时钟**

## 4.外设时钟初始化代码

这里以ADC时钟配置为例

1. */\**
2. \*函数名称:InitPeripheralClocks
3. \*输入参数：无
4. \*输出参数：无
5. \*函数功能：初始化外设时钟模块，主要包括产生高低速时钟、开关所需片内外设模块时\*钟等
6. \*/
7. void InitPeripheralClocks(void)
8. {
9. EALLOW;
10. *// 高低频外设时钟分频寄存器HISPCP/LOSPCP设置，正常情况下采用默认值，即高频时钟为SYSCLKOUT/2，低速时钟为SYSCLKOUT/4*
11. SysCtrlRegs.HISPCP.all = 0x0001;
12. SysCtrlRegs.LOSPCP.all = 0x0002;
13. *// 时钟输出引脚XCLKOUT设置，默认情况XCLKOUT = SYSCLKOUT/4*
14. *// XTIMCLK = SYSCLKOUT/2*
15. XintfRegs.XINTCNF2.bit.XTIMCLK = 1;*//extern interface clock<->XinfCLK*
16. *// XCLKOUT = XTIMCLK/2*
17. XintfRegs.XINTCNF2.bit.CLKMODE = 1;
18. *// Enable XCLKOUT*
19. XintfRegs.XINTCNF2.bit.CLKOFF = 0;
20. *// 给所选用外设使能外设时钟*
21. *// 如果不使用某外设模块，禁用它的时钟以省电*
22. *// 下面代码要根据自己使用的外设模块进行相应的修改*
23. SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;    *// ADC*
24. *// \*重要提醒\**
25. *// ADC\_cal（）函数, 可以从TI保留的OTP中复制ADC校验值，并将校验值赋值给ADCREFSEL和ADCOFFTRIM寄存器，该过程在BOOT ROM中自动地完成*
26. *//如果在调试过程中，BOOT ROM代码旁路未使用，那么必须显示调用下面 ADC\_cal()函数（推荐显示调用）*
27. *//在调用 ADC\_cal()前，必须使能ADC时钟*
28. *// 有关ADC更多的信息参见设备数据手册*
29. ADC\_cal();
30. *//本例中使能了所有片内外设模块时钟*
31. SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;   *// I2C*
32. SysCtrlRegs.PCLKCR0.bit.SCIAENCLK = 1;   *// SCI-A*
33. SysCtrlRegs.PCLKCR0.bit.SCIBENCLK = 1;   *// SCI-B*
34. SysCtrlRegs.PCLKCR0.bit.SCICENCLK = 1;   *// SCI-C*
35. SysCtrlRegs.PCLKCR0.bit.SPIAENCLK = 1;   *// SPI-A*
36. SysCtrlRegs.PCLKCR0.bit.MCBSPAENCLK = 1; *// McBSP-A*
37. SysCtrlRegs.PCLKCR0.bit.MCBSPBENCLK = 1; *// McBSP-B*
38. SysCtrlRegs.PCLKCR0.bit.ECANAENCLK=1;    *// eCAN-A*
39. SysCtrlRegs.PCLKCR0.bit.ECANBENCLK=1;    *// eCAN-B*
40. SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;   *// Disable TBCLK within the ePWM*
41. SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  *// ePWM1*
42. SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;  *// ePWM2*
43. SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;  *// ePWM3*
44. SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;  *// ePWM4*
45. SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;  *// ePWM5*
46. SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;  *// ePWM6*
47. SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;   *// Enable TBCLK within the ePWM*
48. SysCtrlRegs.PCLKCR1.bit.ECAP3ENCLK = 1;  *// eCAP3*
49. SysCtrlRegs.PCLKCR1.bit.ECAP4ENCLK = 1;  *// eCAP4*
50. SysCtrlRegs.PCLKCR1.bit.ECAP5ENCLK = 1;  *// eCAP5*
51. SysCtrlRegs.PCLKCR1.bit.ECAP6ENCLK = 1;  *// eCAP6*
52. SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 1;  *// eCAP1*
53. SysCtrlRegs.PCLKCR1.bit.ECAP2ENCLK = 1;  *// eCAP2*
54. SysCtrlRegs.PCLKCR1.bit.EQEP1ENCLK = 1;  *// eQEP1*
55. SysCtrlRegs.PCLKCR1.bit.EQEP2ENCLK = 1;  *// eQEP2*
56. SysCtrlRegs.PCLKCR3.bit.CPUTIMER0ENCLK = 1; *// CPU Timer 0*
57. SysCtrlRegs.PCLKCR3.bit.CPUTIMER1ENCLK = 1; *// CPU Timer 1*
58. SysCtrlRegs.PCLKCR3.bit.CPUTIMER2ENCLK = 1; *// CPU Timer 2*
59. SysCtrlRegs.PCLKCR3.bit.DMAENCLK = 1;       *// DMA Clock*
60. SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;     *// XTIMCLK*
61. SysCtrlRegs.PCLKCR3.bit.GPIOINENCLK = 1;    *// GPIO input clock*
62. EDIS;
63. }

# 二 配置GPIO

## 1.GPIO概念

GPIO（ general purpose intput output） 是通用输入输出端口的简称，可以通过软件来控制其输入和输出。 F28335 芯片的 GPIO 引脚与外部设备连接起来， 从而实现与外部通讯、 控制以及数据采集的功能。

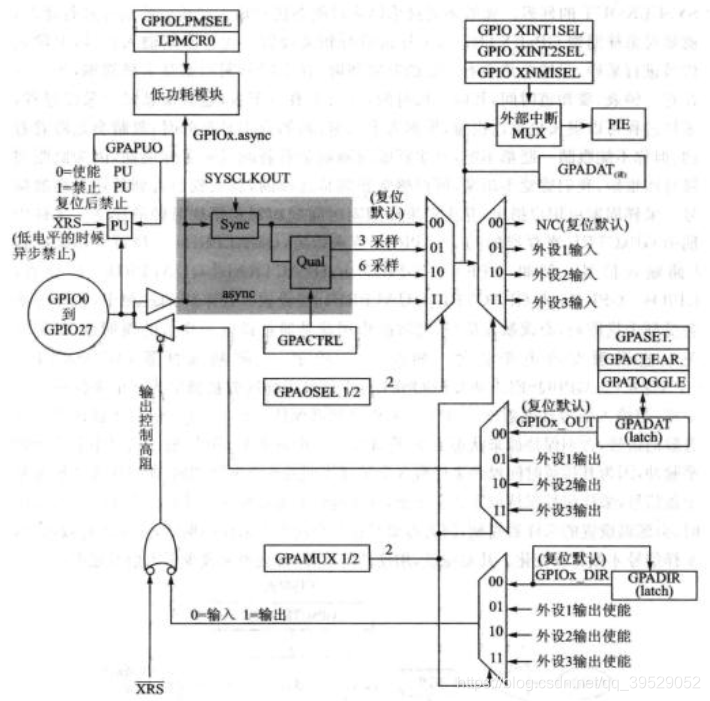
F28335中GPIO 占用了芯片大部分的引脚， 共达 88 个， 分为了 3 组。

A组：GPIO0——GPIO31

B组：GPIO32——GPIO63

C组：GPIO64——GPIO87

每一个GPIO都具备复用功能，只要通过相应的寄存器设置即可配置对应的复用功能，同一时刻，每个引脚只能使用该引脚的一个功能。



GPIO的结构图如上所示。图中最左侧圈内GPIO0~GPIO27为GPIO引脚。在其上方位置有个PU，表示PULL UP是上拉的意思。即GPIO引脚是通过软件可编程控制其电平是否上拉，控制寄存器为GPAUD，0的时候使能上拉，1的时候禁止上拉。

上拉的作用：可以让管脚默认有个初始的电平（高电平），同时增强输出能力。

两个三角形：指向外面的三角形是输出，由GPADIR寄存器控制，在作为输出引脚时，可以看到输出的线连到了一个输出控制的上面，由多路复用寄存器GPAMUX控制，每个管脚有四种功能：

00代表普通的GPIO的功能，01、10/11分别代表不同的外设输出。

GPIO的输出数据存放在GPADAT寄存器中，还可以通过GPASET等寄存器设置该GPADAT的值。

如对一系列的管脚控制输出，使用GPADAT寄存器；通常使用的是GPASET、GPACLEAR、GPATOGGLE等寄存器。

指向内部的三角形是输入，GPIO输入后经滤波电路，引脚的功能选择由多路选择器控制，且输入的数据存入GPADAT中，输入模式下只能读GPADAT的值。在上方的是GPIO输入引脚引入的相关PIE中断。

## 2.GPIO主要配置寄存器

### （1）GPxCTRL寄存器

为输入引脚指定采样周期，GPACTRL为输入引脚0-31指定采样周期。

### （2）GPAQSEL寄存器

用来配置采样数，也可以认为是滤波数，当干扰信号持续采样周期小于该寄存器设置的采样周期时，干扰信号被滤除。

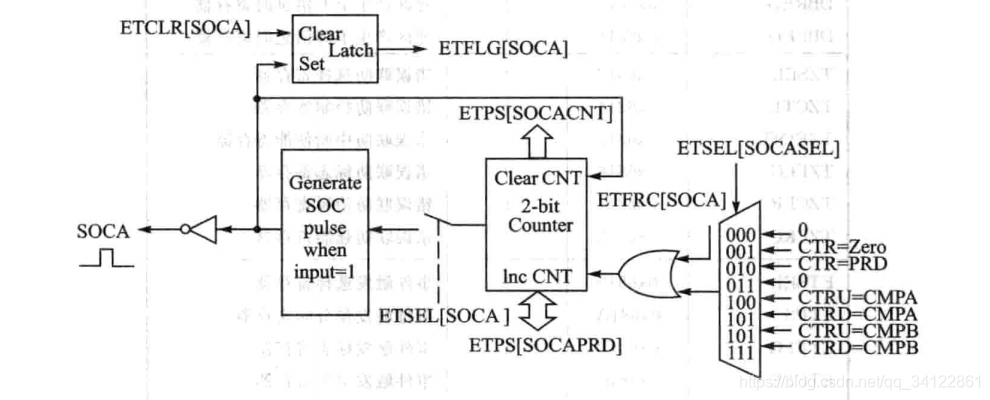
### （3）GPxDIR寄存器

用来配置GPIO的输入输出方向。

## 3.GPIO主要配置过程（ADC）

这里通过ADC的例程来观察GPIO的配置方式

### 1 SOC产生原理



### 2 EPWM1SOCA启动ADC配置

**2.1 使能ADC、TB、ePWM1外设时钟**

1. EALLOW;
2. SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;    *// ADC*
3. SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;   *// Disable TBCLK within the ePWM*
4. SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  *// ePWM1*
5. EDIS;

**2.2 配置ADC时钟为25MHz**

1. EALLOW;
2. SysCtrlRegs.HISPCP.all = 3; *// HSPCLK = SYSCLKOUT/6*
3. EDIS;

**2.3 修改ADCINT的ISR地址**

1. EALLOW;  *// This is needed to write to EALLOW protected register*
2. PieVectTable.ADCINT = &adc\_isr;
3. EDIS;    *// This is needed to disable write to EALLOW protected registers*

**2.4 初始化ADC**

使能ADC外设时钟 （已配置，屏蔽），设置为顺序采样。

1. void InitAdc(void)
2. {
3. extern void DSP28x\_usDelay(Uint32 Count);
4. *// EALLOW;*
5. *// SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;*
6. *// ADC\_cal();*
7. *// EDIS;*
8. AdcRegs.ADCTRL3.all = 0x00E0;  *// Power up bandgap/reference/ADC circuits*
9. DELAY\_US(ADC\_usDELAY);         *// Delay before converting ADC channels*
10. }

**2.5 使能中断ADCINT**

1. *// Enable ADCINT in PIE*
2. PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
3. IER |= M\_INT1; *// Enable CPU Interrupt 1*
4. EINT;          *// Enable Global interrupt INTM*
5. ERTM;          *// Enable Global realtime interrupt DBGM*

**2.6 配置ADC**

设置为两路采样：第一路采ADCINA3，第二路采ADCINA2。

允许ePWMxSOCA启动SEQ1，允许SEQ1中断。

1. *// Configure ADC*
2. AdcRegs.ADCMAXCONV.all = 0x0001;       *// Setup 2 conv's on SEQ1*
3. AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x3; *// Setup ADCINA3 as 1st SEQ1 conv.*
4. AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x2; *// Setup ADCINA2 as 2nd SEQ1 conv.*
5. AdcRegs.ADCTRL2.bit.EPWM\_SOCA\_SEQ1 = 1;*// Enable SOCA from ePWM to start SEQ1*
6. AdcRegs.ADCTRL2.bit.INT\_ENA\_SEQ1 = 1;  *// Enable SEQ1 interrupt (every EOS)*

**2.7 配置ePWM1**

使能ePWMxSOCA。

当TBCTR=CMPA且为增计数时，产生ePWMxSOCA。

每发生一次事件，便产生一次ADC启动信号。

CMPA、TBPRD赋值，EPwm1工作在增计数模式。

1. *// Assumes ePWM1 clock is already enabled in InitSysCtrl();*
2. EPwm1Regs.ETSEL.bit.SOCAEN = 1;        *// Enable SOC on A group*
3. EPwm1Regs.ETSEL.bit.SOCASEL = 4;       *// Select SOC from from CPMA on upcount*
4. EPwm1Regs.ETPS.bit.SOCAPRD = 1;        *// Generate pulse on 1st event*
5. EPwm1Regs.CMPA.half.CMPA = 0x0080;   *// Set compare A value*
6. EPwm1Regs.TBPRD = 0xFFFF;              *// Set period for ePWM1*
7. EPwm1Regs.TBCTL.bit.CTRMODE = 0;    *// count up and start*

**2.8 ADCINT ISR**

1. interrupt void  adc\_isr(void)
2. {
3. Voltage1[ConversionCount] = AdcRegs.ADCRESULT0 >>4;
4. Voltage2[ConversionCount] = AdcRegs.ADCRESULT1 >>4;
5. *// If 40 conversions have been logged, start over*
6. if(ConversionCount == 9)
7. {
8. ConversionCount = 0;
9. }
10. else ConversionCount++;
11. *// Reinitialize for next ADC sequence*
12. AdcRegs.ADCTRL2.bit.RST\_SEQ1 = 1;         *// Reset SEQ1*
13. AdcRegs.ADCST.bit.INT\_SEQ1\_CLR = 1;       *// Clear INT SEQ1 bit*
14. PieCtrlRegs.PIEACK.all = PIEACK\_GROUP1;   *// Acknowledge interrupt to PIE*
15. return;
16. }

### 3 TI EPWM1SOCA启动ADC源码

EPWM1SOC启动ADC，SEQ1转换完成后产生中断ADCINT。

1. #include "DSP28x\_Project.h"     *// Device Headerfile and Examples Include File*
2. *// Prototype statements for functions found within this file.*
3. interrupt void adc\_isr(void);
4. *// Global variables used in this example:*
5. Uint16 LoopCount;
6. Uint16 ConversionCount;
7. Uint16 Voltage1[10];
8. Uint16 Voltage2[10];
9. main()
10. {
11. *// Step 1. Initialize System Control:*
12. *// PLL, WatchDog, enable Peripheral Clocks*
13. *// This example function is found in the DSP2833x\_SysCtrl.c file.*
14. InitSysCtrl();
16. *// Define ADCCLK clock frequency ( less than or equal to 25 MHz )*
17. *// Assuming InitSysCtrl() has set SYSCLKOUT to 150 MHz*
18. EALLOW;
19. SysCtrlRegs.HISPCP.all = ADC\_MODCLK;
20. SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;    *// ADC*
21. SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;   *// Disable TBCLK within the ePWM*
22. SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  *// ePWM1*
23. EDIS;
24. *// Step 2. Initialize GPIO:*
25. *// This example function is found in the DSP2833x\_Gpio.c file and*
26. *// illustrates how to set the GPIO to it's default state.*
27. *// InitGpio();  // Skipped for this example*
28. *// Step 3. Clear all interrupts and initialize PIE vector table:*
29. *// Disable CPU interrupts*
30. DINT;
31. *// Initialize the PIE control registers to their default state.*
32. *// The default state is all PIE interrupts disabled and flags*
33. *// are cleared.*
34. *// This function is found in the DSP2833x\_PieCtrl.c file.*
35. InitPieCtrl();
36. *// Disable CPU interrupts and clear all CPU interrupt flags:*
37. IER = 0x0000;
38. IFR = 0x0000;
39. *// Initialize the PIE vector table with pointers to the shell Interrupt*
40. *// Service Routines (ISR).*
41. *// This will populate the entire table, even if the interrupt*
42. *// is not used in this example.  This is useful for debug purposes.*
43. *// The shell ISR routines are found in DSP2833x\_DefaultIsr.c.*
44. *// This function is found in DSP2833x\_PieVect.c.*
45. InitPieVectTable();
46. *// Interrupts that are used in this example are re-mapped to*
47. *// ISR functions found within this file.*
48. EALLOW;  *// This is needed to write to EALLOW protected register*
49. PieVectTable.ADCINT = &adc\_isr;
50. EDIS;    *// This is needed to disable write to EALLOW protected registers*
51. *// Step 4. Initialize all the Device Peripherals:*
52. *// This function is found in DSP2833x\_InitPeripherals.c*
53. *// InitPeripherals(); // Not required for this example*
54. InitAdc();  *// For this example, init the ADC*
55. *// Step 5. User specific code, enable interrupts:*
56. *// Enable ADCINT in PIE*
57. PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
58. IER |= M\_INT1; *// Enable CPU Interrupt 1*
59. EINT;          *// Enable Global interrupt INTM*
60. ERTM;          *// Enable Global realtime interrupt DBGM*
61. LoopCount = 0;
62. ConversionCount = 0;
63. *// Configure ADC*
64. AdcRegs.ADCMAXCONV.all = 0x0001;       *// Setup 2 conv's on SEQ1*
65. AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x3; *// Setup ADCINA3 as 1st SEQ1 conv.*
66. AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x2; *// Setup ADCINA2 as 2nd SEQ1 conv.*
67. AdcRegs.ADCTRL2.bit.EPWM\_SOCA\_SEQ1 = 1;*// Enable SOCA from ePWM to start SEQ1*
68. AdcRegs.ADCTRL2.bit.INT\_ENA\_SEQ1 = 1;  *// Enable SEQ1 interrupt (every EOS)*
69. *// Assumes ePWM1 clock is already enabled in InitSysCtrl();*
70. EPwm1Regs.ETSEL.bit.SOCAEN = 1;        *// Enable SOC on A group*
71. EPwm1Regs.ETSEL.bit.SOCASEL = 4;       *// Select SOC from from CPMA on upcount*
72. EPwm1Regs.ETPS.bit.SOCAPRD = 1;        *// Generate pulse on 1st event*
73. EPwm1Regs.CMPA.half.CMPA = 0x0080;   *// Set compare A value*
74. EPwm1Regs.TBPRD = 0xFFFF;              *// Set period for ePWM1*
75. EPwm1Regs.TBCTL.bit.CTRMODE = 0;    *// count up and start*
76. *// Wait for ADC interrupt*
77. for(;;)
78. {
79. LoopCount++;
80. }
81. }

### 4 定时器Timer0启动配置

**4.1 使能定时器Timer0外设时钟**

1. EALLOW;
2. *// 使能定时器时钟*
3. SysCtrlRegs.PCLKCR3.bit.CPUTIMER0ENCLK = 1; *// CPU Timer 0*
4. EDIS;

**4.2 修改TINT0的ISR地址**

1. EALLOW;
2. *// 修改TINT0的ISR地址*
3. PieVectTable.TINT0 = &TIM0\_IRQn;
4. EDIS;

**4.3 初始化定时器Timer0**

1. *// 指定定时器0寄存器*
2. CpuTimer0.RegsAddr = &CpuTimer0Regs;
3. *// 周期值PRD*
4. CpuTimer0Regs.PRD.all  = 0xFFFFFFFF;
5. *// 预定标计数值TPR*
6. CpuTimer0Regs.TPR.all  = 0;
7. CpuTimer0Regs.TPRH.all = 0;
8. *// 停止定时器0*
9. CpuTimer0Regs.TCR.bit.TSS = 1;
10. *// 将周期值PRD装入计数器TIM*
11. CpuTimer0Regs.TCR.bit.TRB = 1;
12. *// 初始化定时器0中断次数*
13. CpuTimer0.InterruptCount = 0;

**4.4 配置定时器Timer0**

CPUTIMER\_VARS \*Timer为定时器，Freq为系统时钟（MHz），Period为周期 （us）.

Period即为中断TINT0触发的间隔。

1. void ConfigCpuTimer(struct CPUTIMER\_VARS \*Timer, float Freq, float Period)
2. {
3. Uint32  temp;
4. *// Initialize timer period:*
5. Timer->CPUFreqInMHz = Freq;
6. Timer->PeriodInUSec = Period;
7. temp = (long) (Freq \* Period);
8. Timer->RegsAddr->PRD.all = temp;
9. *// Set pre-scale counter to divide by 1 (SYSCLKOUT):*
10. Timer->RegsAddr->TPR.all  = 0;
11. Timer->RegsAddr->TPRH.all  = 0;
12. *// Initialize timer control register:*
13. Timer->RegsAddr->TCR.bit.TSS = 1;      *// 1 = Stop timer, 0 = Start/Restart Timer*
14. Timer->RegsAddr->TCR.bit.TRB = 1;      *// 1 = reload timer*
15. Timer->RegsAddr->TCR.bit.SOFT = 0;
16. Timer->RegsAddr->TCR.bit.FREE = 0;     *// Timer Free Run Disabled*
17. Timer->RegsAddr->TCR.bit.TIE = 1;      *// 0 = Disable/ 1 = Enable Timer Interrupt*
18. *// Reset interrupt counter:*
19. Timer->InterruptCount = 0;
20. }

**4.5 启动定时器Timer0**

1. *// 启动定时器0*
2. CpuTimer0Regs.TCR.bit.TSS=0;

**4.6 使能中断TINT0**

1. *// 使能TINT0*
2. IER |= M\_INT1;
3. PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
4. EINT;
5. ERTM;

**4.7 配置ADC**

设置为单路采样：采ADCINA0。

1. void ADC\_Init(void)
2. {
3. EALLOW;
4. SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;    *// ADC*
5. EDIS;
6. *// Specific clock setting for this example:*
7. EALLOW;
8. SysCtrlRegs.HISPCP.all = ADC\_MODCLK; *// HSPCLK = SYSCLKOUT/2\*ADC\_MODCLK*
9. EDIS;
10. InitAdc();  *// For this example, init the ADC*
12. AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; *// Setup ADCINA0 as 1st SEQ1 conv.*
13. AdcRegs.ADCMAXCONV.bit.MAX\_CONV1= 0x0;  *// Setup 2 conv's on SEQ1*
14. AdcRegs.ADCTRL2.bit.EPWM\_SOCA\_SEQ1 = 1;*// Enable SOCA from ePWM to start SEQ1*
15. AdcRegs.ADCTRL2.bit.INT\_ENA\_SEQ1 = 1;  *// Enable SEQ1 interrupt (every EOS)*
16. }

**4.8 TINT0 ISR**

启动SEQ1

1. interrupt void TIM0\_IRQn(void)
2. {
3. AdcRegs.ADCTRL2.bit.SOC\_SEQ1 = 1;
4. PieCtrlRegs.PIEACK.bit.ACK1=1;
5. }

### 5 Timer0启动ADC源码

中断TINT0启动ADC，采用查询方式读取转换结果。

当有SEQ1中断标志位产生，清除后读取采样结果。

同连续采样的软件触发。

1. #include "DSP2833x\_Device.h"     *// DSP2833x Headerfile Include File*
2. #include "DSP2833x\_Examples.h"   *// DSP2833x Examples Include File*
3. *// 采样值*
4. float adc\_vol;
5. void main()
6. {
7. InitSysCtrl();
8. InitPieCtrl();
9. IER = 0x0000;
10. IFR = 0x0000;
11. InitPieVectTable();
12. TIM0\_Init(150,1000);
13. ADC\_Init();
14. while(1)
15. {
16. while (AdcRegs.ADCST.bit.INT\_SEQ1 == 0);
17. AdcRegs.ADCST.bit.INT\_SEQ1\_CLR = 1;
18. adc\_vol=(float)(AdcRegs.ADCRESULT0>>4)\*3.0/4096;
19. SMG\_DisplayFloat(adc\_vol,2);
20. }
21. }