数字与模拟转换2

程晨闻

东南大学电气工程学院

➢ 内容回顾

– 数字/模拟转换器DAC 的工作原理

- 译码特点
- 倒T型网络DAC
- 电阻串结构DAC
- DAC指标（转换精度，分辨率，线性度，偏移，转换时间，毛刺，温度系数…）

– 模拟/数字转换器ADC的工作原理

- 采样，保持，量化，编码
- 计数器式，逐次逼近式，双积分式，并行式，Delta-Sigma…
- ADC指标（分辨率，转换速度，精度，量化误差，偏移误差，满刻度误差，线性度…）

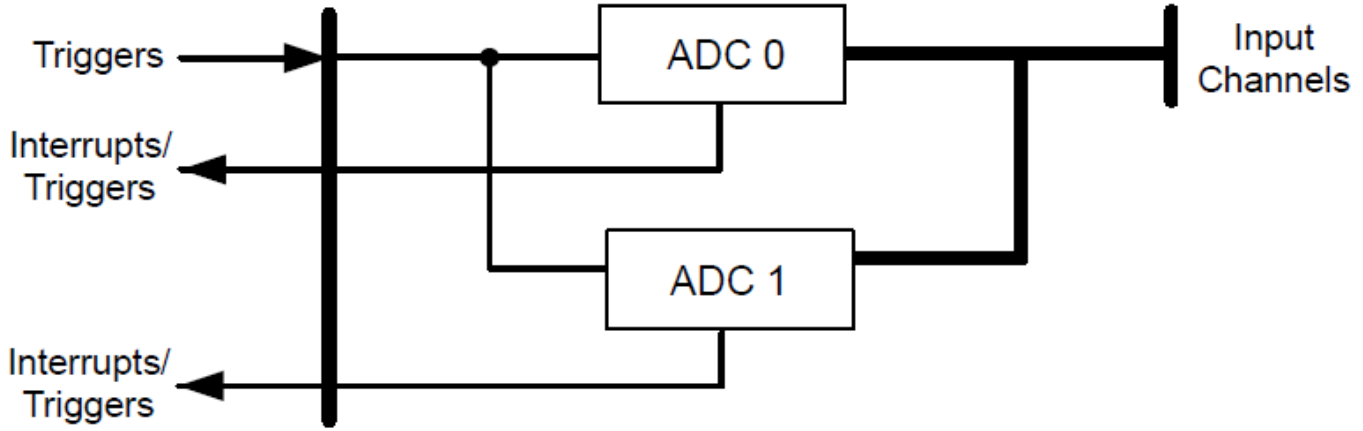
内容概要

– TM4C1294的ADC模块及其使用方法

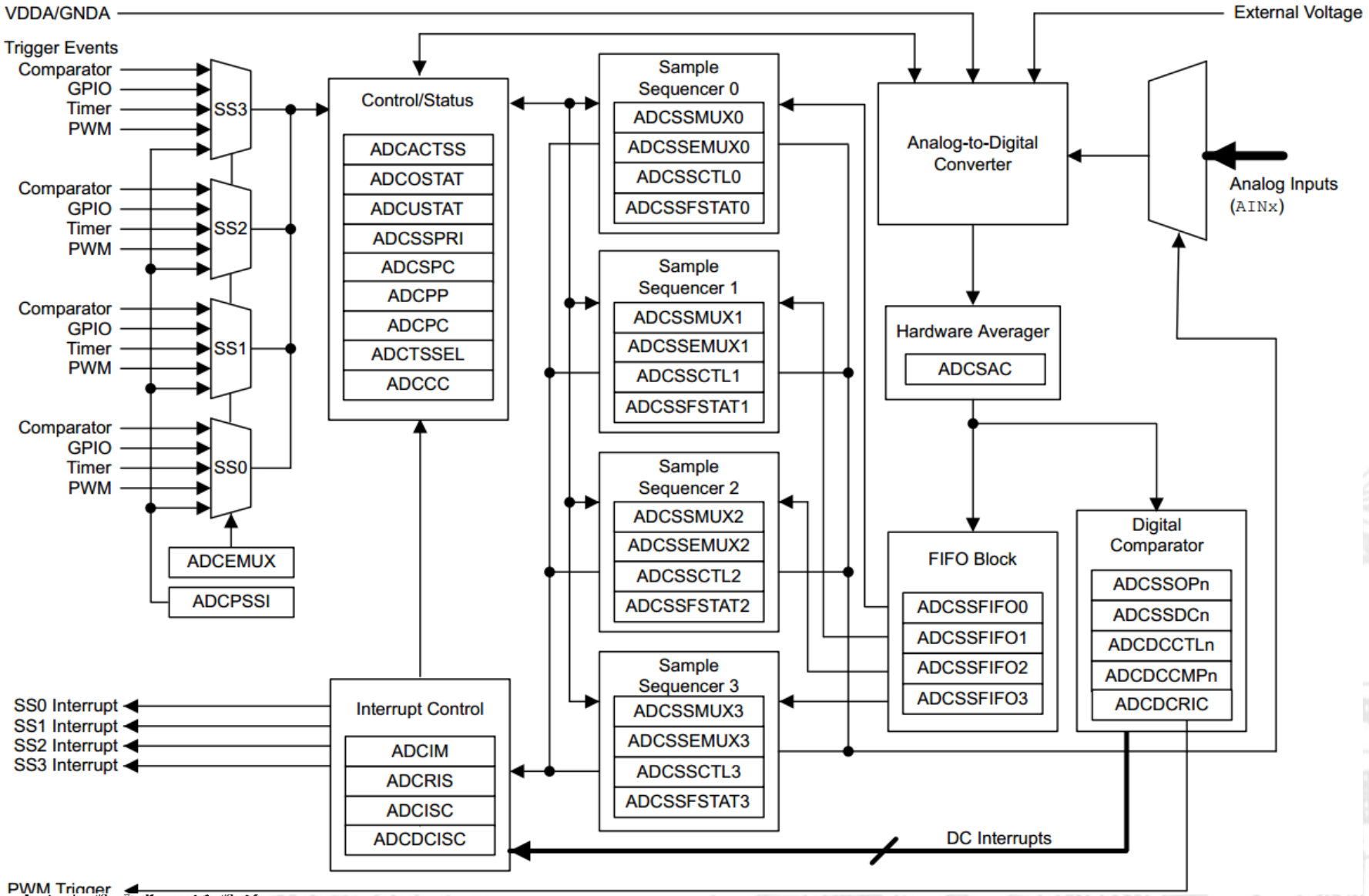
TM4C的ADC模块

- 有两个12位逐次逼近型ADC，ADC0和ADC1
- 两个ADC共享20路模拟信号和一个片内温度传感器信号
- 每个ADC的最高采样频率为2Msps，采样一次需要至少16个ADC时钟，12个采样周期+ 4个保持周期
- 以采样序列为单位进行采样，有四个可编程采样序列，每个序列的结果都有FIFO
- 序列采样完成后可以产生中断

Figure 15-1. Implementation of Two ADC Blocks



ADC模块的基本结构


VDDA/GNDA
External Voltage
Trigger Events
Comparator
GPIO
Timer
PWM
SS3
SS2
SS1
SS0
ADCEMUX
ADCPSSI
Control/Status
ADCACTSS
ADCOSTAT
ADCUSTAT
ADCSSPRI
ADCSPC
ADCPP
ADCPC
ADCTSSEL
ADCCC
Sample Sequencer 0
ADCSSMUX0
ADCSSEMUX0
ADCSSCTL0
ADCSSFSTAT0
Sample Sequencer 1
ADCSSMUX1
ADCSSEMUX1
ADCSSCTL1
ADCSSFSTAT1
Sample Sequencer 2
ADCSSMUX2
ADCSSEMUX2
ADCSSCTL2
ADCSSFSTAT2
Sample Sequencer 3
ADCSSMUX3
ADCSSEMUX3
ADCSSCTL3
ADCSSFSTAT3
Analog-to-Digital Converter
Analog Inputs (AINx)
Hardware Averager
ADCSAC
FIFO Block
ADCSSFIFO0
ADCSSFIFO1
ADCSSFIFO2
ADCSSFIFO3
Digital Comparator
ADCSSOPn
ADCSSDCn
ADCDCCTLn
ADCDCCMPn
ADCDCRIC
SS0 Interrupt
SS1 Interrupt
SS2 Interrupt
SS3 Interrupt
Interrupt Control
ADCIM
ADCRIS
ADCISC
ADCDCISC
DC Interrupts
PWM Trigger

TM4C1294的ADC模块的使用方法

- 1. 在系统设置模块中，使能ADC模块的时钟
- 2. 使能使用到的ADC引脚所在的GPIO模块的时钟
- 3. 将GPIO的引脚配置为模拟输入
- 4. 设置ADC模块的采样速率
- 5. 配置采样序列
- 6. 使能并配置中断
- 7. 开始采样
- 8. 编写中断服务函数，读取数据

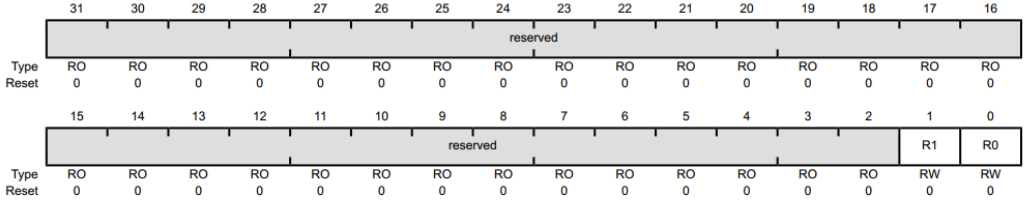
➢ TM4C1294的ADC模块的使用方法

– **1. 在系统设置模块中，使能ADC模块的时钟**

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```

Analog-to-Digital Converter Run Mode Clock Gating Control (RCGCADC)

Base 0x400F.E000
Offset 0x638
Type RW, reset 0x0000.0000



Bit/Field	Name	Type	Reset	Description
31:2	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
1	R1	RW	0	ADC Module 1 Run Mode Clock Gating Control Value Description 0 ADC module 1 is disabled. 1 Enable and provide a clock to ADC module 1 in Run mode.

```
#define SYSCTL_RCGCBASE         0x400fe600
#define SYSCTL_PERIPH_ADC0      0xf0003800  // ADC 0
```

```
void
SysCtlPeripheralEnable(uint32_t ui32Peripheral)
{
    // Check the arguments.
    ASSERT(_SysCtlPeripheralValid(ui32Peripheral));
    // Enable this peripheral.
    HWREGBITW(SYSCTL_RCGCBASE + ((ui32Peripheral & 0xff00) >> 8),
              ui32Peripheral & 0xff) = 1;
}
```

➢ TM4C1294的ADC模块的使用方法

– 2. 使能使用到的ADC引脚所在的GPIO模块的时钟

- ADC输入引脚与GPIO引脚的对应关系为

Table 15-1. ADC Signals (128TQFP)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
AIN0	12	PE3	I	Analog	Analog-to-digital converter input 0.
AIN1	13	PE2	I	Analog	Analog-to-digital converter input 1.
AIN2	14	PE1	I	Analog	Analog-to-digital converter input 2.
AIN3	15	PE0	I	Analog	Analog-to-digital converter input 3.
AIN4	128	PD7	I	Analog	Analog-to-digital converter input 4.
AIN5	127	PD6	I	Analog	Analog-to-digital converter input 5.
AIN6	126	PD5	I	Analog	Analog-to-digital converter input 6.
AIN7	125	PD4	I	Analog	Analog-to-digital converter input 7.
AIN8	124	PE5	I	Analog	Analog-to-digital converter input 8.
AIN9	123	PE4	I	Analog	Analog-to-digital converter input 9.
AIN10	121	PB4	I	Analog	Analog-to-digital converter input 10.
AIN11	120	PB5	I	Analog	Analog-to-digital converter input 11.
AIN12	4	PD3	I	Analog	Analog-to-digital converter input 12.
AIN13	3	PD2	I	Analog	Analog-to-digital converter input 13.
AIN14	2	PD1	I	Analog	Analog-to-digital converter input 14.
AIN15	1	PD0	I	Analog	Analog-to-digital converter input 15.
AIN16	18	PK0	I	Analog	Analog-to-digital converter input 16.
AIN17	19	PK1	I	Analog	Analog-to-digital converter input 17.
AIN18	20	PK2	I	Analog	Analog-to-digital converter input 18.
AIN19	21	PK3	I	Analog	Analog-to-digital converter input 19.
VREFA+	9	fixed	-	Analog	A reference voltage used to specify the voltage at which the ADC converts to a maximum value. This pin is used in conjunction with GNDA. The voltage that is applied to VREFA+ is the voltage with which an AINn signal is converted to 4095. The VREFA+ voltage is limited to the range specified in Table 27-44 on page 1861.

TM4C1294的ADC模块的使用方法

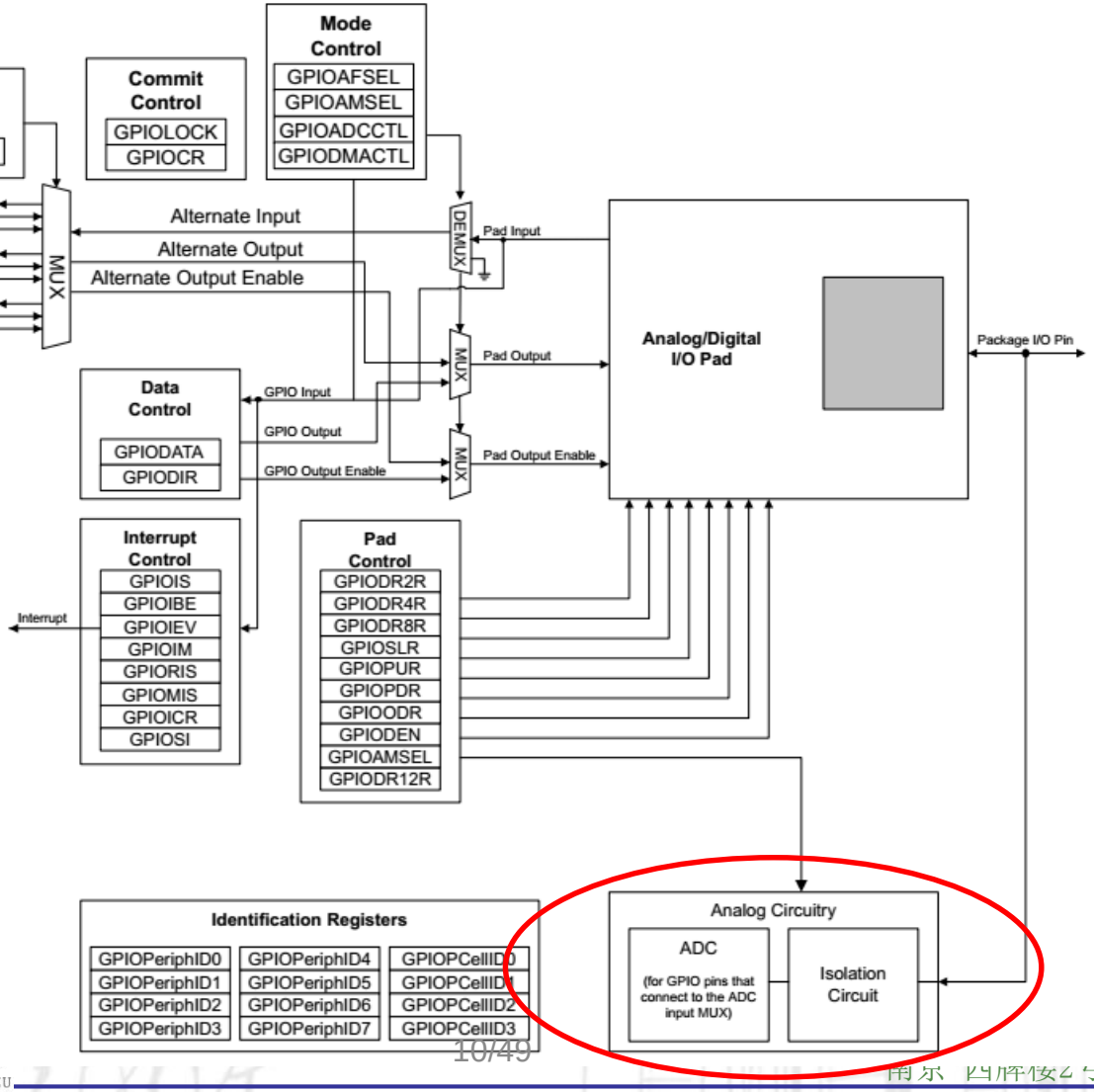
– **2. 使能使用到的ADC引脚所在的GPIO模块的时钟**

- ADC输入引脚与GPIO引脚的对应关系为

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
```

TM4C1294的ADC模块的使用方法

– **3. 将GPIO的引脚配置为模拟输入**



➢ TM4C1294的ADC模块的使用方法

– **3. 将GPIO的引脚配置为模拟输入**

GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0);

```
void
GPIOPinTypeADC(uint32_t ui32Port, uint8_t ui8Pins)
{
    //
    // Check the arguments.
    //
    ASSERT(_GPIOBaseValid(ui32Port));

    //
    // Make the pin(s) be inputs.
    //
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_IN);

    //
    // Set the pad(s) for analog operation.
    //
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_ANALOG);
}
```

➢ TM4C1294的ADC模块的使用方法

– **3. 将GPIO的引脚配置为模拟输入**

14.2.3.18 GPIOPadConfigSet

Sets the pad configuration for the specified pin(s).

Prototype:

```
void
GPIOPadConfigSet(uint32_t ui32Port,
                 uint8_t ui8Pins,
                 uint32_t ui32Strength,
                 uint32_t ui32PinType)
```

Parameters:

ui32Port is the base address of the GPIO port.

ui8Pins is the bit-packed representation of the pin(s).

ui32Strength specifies the output drive strength.

ui32PinType specifies the pin type.

The parameter *ui32PinType* can be one of the following values:

- **GPIO_PIN_TYPE_STD**
- **GPIO_PIN_TYPE_STD_WPU**
- **GPIO_PIN_TYPE_STD_WPD**
- **GPIO_PIN_TYPE_OD**
- **GPIO_PIN_TYPE_ANALOG**
- **GPIO_PIN_TYPE_WAKE_HIGH**
- **GPIO_PIN_TYPE_WAKE_LOW**

where **GPIO_PIN_TYPE_STD**∗ specifies a push-pull pin, **GPIO_PIN_TYPE_OD**∗ specifies an open-drain pin, ∗**_WPU** specifies a weak pull-up, ∗**_WPD** specifies a weak pull-down, and **GPIO_PIN_TYPE_ANALOG** specifies an analog input.

The **GPIO_PIN_TYPE_WAKE_**∗ settings specify the pin to be used as a hibernation wake source. The pin sense level can be high or low. These settings are only available on some Tiva devices.

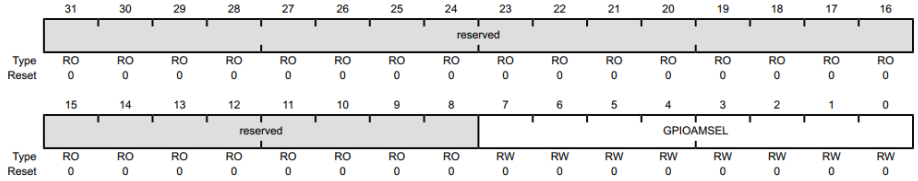
TM4C1294的ADC模块的使用方法

– 3. 将GPIO的引脚配置为模拟输入

Register 21: GPIO Analog Mode Select (GPIOAMSEL), offset 0x528

GPIO Analog Mode Select (GPIOAMSEL)

GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (AHB) base: 0x4005.D000
GPIO Port G (AHB) base: 0x4005.E000
GPIO Port H (AHB) base: 0x4005.F000
GPIO Port J (AHB) base: 0x4006.0000
GPIO Port K (AHB) base: 0x4006.1000
GPIO Port L (AHB) base: 0x4006.2000
GPIO Port M (AHB) base: 0x4006.3000
GPIO Port N (AHB) base: 0x4006.4000
GPIO Port P (AHB) base: 0x4006.5000
GPIO Port Q (AHB) base: 0x4006.6000
Offset 0x528
Type RW, reset 0x0000.0000



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	GPIOAMSEL	RW	0x00	GPIO Analog Mode Select

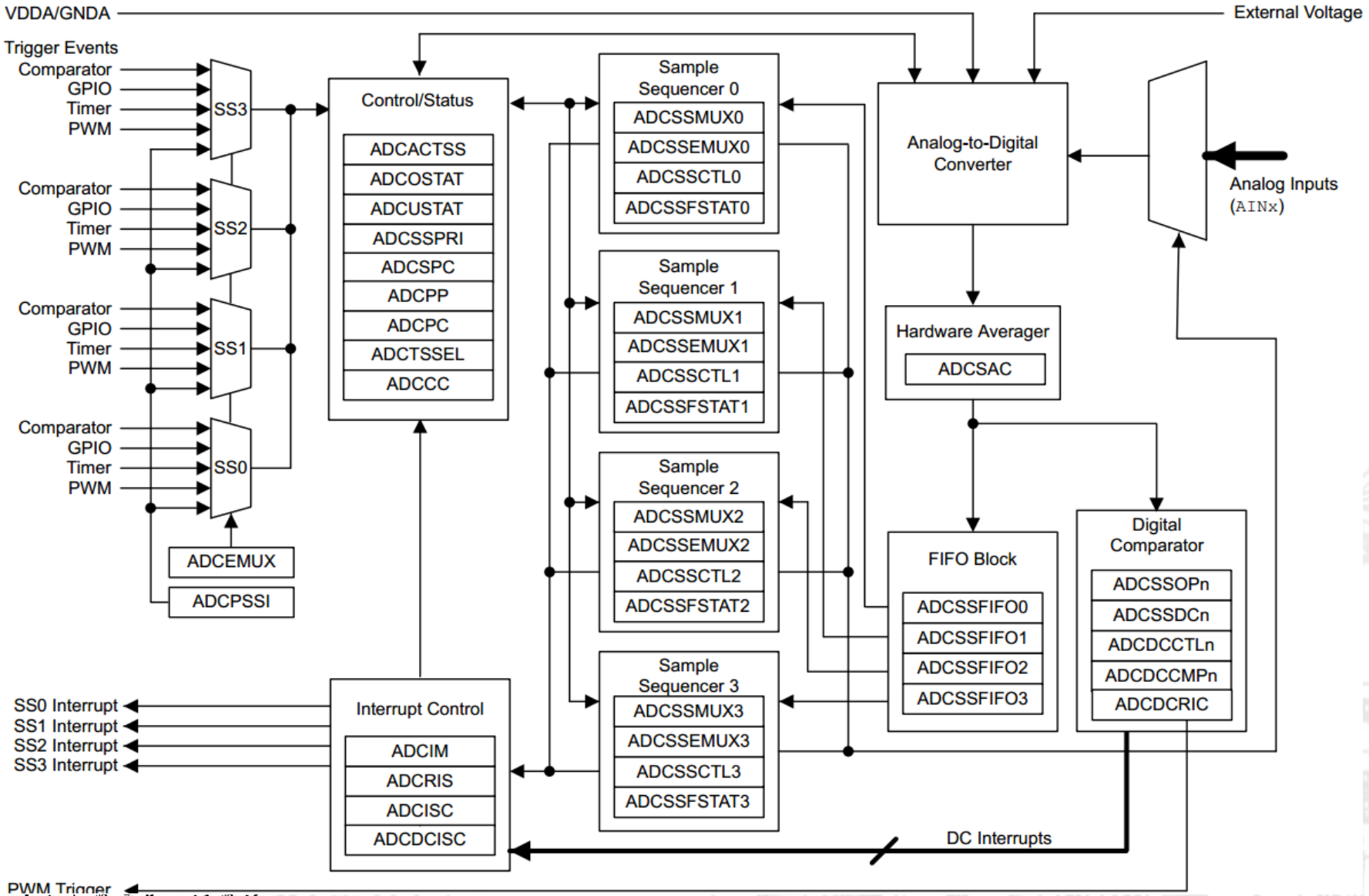
Value	Description
0	The analog function of the pin is disabled, the isolation is enabled, and the pin is capable of digital functions as specified by the other GPIO configuration registers.
1	The analog function of the pin is enabled, the isolation is disabled, and the pin is capable of analog functions.

Note: This register and bits are only valid for GPIO signals that share analog function through a unified I/O pad.

The reset state of this register is 0 for all signals.

```
void
GPIOPadConfigSet(uint32_t ui32Port, uint8_t ui8Pins,
                 uint32_t ui32Strength, uint32_t ui32PinType)
{
    // …
    // Set the analog mode select register.
    HWREG(ui32Port + GPIO_O_AMSEL) =
          ((ui32PinType == GPIO_PIN_TYPE_ANALOG) ?
           (HWREG(ui32Port + GPIO_O_AMSEL) | ui8Pins) :
           (HWREG(ui32Port + GPIO_O_AMSEL) & ~(ui8Pins)));
}
```

ADC模块的基本结构

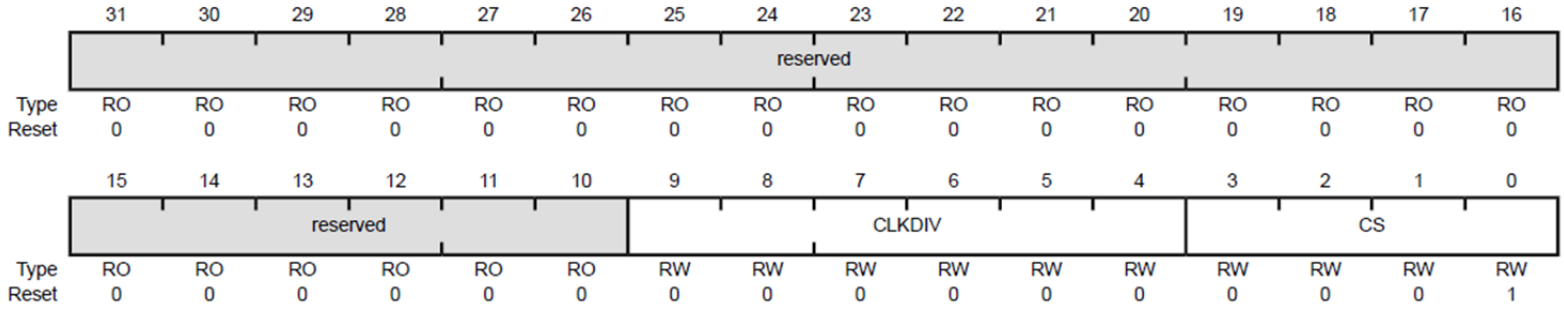

VDDA/GNDA
External Voltage
Trigger Events
Comparator
GPIO
Timer
PWM
SS3
SS2
SS1
SS0
ADCEMUX
ADCPSSI
Control/Status
ADCACTSS
ADCOSTAT
ADCUSTAT
ADCSSPRI
ADCSPC
ADCPP
ADCPC
ADCTSSEL
ADCCC
Sample Sequencer 0
ADCSSMUX0
ADCSSEMUX0
ADCSSCTL0
ADCSSFSTAT0
Sample Sequencer 1
ADCSSMUX1
ADCSSEMUX1
ADCSSCTL1
ADCSSFSTAT1
Sample Sequencer 2
ADCSSMUX2
ADCSSEMUX2
ADCSSCTL2
ADCSSFSTAT2
Sample Sequencer 3
ADCSSMUX3
ADCSSEMUX3
ADCSSCTL3
ADCSSFSTAT3
Analog-to-Digital Converter
Analog Inputs (AINx)
Hardware Averager
ADCSAC
FIFO Block
ADCSSFIFO0
ADCSSFIFO1
ADCSSFIFO2
ADCSSFIFO3
Digital Comparator
ADCSSOPn
ADCSSDCn
ADCDCCTLn
ADCDCCMPn
ADCDCRIC
SS0 Interrupt
SS1 Interrupt
SS2 Interrupt
SS3 Interrupt
Interrupt Control
ADCIM
ADCRIS
ADCISC
ADCDCISC
DC Interrupts
PWM Trigger

➢ TM4C1294的ADC模块的使用方法

– **4. 设置ADC模块的采样速率**

有两个寄存器影响ADC的采样速率

ADC Clock Configuration (**ADCCC**)寄存器



CLKDIV域：PLL VCO的分频系数

PLL VCO Clock Divisor

Value	Description
0x0	/1
0x1	/2
0x2	/3
0xN	/(N + 1)

CS域：时钟源选择

Value	Description
0x0	PLL VCO divided by CLKDIV.
0x1	Alternate clock source as defined by **ALTCLKCFG** register in System Control Module.
0x2	MOSC
0x2 - 0xF	Reserved

VCO只有两个选项：480MHz或者320MHz

```
ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                   SYSCTL_OSC_MAIN |
                                   SYSCTL_USE_PLL |
                                   SYSCTL_CFG_VCO_480), 120000000);
```

TM4C1294的ADC模块的使用方法

– **4. 设置ADC模块的采样速率**

有两个寄存器影响ADC的采样速率

ADC Clock Configuration (**ADCCC**)寄存器

CLKDIV域：PLL VCO的分频系数

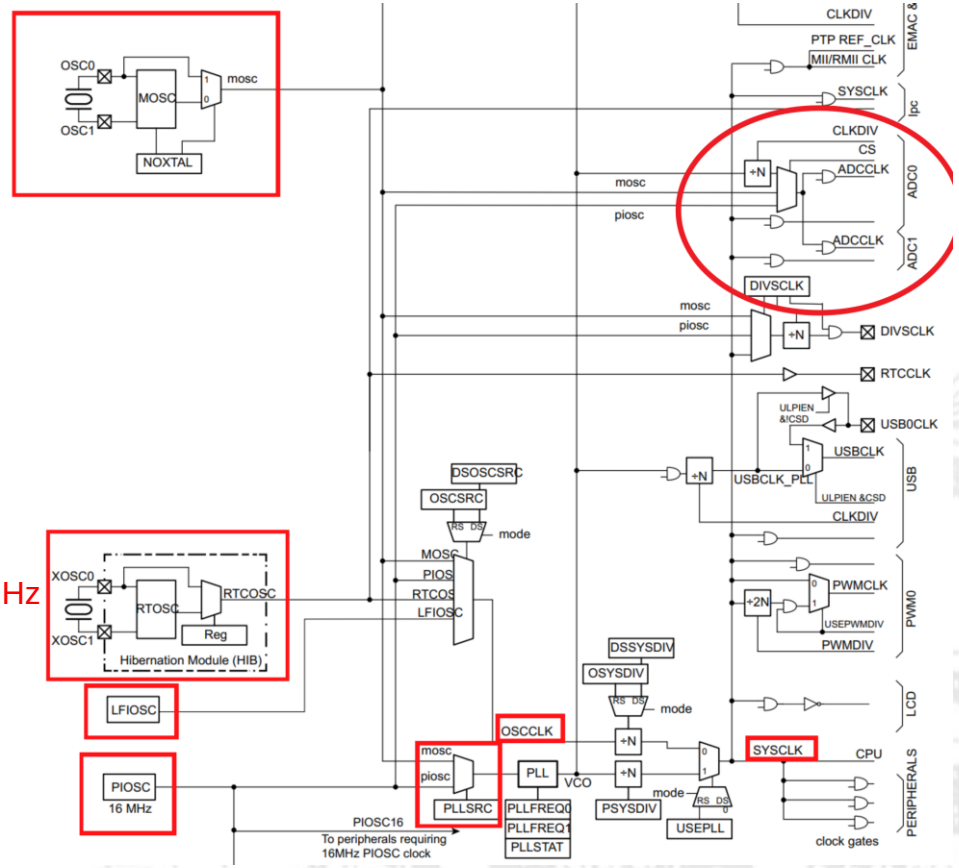
PLL VCO Clock Divisor

Value	Description
0x0	/1
0x1	/2
0x2	/3
0xN	/(N + 1)

CS域：时钟源选择

Value	Description
0x0	PLL VCO divided by CLKDIV.
0x1	Alternate clock source as defined by **ALTCLKCFG** register in System Control Module.
0x2	MOSC
0x2 - 0xF	Reserved

VCO只有两个选项：480MHz或者320MHz

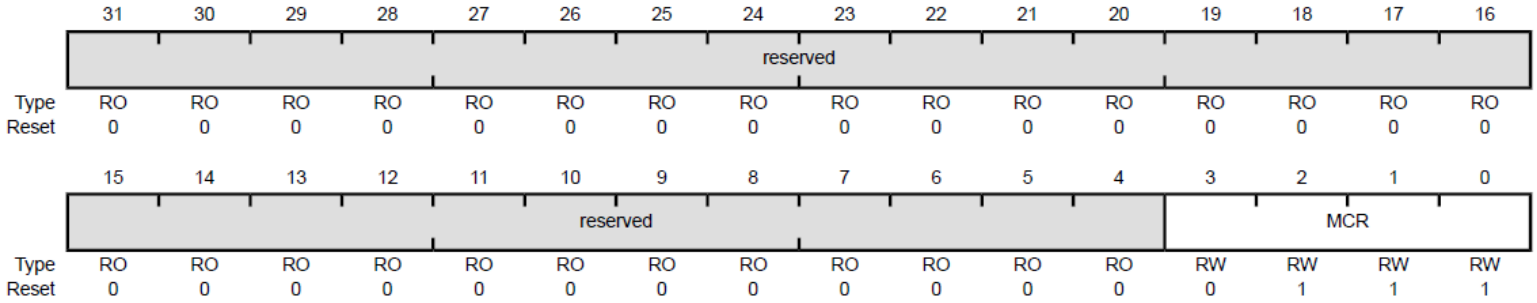


➢ TM4C1294的ADC模块的使用方法

– 4. 设置ADC模块的采样速率

有两个寄存器影响ADC的采样速率

ADC Peripheral Configuration (**ADCPC**)寄存器



用于在两次采样之间，插入一定的时间，降低采样速率。

MCR域：

0x07：全速模式，两次采样之间不插入时间
0x05：半速模式，两次采样之间插入16个ADC时钟周期
0x03：1/4速模式，两次采样之间插入48个ADC时钟周期
0x01：1/8速模式，两次采样之间插入112个ADC时钟周期

TM4C1294的ADC模块的使用方法

– 4. 设置ADC模块的采样速率

使用TivaWare提供的ADCClockConfigSet函数，设置ADC的采样速率

```
void ADCClockConfigSet(uint32_t ui32Base, uint32_t ui32Config, uint32_t ui32ClockDiv)
{
    // …
    HWREG(ui32Base + ADC_O_PC) = (ui32Config >> 4) & ADC_PC_SR_M;
    HWREG(ui32Base + ADC_O_CC) = (ui32Config & ADC_CC_CS_M) |
                                 (((ui32ClockDiv - 1) << ADC_CC_CLKDIV_S)) ;
}
```

```
#define ADC_PC_SR_M   0x0000000F  // ADC Sample Rate
#define ADC_CC_CLKDIV_S  4
#define ADC_CC_CS_M  0x0000000F  // ADC Clock Source
```

由于两个ADC**共享**ADC时钟，所以第一个参数只能是ADC0_BASE

ADCClockConfigSet函数的第二个参数用于设置ADCCC寄存器的**CS**域和ADCPC寄存器的**MCR**域

ADCCC寄存器的**CS**域可选的参数为：　#define ADC_CLOCK_SRC_PLL　0x00000000

ADC_CLOCK_SRC_PLL - The main PLL output.
ADC_CLOCK_SRC_PIOSC - The internal PIOSC at 16 MHz.
ADC_CLOCK_SRC_ALTCLK - The output of the ALTCLK in the system
ADC_CLOCK_SRC_MOSC - The external MOSC .

ADCPC寄存器的**MCR**域可选的参数为：　#define ADC_CLOCK_RATE_FULL　0x00000070

ADC_CLOCK_RATE_FULL - All samples.
ADC_CLOCK_RATE_HALF - Every other sample.
ADC_CLOCK_RATE_QUARTER - Every fourth sample.
ADC_CLOCK_RATE_EIGHTH - Every eitghth sample.

➢ TM4C1294的ADC模块的使用方法

– 4. 设置ADC模块的采样速率

如果系统时钟初始化函数为：

```
ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                   SYSCTL_OSC_MAIN |
                                   SYSCTL_USE_PLL |
                                   SYSCTL_CFG_VCO_480), 120000000);
```

ADC时钟初始化函数为：

```
ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PLL | ADC_CLOCK_RATE_FULL, 24);
```

那么ADC时钟为：480MHz/24=20MHz。

最高采样速率为20MHz/16=1.25Msps。

➢ TM4C1294的ADC模块的使用方法

– 4. 设置ADC模块的采样速率

如果系统时钟初始化函数为：

```
ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                   SYSCTL_OSC_MAIN |
                                   SYSCTL_USE_PLL |
                                   SYSCTL_CFG_VCO_480), 120000000);
```

如果要设置ADC的最高采样频率为2Msps，ADC时钟初始化函数应该怎样写？

ADC时钟初始化函数为：

```
ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PLL | ADC_CLOCK_RATE_FULL, 15);
```

➢ TM4C1294的ADC模块的使用方法

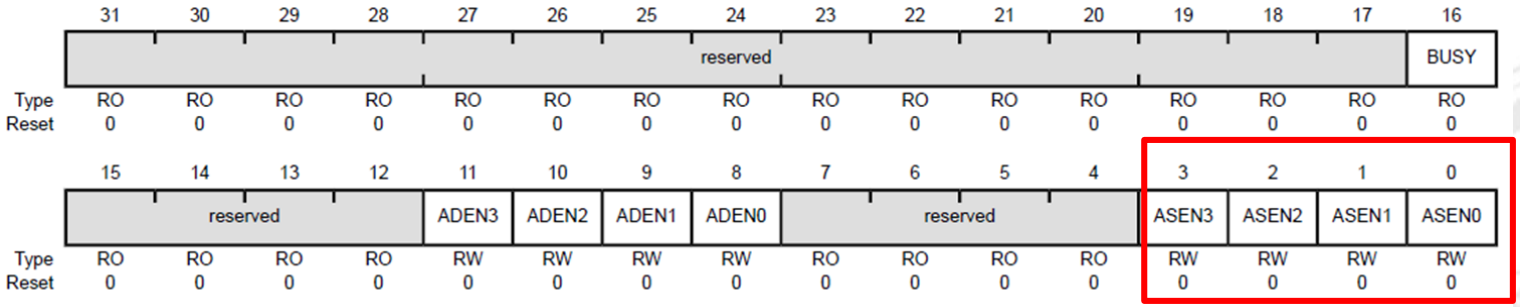
– 5. 配置采样序列

5.1 选择并设置ADC的采样序列

ADC模块有四个采样序列

Sequencer	Number of Samples	Depth of FIFO
SS3	1	1
SS2	4	4
SS1	4	4
SS0	8	8

ADC Active Sample Sequencer (**ADCACTSS**)寄存器用于控制四个采样序列的使能



ASEN0：SS0的使能位；
ASEN1：SS1的使能位；
ASEN2：SS2的使能位；
ASEN3：SS3的使能位；

➢ TM4C1294的ADC模块的使用方法

– **5. 配置采样序列**

5.1 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceEnable函数，来设置ADCACTSS寄存器，使能采样序列

```
void
ADCSequenceEnable(uint32_t ui32Base, uint32_t ui32SequenceNum)
{
    //
    // Check the arguments.
    //
    ASSERT((ui32Base == ADC0_BASE) || (ui32Base == ADC1_BASE));
    ASSERT(ui32SequenceNum < 4);

    //
    // Enable the specified sequence.
    //
    HWREG(ui32Base + ADC_O_ACTSS) |= 1 << ui32SequenceNum;
}
```

```
ADCSequenceEnable(ADC0_BASE, 0);     //使能ADC0模块的SS0采样序列
```

➢ TM4C1294的ADC模块的使用方法

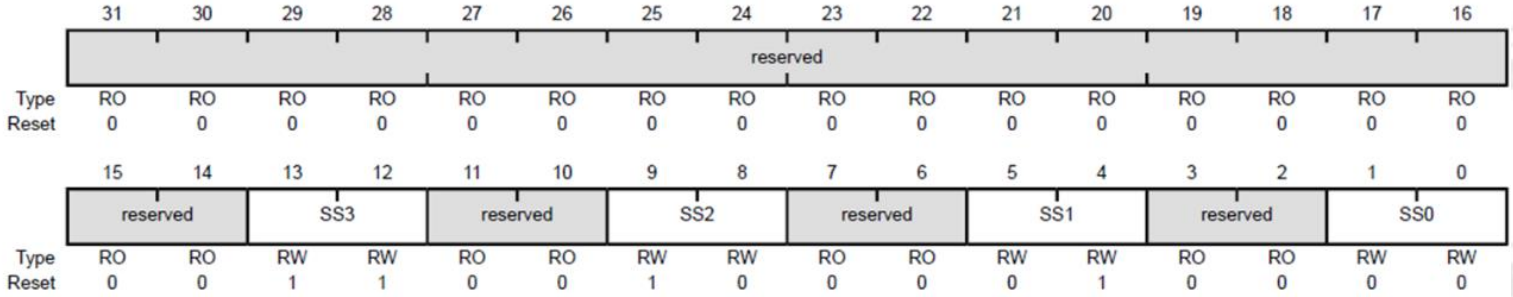
– 5. 配置采样序列

5.1 选择并设置ADC的采样序列

ADC模块有四个采样序列

Sequencer	Number of Samples	Depth of FIFO
SS3	1	1
SS2	4	4
SS1	4	4
SS0	8	8

由ADC Sample Sequencer Priority (**ADCSSPRI**) ADC采样序列优先级寄存器决定采样序列的优先级



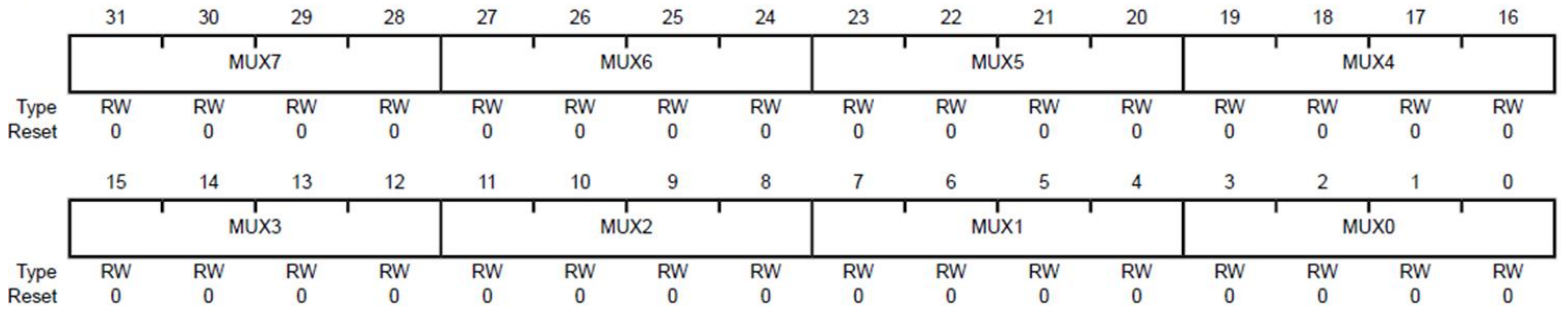
- 域SS0到SS3分别配置采样序列SS0到SS3的优先级
- 数值越低，优先级越高
- 默认情况下，SS0的优先级最高

➢ TM4C1294的ADC模块的使用方法

– **5. 配置采样序列**

5.2 选择采样序列中需要转换的通道，并设置采样属性

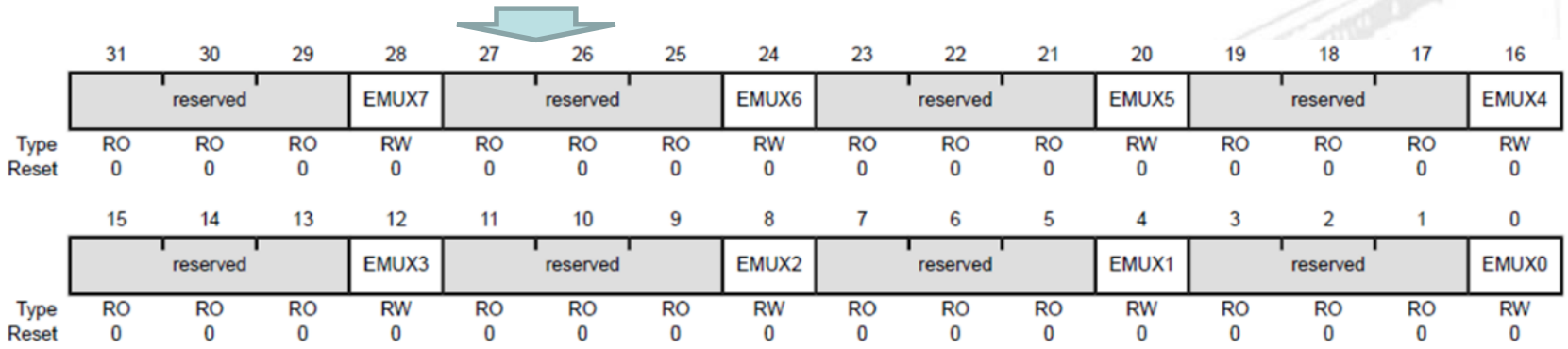
由ADC Sample Sequence Input Multiplexer Select 0（**ADCSSMUX0**）ADC采样序列复选寄存器决定要转换的通道



每四位控制一个AD转换的采样通道，值可以使0-15。假设MUXn的值为**m**

如果**ADCSSEMUX0**.EMUXn为0：MUXn表示相应的AIN**m**通道

如果**ADCSSEMUX0**.EMUXn为1：MUXn表示相应的AIN(**m**+16)通道



ADCSSEMUX0寄存器

➢ TM4C1294的ADC模块的使用方法：

– 5. 配置采样序列

5.2 选择采样序列中需要转换的通道，并设置采样属性

一个采样序列中，一共要进行多少次采样，由ADC Sample Sequence Control 0 (**ADCSSCTL0**) ADC采样序列控制寄存器决定

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TS7	IE7	END7	D7	TS6	IE6	END6	D6	TS5	IE5	END5	D5	TS4	IE4	END4	D4
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TS3	IE3	END3	D3	TS2	IE2	END2	D2	TS1	IE1	END1	D1	TS0	IE0	END0	D0
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADCSSCTL0寄存器中，每4位为1组，决定了一个采样通道的属性，一组中的4个位有不用的含义：

- TSn:表示第n次采样为片内**温度采样**
- IEn: 表示此次采样完成后**触发序列中断**
- ENDn: 本次采样为序列中的**最后一次**采样
- Dn: 本次采样为**差分采样**，采样通道为2i和2i+1

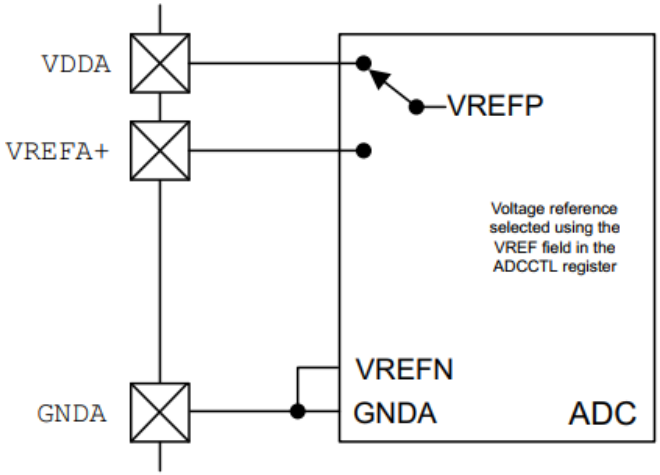
TM4C1294的ADC模块的使用方法

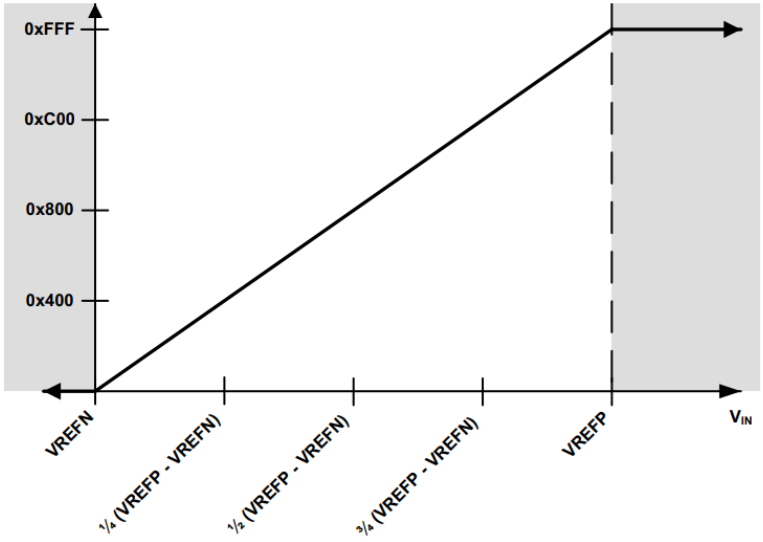
– 5. 配置采样序列

5.2 选择采样序列中需要转换的通道，并设置采样属性

一个采样序列中，一共要进行多少次采样，由ADC Sample Sequence Control 0 (**ADCSSCTL0**) ADC采样序列控制寄存器决定

单端采样：转换结果为单个采样通道的值





- If VIN = 0, 转换结果= 0x000
- If VIN = 0.5*(VREFP - VREFN), 转换结果= 0x800
- If VIN = VREFP, 转换结果= 0xFFF

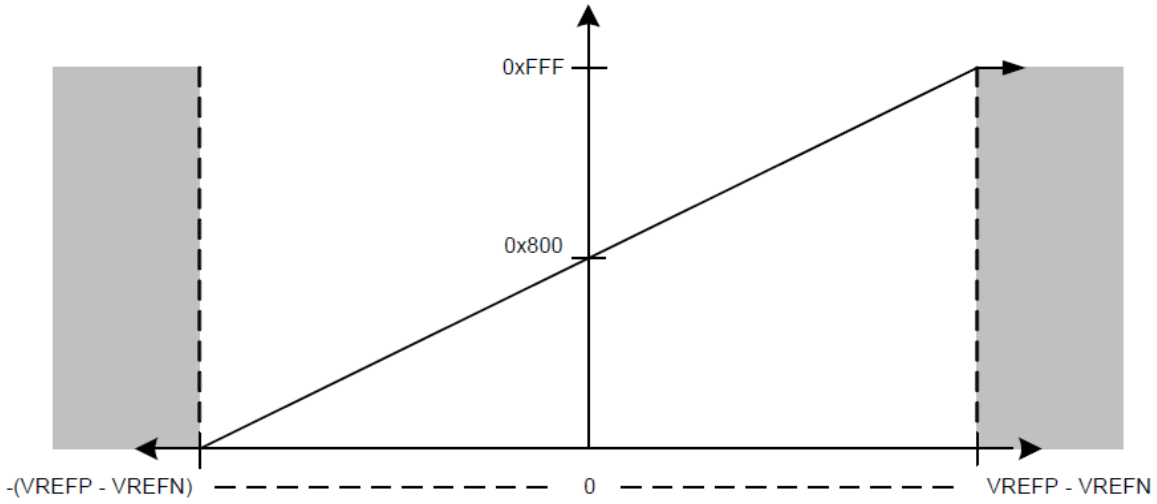
TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.2 选择采样序列中需要转换的通道，并设置采样属性

一个采样序列中，一共要进行多少次采样，由ADC Sample Sequence Control 0 (**ADCSSCTL0**) ADC采样序列控制寄存器决定

差分采样：转换结果为两个采样通道（2i和2i+1）的差



- If VIND = 0, 转换结果= 0x800
- If VIND > 0, 转换结果> 0x800 (range is 0x800–0xFFF)
- If VIND < 0, 转换结果< 0x800 (range is 0–0x800)

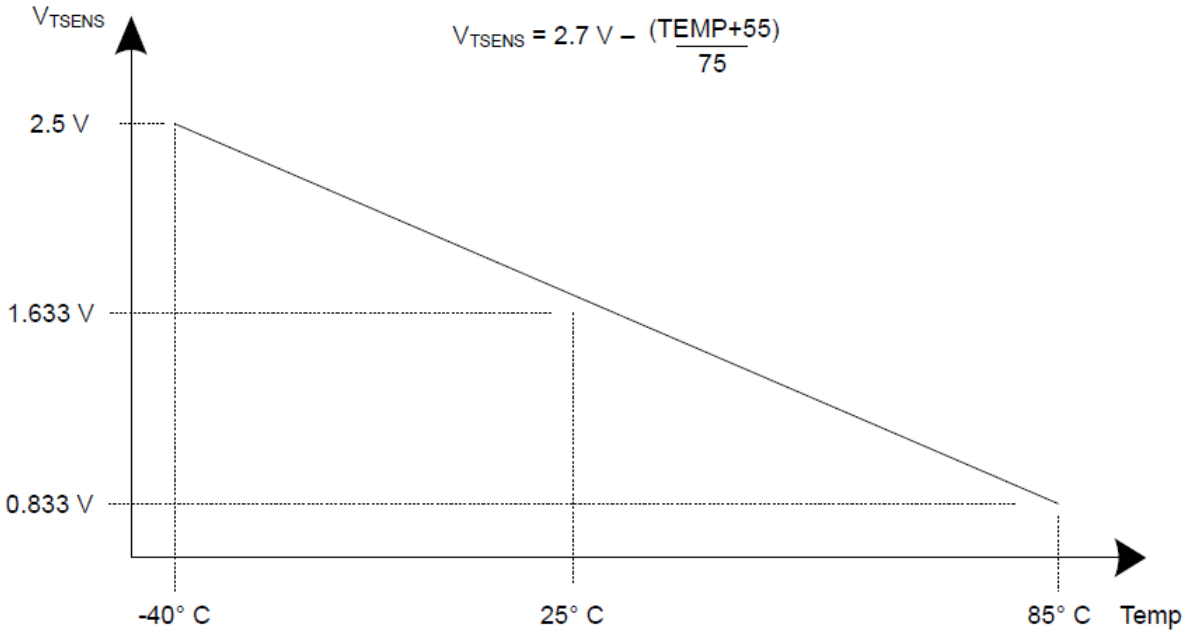
TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.2 选择采样序列中需要转换的通道，并设置采样属性

一个采样序列中，一共要进行多少次采样，由ADC Sample Sequence Control 0 (**ADCSSCTL0**) ADC采样序列控制寄存器决定

温度采样：



TEMP = 147.5 - ((75 * (VREFP - VREFN) × ADCCODE) / 4096)

VREFP默认3.3V，VREFN默认0V。

➢ TM4C1294的ADC模块的使用方法

– **5.** 配置采样序列

5.2 选择采样序列中需要转换的通道，并设置采样属性

ADC Sample Sequence 0 Sample and Hold Time (**ADCSSTSH0**)寄存器，用于设置每次采样的保持时间：

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TSH7				TSH6				TSH5				TSH4			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TSH3				TSH2				TSH1				TSH0			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TSHn Encoding	N_{SH}
0x0	4
0x1	reserved
0x2	8
0x3	reserved
0x4	16
0x5	reserved
0x6	32
0x7	reserved
0x8	64
0x9	reserved
0xA	128
0xB	reserved
0xC	256
0xD-0xF	reserved

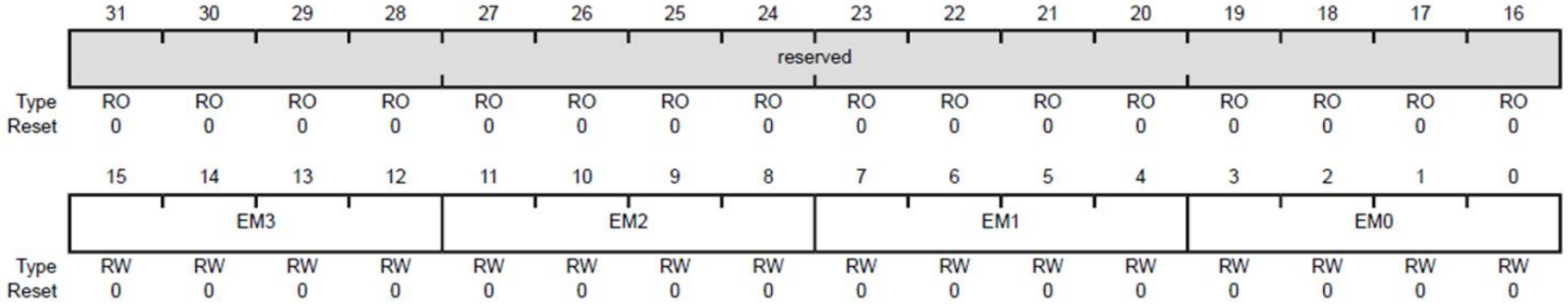
默认为4个ADC时钟周期

➢ TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.3 选择采样序列开始转换的条件（触发条件）

ADC Event Multiplexer Select (**ADCEMUX**) ADC事件选择寄存器用于设置四个采样序列的触发条件:



EMn域由4位组成，不同的值代表不同的触发条件:

0x00: 由 ADCPSSI. SSn软件手动触发
0x01: 触发条件由ACCTL0配置
0x02: 触发条件由ACCTL1配置
0x03: 触发条件由ACCTL2配置
0x04: 由GPIO引脚触发，由GPIO模块中断的GPIOADCCTL寄存器配置
0x05: 由定时器触发
0x06: 由PWM0中断触发
0x07: 由PWM1中断触发
0x08: 由PWM2中断触发
0x09: 由PWM3中断触发
0x0E: 不触发
0x0F: 连续触发

➢ TM4C1294的ADC模块的使用方法

– **5. 配置采样序列**

5.3 选择采样序列开始转换的条件（触发条件）

ADC Processor Sample Sequence Initiate (**ADCPSSI**) ADC采样序列初始化寄存器，使用户可以用软件手动触发ADC的采样序列。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	GSYNC	reserved			SYNCWAIT	reserved										
Type	RW	RO	RO	RO	RW	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												SS3	SS2	SS1	SS0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	WO	WO	WO	WO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-

- 位SS0到SS3分别控制采样序列SS0到SS3的初始化
- 如果对SSn写1，且SYNCWAIT为零，采样序列初始化后立刻开始采样

➢ TM4C1294的ADC模块的使用方法

– **5. 配置采样序列**

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceConfigure函数，设置采样序列的触发条件和优先级

```
void ADCSequenceConfigure(uint32_t ui32Base, uint32_t ui32SequenceNum,
                          uint32_t ui32Trigger, uint32_t ui32Priority)
{
    uint32_t ui32Gen;
    // …
    // Compute the shift for the bits that control this sample sequence.
    ui32SequenceNum *= 4;
    // Set the trigger event for this sample sequence.
    HWREG(ui32Base + ADC_O_EMUX) = ((HWREG(ui32Base + ADC_O_EMUX) &
                                     ~(0xf << ui32SequenceNum)) |
                                    ((ui32Trigger & 0xf) << ui32SequenceNum));
    // Set the priority for this sample sequence.
    HWREG(ui32Base + ADC_O_SSPRI) = ((HWREG(ui32Base + ADC_O_SSPRI) &
                                      ~(0xf << ui32SequenceNum)) |
                                     ((ui32Priority & 0x3) <<
                                      ui32SequenceNum));
    // Set the source PWM module for this sequence's PWM triggers.
    // …
    }
}
```

TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceConfigure函数，设置采样序列的触发条件和优先级

void ADCSequenceConfigure (uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t ui32Trigger, uint32_t ui32Priority)

ui32Trigger为采样序列的触发方式，设置用于ADCEMUX寄存器，值可以为：

```
#define ADC_TRIGGER_PROCESSOR   0x00000000  // Processor event
#define ADC_TRIGGER_COMP0       0x00000001  // Analog comparator 0 event
#define ADC_TRIGGER_COMP1       0x00000002  // Analog comparator 1 event
#define ADC_TRIGGER_COMP2       0x00000003  // Analog comparator 2 event
#define ADC_TRIGGER_EXTERNAL    0x00000004  // External event
#define ADC_TRIGGER_TIMER       0x00000005  // Timer event
#define ADC_TRIGGER_PWM0        0x00000006  // PWM0 event
#define ADC_TRIGGER_PWM1        0x00000007  // PWM1 event
#define ADC_TRIGGER_PWM2        0x00000008  // PWM2 event
#define ADC_TRIGGER_PWM3        0x00000009  // PWM3 event
#define ADC_TRIGGER_NEVER       0x0000000E  // Never Trigger
#define ADC_TRIGGER_ALWAYS      0x0000000F  // Always event
#define ADC_TRIGGER_PWM_MOD0    0x00000000  // PWM triggers from PWM0
#define ADC_TRIGGER_PWM_MOD1    0x00000010  // PWM triggers from PWM1
```

ui32Priority 为优先级，用于设置ADCSSPRI寄存器，值可以为0, 1, 2, 3，值越小，优先级越高

➢ TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceConfigure函数，设置采样序列的触发条件和优先级

```
ADCSequenceConfigure(ADC0_BASE, 0,ADC_TRIGGER_PROCESSOR, 0);
```

设置ADC0模块的SS0采样序列为软件手动触发，优先级为最高

➢ TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceStepConfigure函数，用于设置采样序列中的采样通道的属性

```
void
ADCSequenceStepConfigure(uint32_t ui32Base, uint32_t ui32SequenceNum,
                         uint32_t ui32Step, uint32_t ui32Config)
{
    // …
    ui32Base += ADC_SEQ + (ADC_SEQ_STEP * ui32SequenceNum);
    // Compute the shift for the bits that control this step.
    ui32Step *= 4;
    // Set the analog mux value for this step.
    HWREG(ui32Base + ADC_SSMUX) = ((HWREG(ui32Base + ADC_SSMUX) &
                                    ~(0x0000000f << ui32Step)) |
                                   ((ui32Config & 0x0f) << ui32Step));
    // Set the upper bits of the analog mux value for this step.
    HWREG(ui32Base + ADC_SSEMUX) = ((HWREG(ui32Base + ADC_SSEMUX) &
                                     ~(0x0000000f << ui32Step)) |
                                    (((ui32Config & 0xf00) >> 8) << ui32Step));
    // Set the control value for this step.
    HWREG(ui32Base + ADC_SSCTL) = ((HWREG(ui32Base + ADC_SSCTL) &
                                    ~(0x0000000f << ui32Step)) |
                                   (((ui32Config & 0xf0) >> 4) << ui32Step));
    // Set the sample and hold time for this step.
    HWREG(ui32Base + ADC_SSTSH) = ((HWREG(ui32Base + ADC_SSTSH) &
                                    ~(0x0000000f << ui32Step)) |
                                  (((ui32Config & 0xf00000) >> 20) << ui32Step));
    // …
}
```

➢ TM4C1294的ADC模块的使用方法

– 5. 配置采样序列

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceStepConfigure函数，用于设置采样序列中的采样通道的属性

void ADCSequenceStepConfigure (uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t ui32Step, uint32_t ui32Config)

ui32Base为ADC模块的基地址

ui32SequenceNum为采样序列号，值可以为0，1，2，3，用于计算该采样序列对应的ADC_SSMUX、ADC_SSEMUX、ADC_SSCTL和ADC_SSTSH等寄存器的地址

ui32Step：采样序列中的第几次采样，对于SS0，这个参数的值可以使0-7。用于计算设置ADC_SSMUX、ADC_SSEMUX、ADC_SSCTL和ADC_SSTSH等寄存器时，需要左移多少位

➢ TM4C1294的ADC模块的使用方法：

– 5. 配置采样序列

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceStepConfigure函数，用于设置采样序列中的采样通道的属性

void ADCSequenceStepConfigure (uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t ui32Step, uint32_t ui32Config)

ui32Config：本次采样的采样通道号，及采样属性、保持时间。

采样通道号可选的值为ADC_CTL_CH0 到ADC_CTL_CH23，用于设置ADC_SSMUX和ADC_SSEMUX寄存器

采样属性的选项有：ADC_CTL_TS，ADC_CTL_IE，ADC_CTL_END，ADC_CTL_D，用于设置ADC_SSCTL寄存器

```
#define ADC_CTL_TS              0x00000080  // Temperature sensor select
#define ADC_CTL_IE              0x00000040  // Interrupt enable
#define ADC_CTL_END             0x00000020  // Sequence end select
#define ADC_CTL_D               0x00000010  // Differential select
```

保持时间用于设置ADC_SSTSH，其选项有：

```
#define ADC_CTL_SHOLD_4         0x00000000  // Sample and hold 4 ADC clocks
#define ADC_CTL_SHOLD_8         0x00200000  // Sample and hold 8 ADC clocks
……
#define ADC_CTL_SHOLD_256       0x00C00000  // Sample and hold 256 ADC clocks
```

将这些选项按位或后，赋值给ui32Config

➢ TM4C1294的ADC模块的使用方法

– **5. 配置采样序列**

5.4 使用TivaWare库提供的函数，配置采样序列

TivaWare提供了ADCSequenceStepConfigure函数，用于设置采样序列中的采样通道的属性

```
ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_TS);
 //SS0序列第0个转换的通道为内部温度传感器的输出电压
```

```
ADCSequenceStepConfigure(ADC0_BASE, 0, 5,
ADC_CTL_CH3|ADC_CTL_IE|ADC_CTL_END);
//SS0序列第5个转换的通道为AIN3引脚的电压，本次转换完成后，触发SS0序列的中断，本次转换为SS0采样序列的最后一次转换。
```

TM4C1294的ADC模块的使用方法

- **6. 使能并配置中断**

6.1 在PIE中使能ADC0模块的SS0序列的中断

ADC0模块的SS0到SS3采样序列，都可以单独向中断管理器发出中断请求

```
IntEnable(INT_ADC0SS0);
```

➢ TM4C1294的ADC模块的使用方法

– 6. 使能并配置中断

6.2 注册中断服务函数

TivaWare提供了ADCIntRegister注册ADC模块指定采样序列的中断服务函数

void ADCIntRegister (uint32_t ui32Base, uint32_t ui32SequenceNum, void (pfnHandler)(void))

ui32Base为ADC模块的基地址;
ui32SequenceNum为采样序列号，值可以为0, 1, 2, 3
pfnHandler 为中断服务函数地址

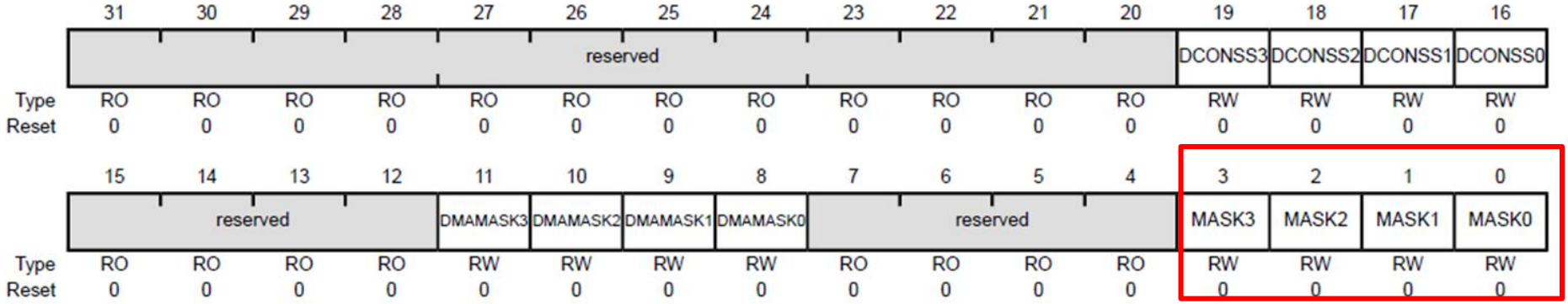
```
ADCIntRegister(ADC0_BASE,0 , adc0_ss0_isr );
```

➢ TM4C1294的ADC模块的使用方法

– 6. 使能并配置中断

6.3 使能ADC模块中用到的采样序列(SS0)的中断

ADC模块的ADC Interrupt Mask (**ADCIM**)寄存器的低四位，控制四个采样序列的中断使能：



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved												DCONSS3	DCONSS2	DCONSS1	DCONSS0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				DMAMASK3	DMAMASK2	DMAMASK1	DMAMASK0	reserved				MASK3	MASK2	MASK1	MASK0
Type	RO	RO	RO	RO	RW	RW	RW	RW	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TivaWare提供了ADCIntEnable函数，通过操作ADCIM寄存器，开启采样序列的中断，其定义为：

```
void
ADCIntEnable(uint32_t ui32Base, uint32_t ui32SequenceNum)
{
// Clear any outstanding interrupts on this sample sequence.
    HWREG(ui32Base + ADC_O_ISC) = 1 << ui32SequenceNum;
// Enable this sample sequence interrupt.
    HWREG(ui32Base + ADC_O_IM) |= 1 << ui32SequenceNum;
}
```

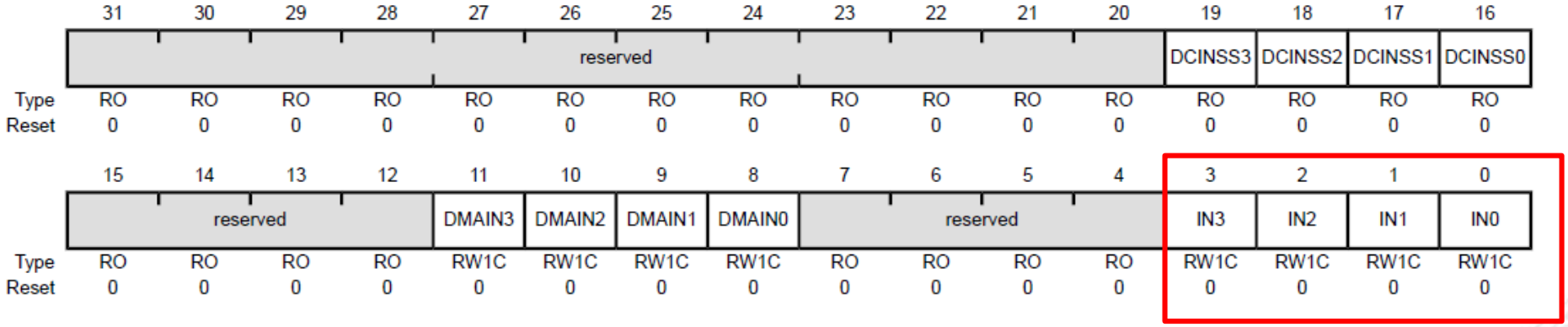
```
ADCIntEnable(ADC0_BASE,0);
```

➢ TM4C1294的ADC模块的使用方法

– 6. 使能并配置中断

6.4 清除ADC模块中用到的采样序列(SS0)的中断

ADC Interrupt Status and Clear (**ADCISC**) ADC中断状态及清除寄存器，可以查看ADC模块中断的状态，并清除中断



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved												DCINSS3	DCINSS2	DCINSS1	DCINSS0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				DMAIN3	DMAIN2	DMAIN1	DMAIN0	reserved				IN3	IN2	IN1	IN0
Type	RO	RO	RO	RO	RW1C	RW1C	RW1C	RW1C	RO	RO	RO	RO	RW1C	RW1C	RW1C	RW1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TivaWare提供了ADCIntClear函数，通过操作ADCISC寄存器，清除ADC模块中指定采样序列的中断标志，其定义为：

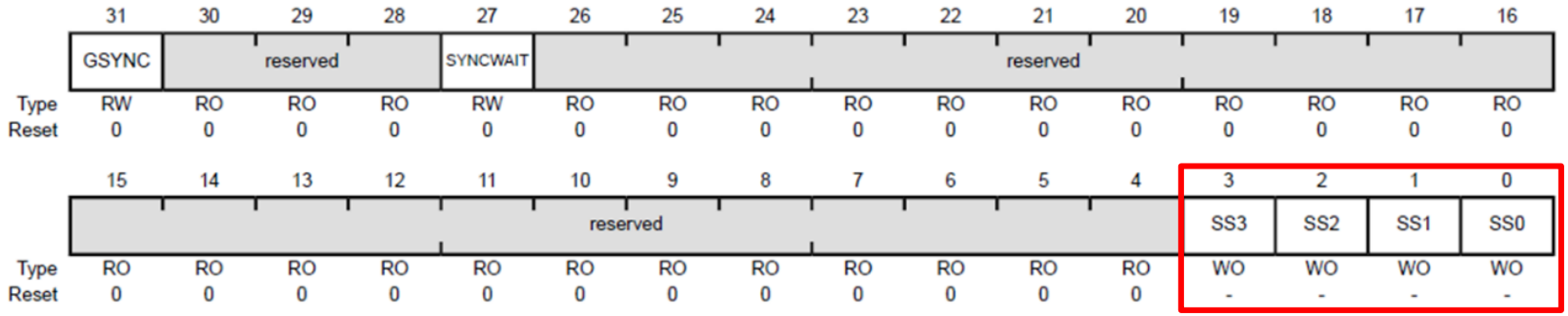
```
void
ADCIntClear(uint32_t ui32Base, uint32_t ui32SequenceNum)
{
    // Clear the interrupt.
HWREG(ui32Base + ADC_O_ISC) = 1 << ui32SequenceNum;
}
```

```
ADCIntClear(ADC0_BASE, 0);
```

TM4C1294的ADC模块的使用方法

– 7. 开始采样

ADC Processor Sample Sequence Initiate (**ADCPSSI**) ADC采样序列初始化寄存器，可以让用户手动触发ADC模块中指定的采样序列。



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	GSYNC	reserved			SYNCWAIT	reserved										
Type	RW	RO	RO	RO	RW	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												SS3	SS2	SS1	SS0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	WO	WO	WO	WO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-

如果ADCEMUX寄存器被配置为软件手动触发，TivaWare提供了ADCProcessorTrigger函数，操作ADCPSSI寄存器，手动触发ADC采样

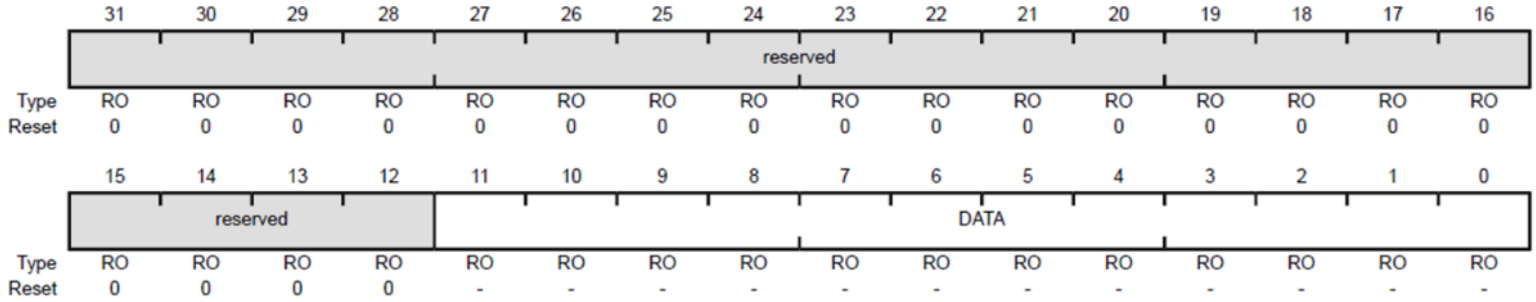
```
void
ADCProcessorTrigger(uint32_t ui32Base, uint32_t ui32SequenceNum)
{
    // Check the arguments.
    ASSERT((ui32Base == ADC0_BASE) || (ui32Base == ADC1_BASE));
    ASSERT(ui32SequenceNum < 4);
    // Generate a processor trigger for this sample sequence.
    HWREG(ui32Base + ADC_O_PSSI) |= ((ui32SequenceNum & 0xffff0000) |
                                     (1 << (ui32SequenceNum & 0xf)));
}
```

```
ADCProcessorTrigger(ADC0_BASE, 0);
```

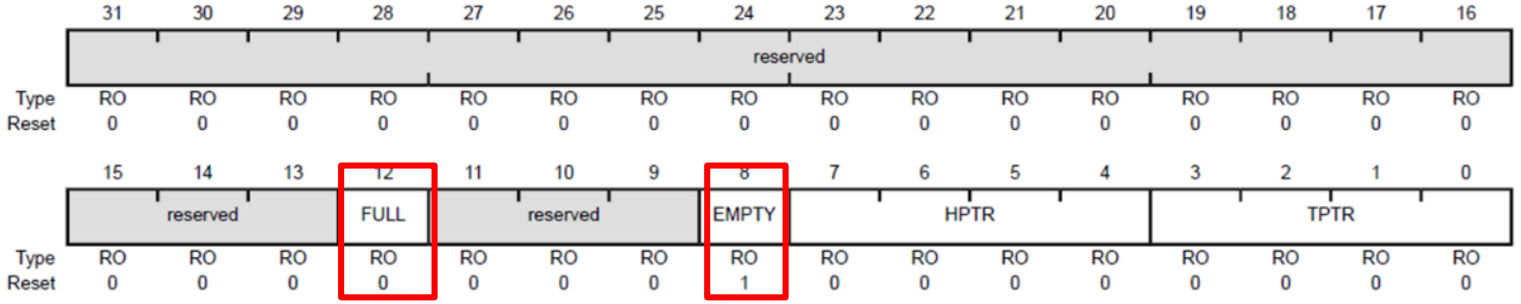
➢ TM4C1294的ADC模块的使用方法

– 8. 编写中断服务函数，读取数据

ADC模块通过ADC Sample Sequence Result FIFO 0/1/2/3读取采样序列SS0/1/2/3的转换结果，通过读该寄存器，把转换结果从FIFIO中读出来



ADC模块提供了FIFO状态寄存器 ADC Sample Sequence FIFO 0/1/2/3 Status (ADCSSFSTATn)查看结果FIFO的状态



EMPTY域表示FIFO为空，FULL域表示FIFO满

➢ TM4C1294的ADC模块的使用方法

– 8. 编写中断服务函数，读取数据

TivaWare提供了ADCSequenceDataGet函数，把FIFO中的转换结果读出来

```
int32_t
ADCSequenceDataGet(uint32_t ui32Base, uint32_t ui32SequenceNum,
                   uint32_t *pui32Buffer)
{
    uint32_t ui32Count;
    // Check the arguments.
    ASSERT((ui32Base == ADC0_BASE) || (ui32Base == ADC1_BASE));
    ASSERT(ui32SequenceNum < 4);
    // Get the offset of the sequence to be read.
    ui32Base += ADC_SEQ + (ADC_SEQ_STEP * ui32SequenceNum);
    // Read samples from the FIFO until it is empty.
    ui32Count = 0;
    while(!(HWREG(ui32Base + ADC_SSFSTAT) & ADC_SSFSTAT0_EMPTY) &&
          (ui32Count < 8))
    {
        // Read the FIFO and copy it to the destination.
        *pui32Buffer++ = HWREG(ui32Base + ADC_SSFIFO);
        // Increment the count of samples read.
        ui32Count++;
    }
    return(ui32Count);
}
```

```
#define ADC_SSFSTAT0_EMPTY 0x00000100  // FIFO Empty
```

该函数连续读取FIFO，把所有转换结果都取出来，直到FIFO为空为止。

TM4C1294的ADC模块的使用方法

- **8. 编写中断服务函数，读取数据**

TivaWare提供了ADCSequenceDataGet函数，把FIFO中的转换结果读出来

```
uint32_t ui32ACCValues[8];
ADCSequenceDataGet(ADC0_BASE, 0, ui32ACCValues);
```

➢ 示例

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0);

ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PLL |
ADC_CLOCK_RATE_FULL, 24);
ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH3);
ADCSequenceStepConfigure(ADC0_BASE, 0, 5,
ADC_CTL_CH3|ADC_CTL_IE|ADC_CTL_END);
IntEnable(INT_ADC0SS0);
ADCIntEnable(ADC0_BASE,0);
ADCSequenceEnable(ADC0_BASE, 0);
ADCIntClear(ADC0_BASE, 0);

ADCProcessorTrigger(ADC0_BASE, 0);
```

➢ 读取ADC数据

```
uint32_t ui32ACCValues[8];

void adc0_ss0_isr(void){
ADCIntClear(ADC0_BASE, 0);
ADCSequenceDataGet(ADC0_BASE, 0, ui32ACCValues);
}
```

谢谢！