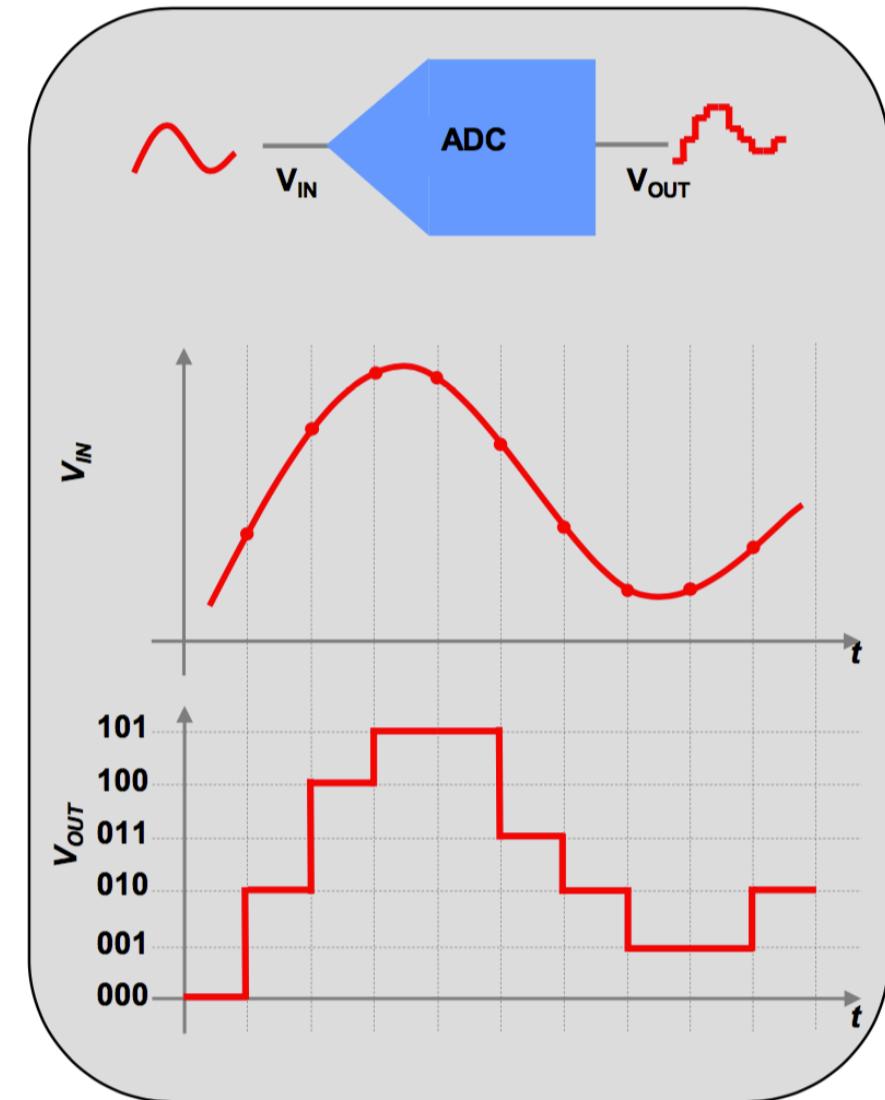
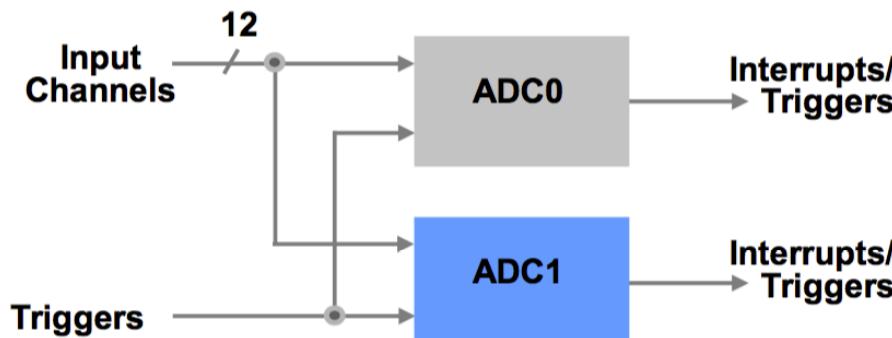




**Il seguente materiale e' stato estratto da:**

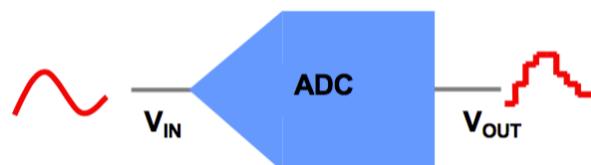
[http://processors.wiki.ti.com/index.php/Getting\\_Started\\_with\\_the\\_TIVA%E2%84%A2\\_C\\_Series\\_TM4C123G\\_LaunchPad#Workshop\\_Material](http://processors.wiki.ti.com/index.php/Getting_Started_with_the_TIVA%E2%84%A2_C_Series_TM4C123G_LaunchPad#Workshop_Material)

- ◆ Tiva TM4C MCUs feature two ADC modules (ADC0 and ADC1) that can be used to convert continuous analog voltages to discrete digital values
- ◆ Each ADC module has 12-bit resolution
- ◆ Each ADC module operates independently and can:
  - Execute different sample sequences
  - Sample any of the shared analog input channels
  - Generate interrupts & triggers



# TM4C123GH6PM ADC Features

- ◆ Two 12-bit 1MSPS ADCs
- ◆ 12 shared analog input channels
- ◆ Single ended & differential input configurations
- ◆ On-chip temperature sensor
- ◆ Maximum sample rate of one million samples/second (1MSPS).
- ◆ Fixed references (VDDA/GNDA) due to pin-count limitations
- ◆ 4 programmable sample conversion sequencers per ADC
- ◆ Separate analog power & ground pins
- ◆ Flexible trigger control
  - Controller/ software
  - Timers
  - Analog comparators
  - GPIO
- ◆ 2x to 64x hardware averaging
- ◆ 8 Digital comparators / per ADC
- ◆ 2 Analog comparators
- ◆ Optional phase shift in sample time, between ADC modules ... programmable from 22.5 ° to 337.5°

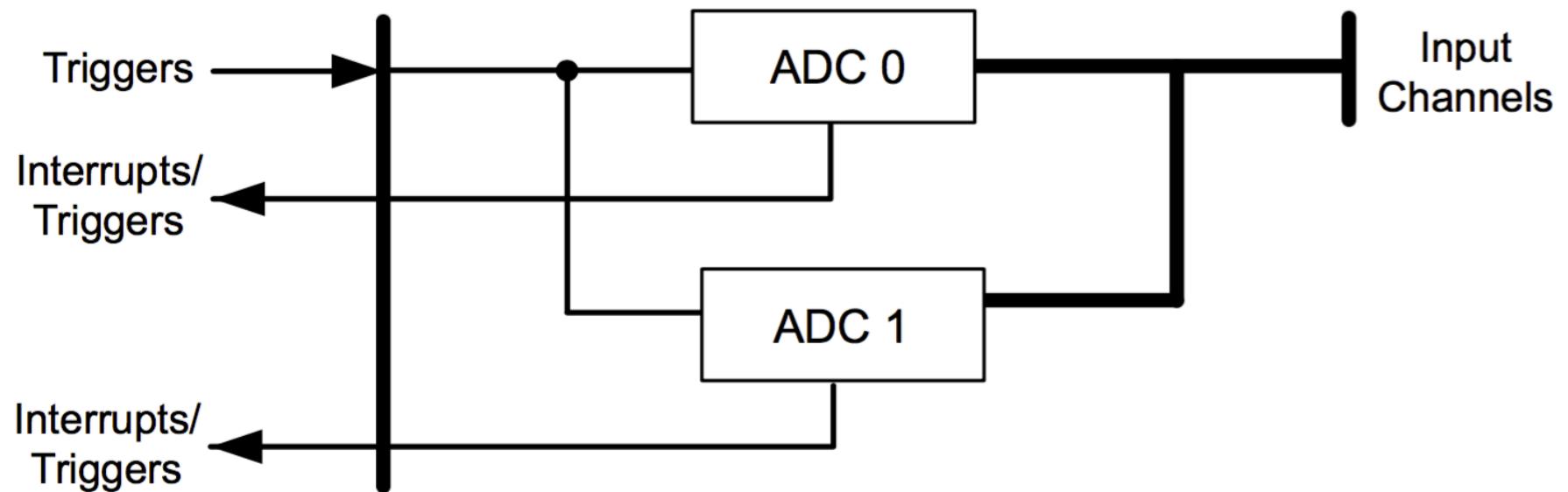




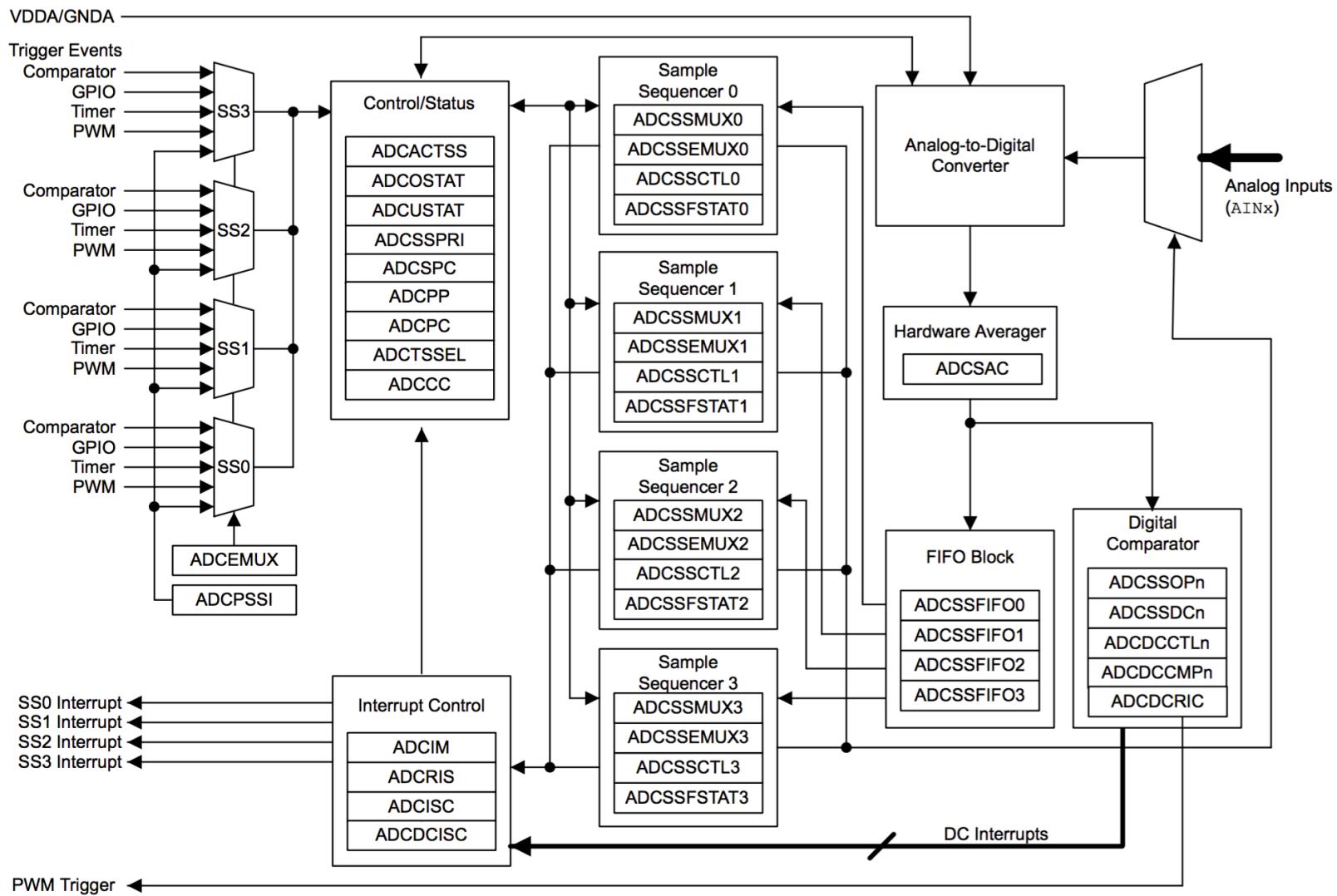
- ◆ Tiva TM4C ADC's collect and sample data using programmable sequencers.
- ◆ Each sample sequence is a fully programmable series of consecutive (back-to-back) samples that allows the ADC module to collect data from multiple input sources without having to be re-configured.
- ◆ Each ADC module has 4 sample sequencers that control sampling and data capture.
- ◆ All sample sequencers are identical except for the number of samples they can capture and the depth of their FIFO.
- ◆ To configure a sample sequencer, the following information is required:
  - Input source for each sample
  - Mode (single-ended, or differential) for each sample
  - Interrupt generation on sample completion for each sample
  - Indicator for the last sample in the sequence
- ◆ Each sample sequencer can transfer data independently through a dedicated µDMA channel.

Sequencer	Number of Samples	Depth of FIFO
SS 3	1	1
SS 2	4	4
SS 1	4	4
SS 0	8	8

**Figure 13-1. Implementation of Two ADC Blocks**



**Figure 13-2. ADC Module Block Diagram**





Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type <sup>a</sup>	Description
AIN0	6	PE3		Analog	Analog-to-digital converter input 0.
AIN1	7	PE2		Analog	Analog-to-digital converter input 1.
AIN2	8	PE1		Analog	Analog-to-digital converter input 2.
AIN3	9	PE0		Analog	Analog-to-digital converter input 3.
AIN4	64	PD3		Analog	Analog-to-digital converter input 4.
AIN5	63	PD2		Analog	Analog-to-digital converter input 5.
AIN6	62	PD1		Analog	Analog-to-digital converter input 6.
AIN7	61	PD0		Analog	Analog-to-digital converter input 7.
AIN8	60	PE5		Analog	Analog-to-digital converter input 8.
AIN9	59	PE4		Analog	Analog-to-digital converter input 9.
AIN10	58	PB4		Analog	Analog-to-digital converter input 10.
AIN11	57	PB5		Analog	Analog-to-digital converter input 11.

**Figure 13-5. Skewed Sampling**

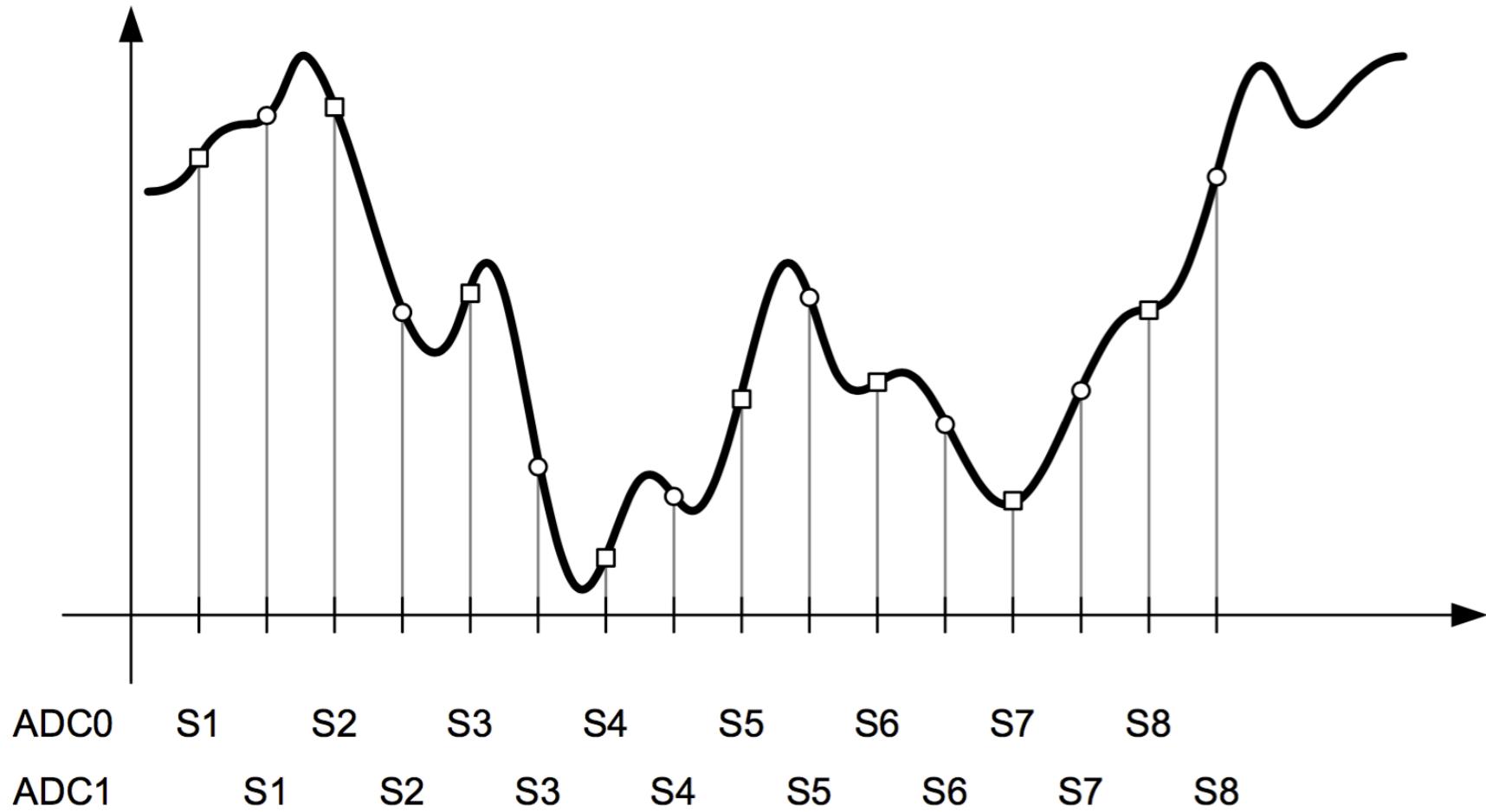
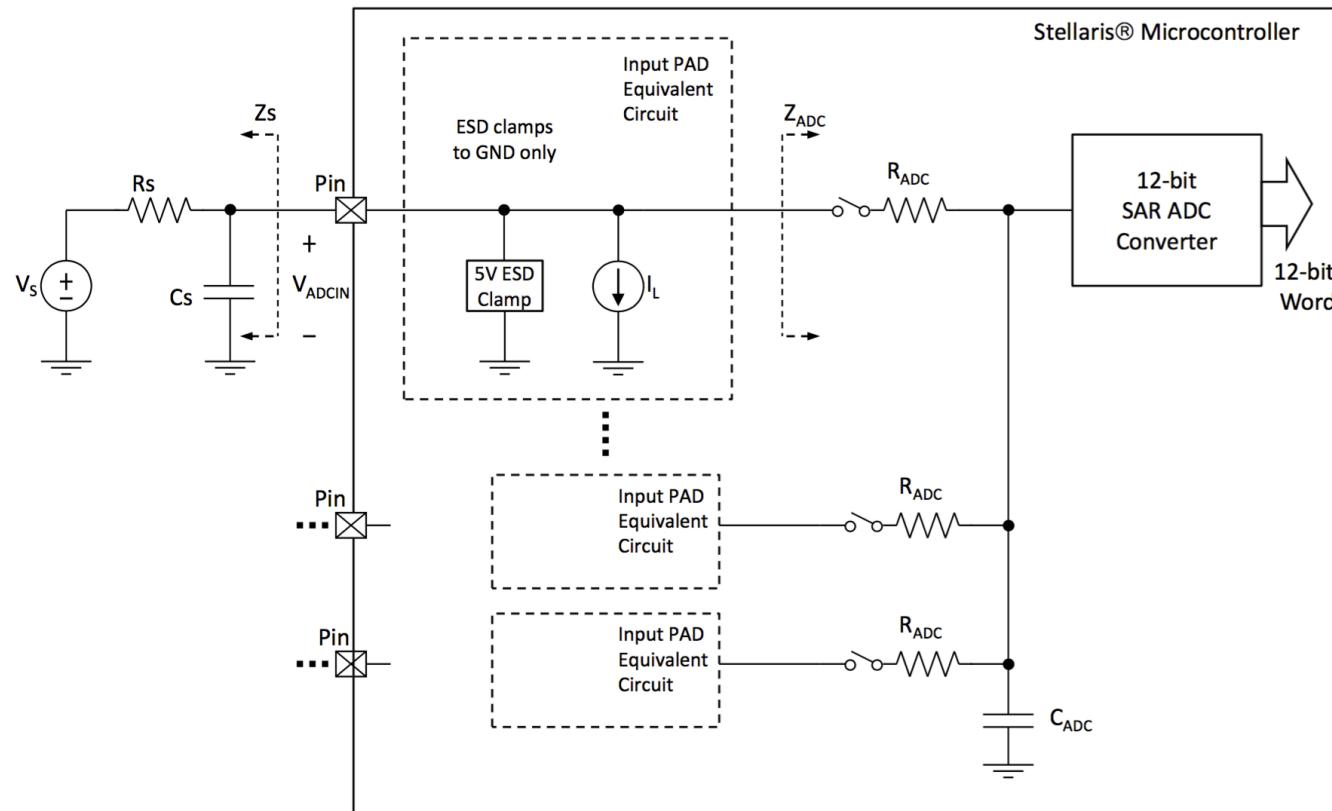
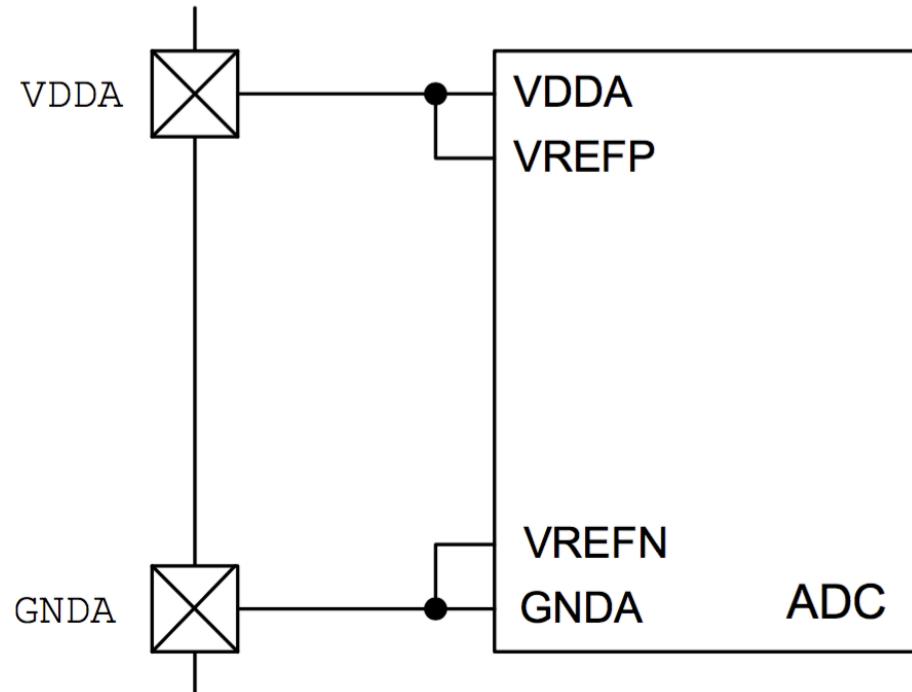


Figure 13-7. ADC Input Equivalency Diagram



$I_L$	ADC input leakage current <sup>h</sup>	-	-	2.0	$\mu\text{A}$
$R_{\text{ADC}}$	ADC equivalent input resistance <sup>h</sup>	-	-	2.5	$\text{k}\Omega$
$C_{\text{ADC}}$	ADC equivalent input capacitance <sup>h</sup>	-	-	10	$\text{pF}$

**Figure 13-8. ADC Voltage Reference**

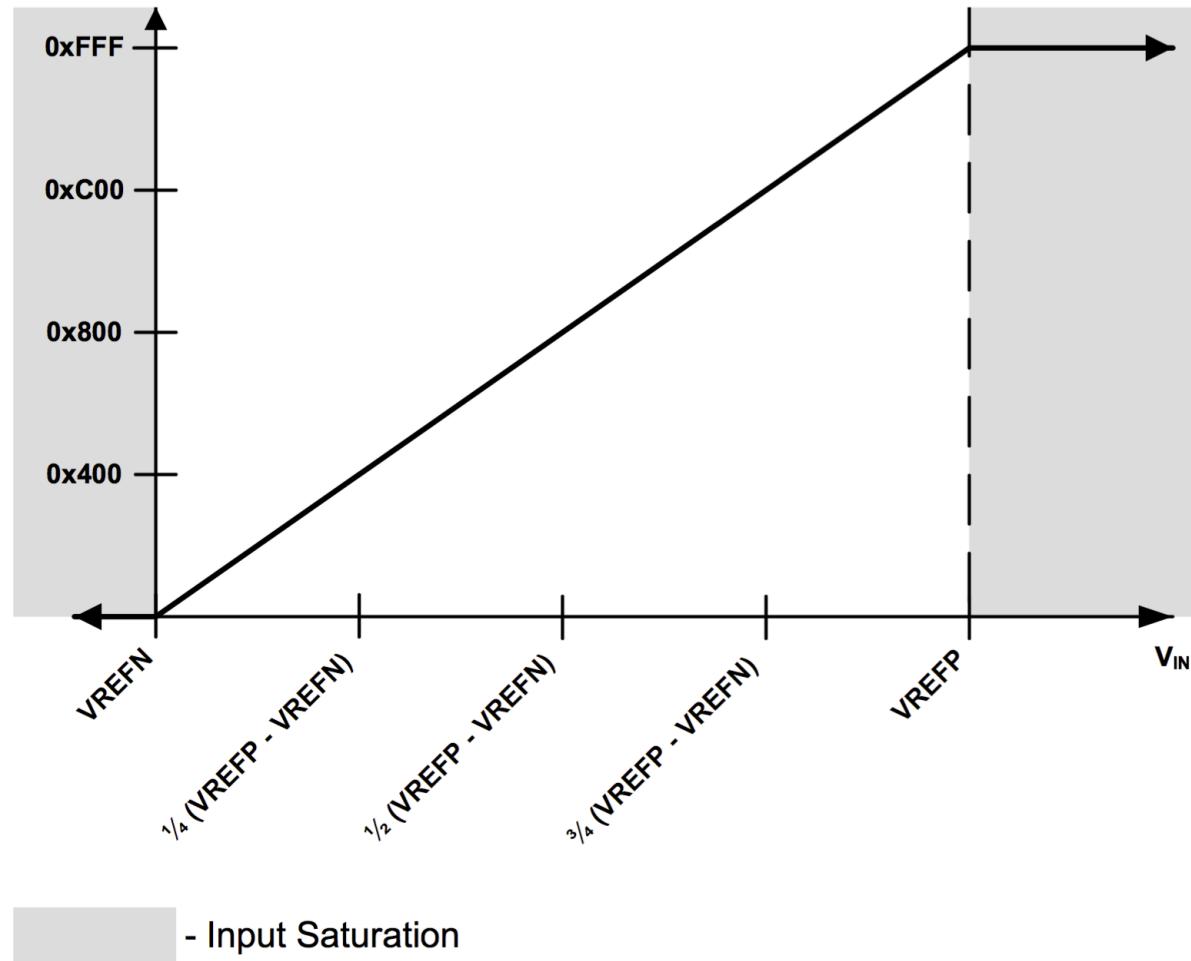


The range of this conversion value is from 0x000 to 0xFFFF. In single-ended-input mode, the 0x000 value corresponds to the voltage level on VREFN; the 0xFFFF value corresponds to the voltage level on VREFP. This configuration results in a resolution that can be calculated using the following equation:

$$\text{mV per ADC code} = (\text{VREFP} - \text{VREFN}) / 4096$$



**Figure 13-9. ADC Conversion Result**



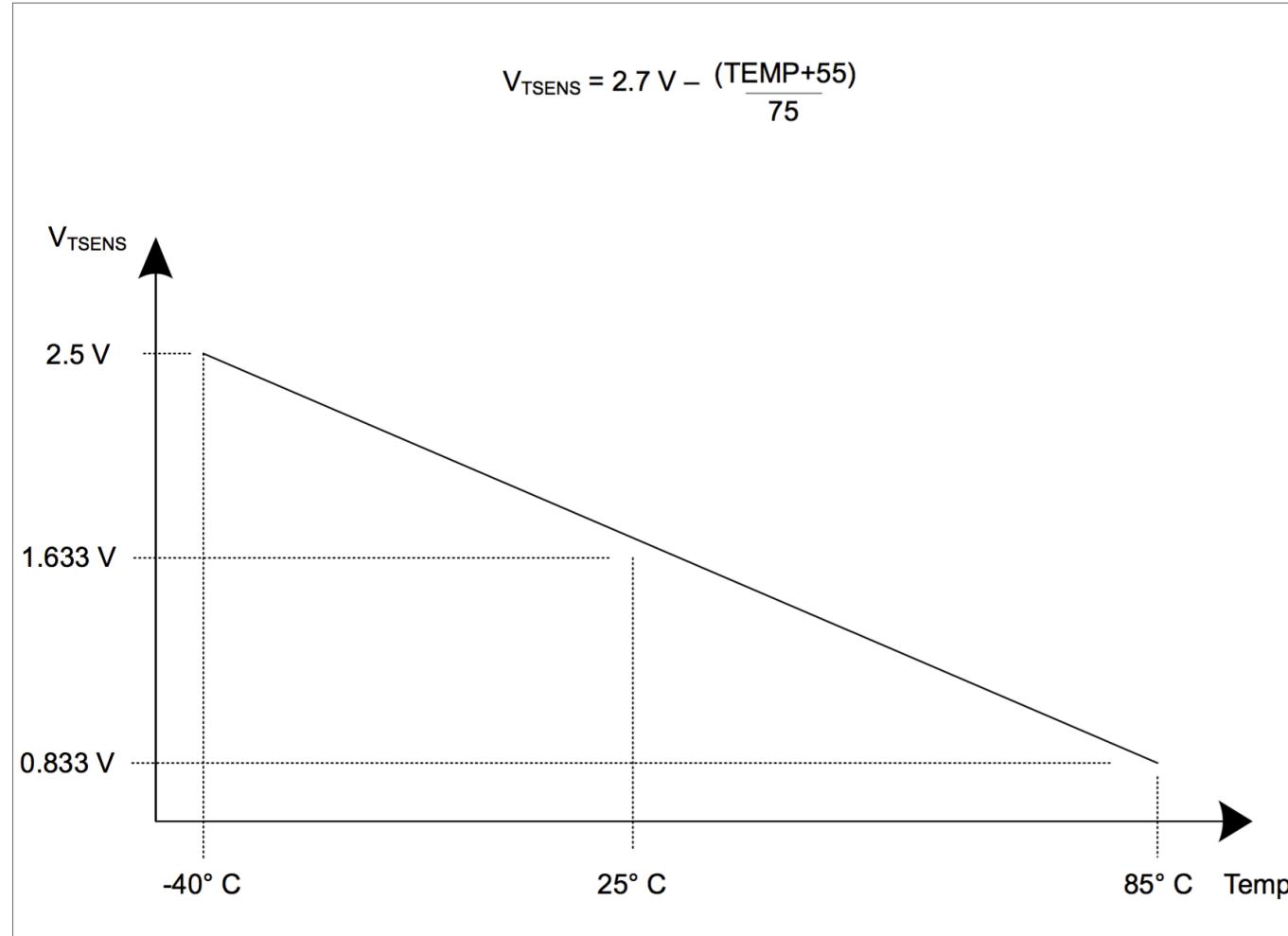


The internal temperature sensor converts a temperature measurement into a voltage. This voltage value,  $V_{TSENS}$ , is given by the following equation (where TEMP is the temperature in  $^{\circ}\text{C}$ ):

$$V_{TSENS} = 2.7 - ((\text{TEMP} + 55) / 75)$$

This relation is shown in Figure 13-11 on page 772.

**Figure 13-11. Internal Temperature Sensor Characteristic**



The temperature sensor reading can be sampled in a sample sequence by setting the **TSn** bit in the **ADCSSCTLn** register. The temperature reading from the temperature sensor can also be given as a function of the ADC value. The following formula calculates temperature (TEMP in °C) based on the ADC reading (ADC<sub>CODE</sub>, given as an unsigned decimal number from 0 to 4095) and the maximum ADC voltage range (VREFP - VREFN):

$$\text{TEMP} = 147.5 - ((75 * (\text{VREFP} - \text{VREFN}) * \text{ADC}_{\text{CODE}}) / 4096)$$



# the code



```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
```



**Usiamo il sequencer1 (profondita' 4) e quindi creiamo un vettore lungo 4**

```
uint32_t ui32ADC0Value[4];
```



**Usiamo il sequencer 1 (profondita' 4) e quindi creiamo un vettore lungo 4**

```
uint32_t ui32ADC0Value[4];
```

**Creiamo le variabili per salvare il valore della temperatura**

```
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
```



**Usiamo il sequencer 1 (profondita' 4) e quindi creiamo un vettore lungo 4**

```
uint32_t ui32ADC0Value[4];
```

**Creiamo le variabili per salvare il valore della temperatura**

```
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;
```

**Settiamo il clock**

```
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
```



Usiamo il sequencer 1 (profondita' 4) e quindi creiamo un vettore lungo 4

```
uint32_t ui32ADC0Value[4];
```

Creiamo le variabili per salvare il valore della temperatura

```
volatile uint32_t ui32TempAvg;  
volatile uint32_t ui32TempValueC;  
volatile uint32_t ui32TempValueF;
```

Settiamo il clock

```
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC  
_MAIN|SYSCTL_XTAL_16MHZ);
```

..ed abilitiamo l'ADC0

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```



**Configuriamo l'ADC:**

**Usiamo ADC0, il primo sequencer, “triggerato” processore, massima priorita’**

```
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
```



**Configuriamo l'ADC:**

**Usiamo ADC0, il primo sequencer, “triggerato” processore, massima priorita’**

```
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
```

**Configuriamo i primi 3 step (leggo i dati dal sensore di temperatura)**

```
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
```



**Configuriamo l'ADC:**

**Usiamo ADC0, il primo sequencer, “triggerato” processore, massima priorita’**

```
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
```

**Configuriamo i primi 3 step (leggo i dati dal sensore di temperatura)**

```
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
```

**Configuriamo l’ultimo step (aspetto che finisce la lettura, tramite interruzione)**

```
ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
```



Infine abilitiamo il sequencer 1

```
ADCSequenceEnable(ADC0_BASE, 1);
```

Dopo l'inizializzazione, passiamo al ciclo while

Visto che la fine del processo di conversione ADC e' indicato con un flag dello stato dell'ADC interrupt, e' sempre bene fare un clear del flag prima che il codice cominci.

```
while (1)
{
    ADCIntClear(ADC0_BASE, 1);

    . . . . .

}
```



**Adesso specifichiamo che il trigger avviene via software**

**ADCProcessorTrigger(ADC0\_BASE, 1);**



**Adesso specifichiamo che il trigger avviene via software**

```
ADCProcessorTrigger(ADC0_BASE, 1);
```

**Aspettiamo che la conversione sia finita**

```
while(!ADCIntStatus(ADC0_BASE, 1, false))  
{  
}
```



**Adesso specifichiamo che il trigger avviene via software**

```
ADCProcessorTrigger(ADC0_BASE, 1);
```

**Aspettiamo che la conversione sia finita**

```
while(!ADCIntStatus(ADC0_BASE, 1, false))  
{  
}
```

**A questo punto leggiamo i dati**

```
ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
```



**Adesso specifichiamo che il trigger avviene via software**

```
ADCProcessorTrigger(ADC0_BASE, 1);
```

**Aspettiamo che la conversione sia finita**

```
while(!ADCIntStatus(ADC0_BASE, 1, false))
{
}
```

**A questo punto leggiamo i dati**

```
ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
```

**E facciamo una media (il 2 e' per l'arrotondamento)**

```
ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
```



**Calcolo la temperatura, ricordando che:**

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

**e che VREFP-VREFN=3.3V**

`ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;`

**e in Fahrenheit**

`ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;`

**Fine ciclo while.**



```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
int main(void)
{
    uint32_t ui32ADC0Value[4]; volatile uint32_t ui32TempAvg; volatile uint32_t
    ui32TempValueC; volatile uint32_t ui32TempValueF;
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);
    while(1)
    {
        ADCIntClear(ADC0_BASE, 1); ADCProcessorTrigger(ADC0_BASE, 1);
        while(!ADCIntStatus(ADC0_BASE, 1, false)) {
            }
        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
        ui32ADC0Value[3] + 2)/4; ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) /
        4096)/10;
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
    } }
```