Tiva C ADC

Refrereces:

James Cockrell and Justin Loveless

Hossam Elkady Youtube Arabic Course

Practical Microcontroller Engineering with ARM Tech by Ying Bai

Analog-Digital Converter (ADC): Agenda

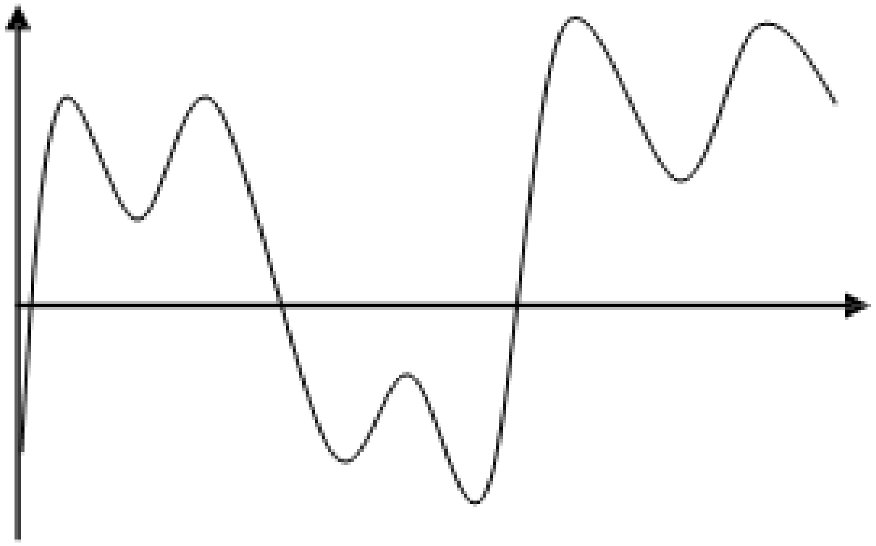
- How it works?
- Successive Approximation
- Register level TIVA ADC
- TIVAWare ADC driver library
- Examples
- Assignment

- **ADC:**
  - **Analog Signal:**

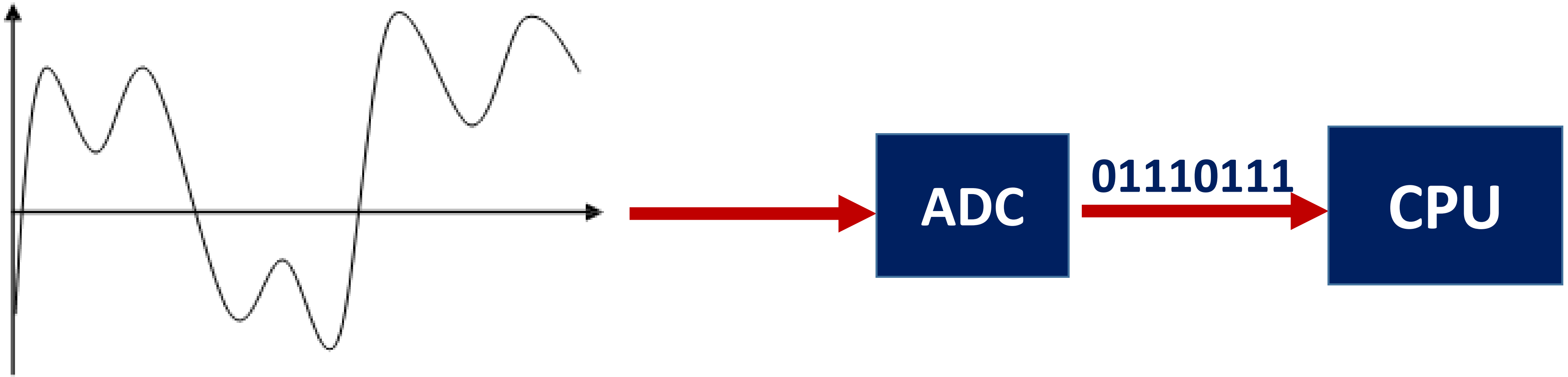
Temperature **Humidity**

Pressure **Light Intensity**

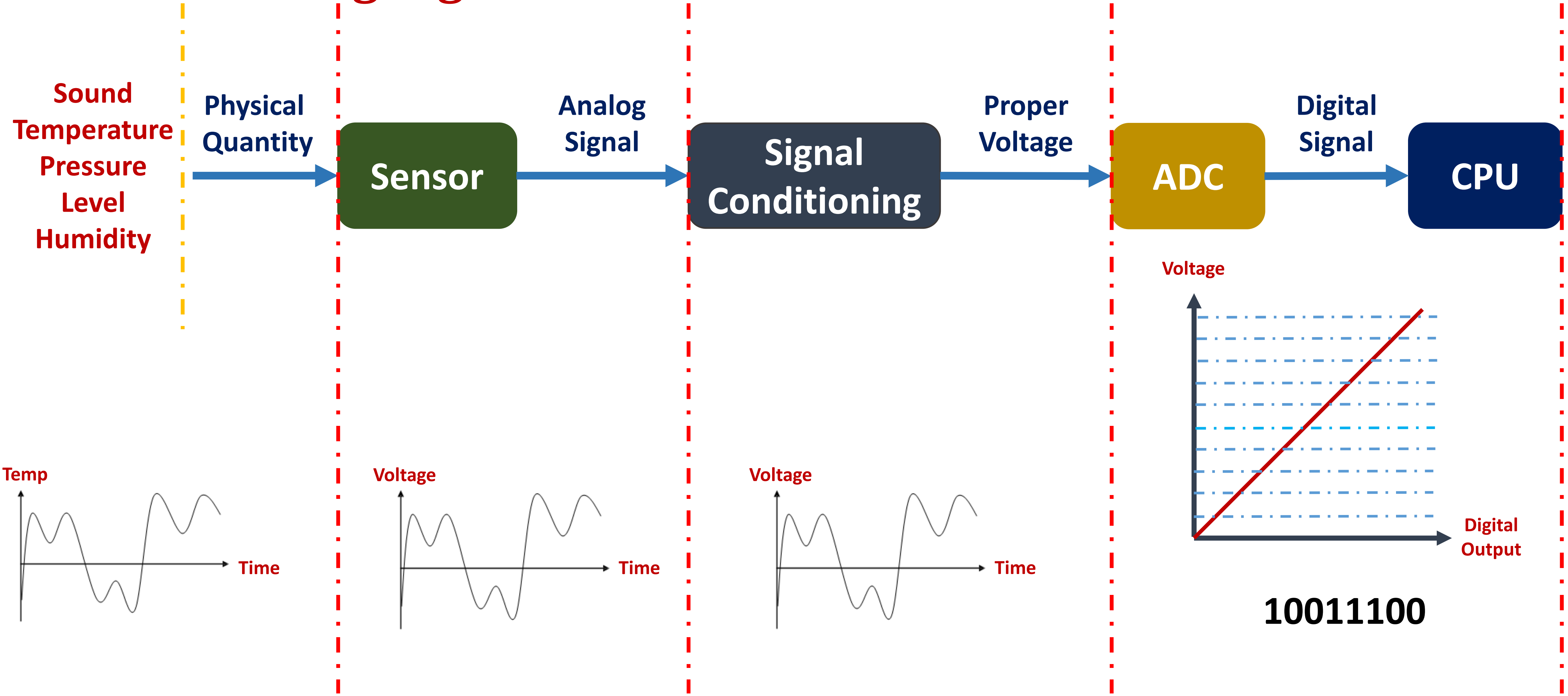
Level **Sound**



- **ADC:**
  - **Analog Signal:**



- **ADC:**
  - **Analog Signal:**



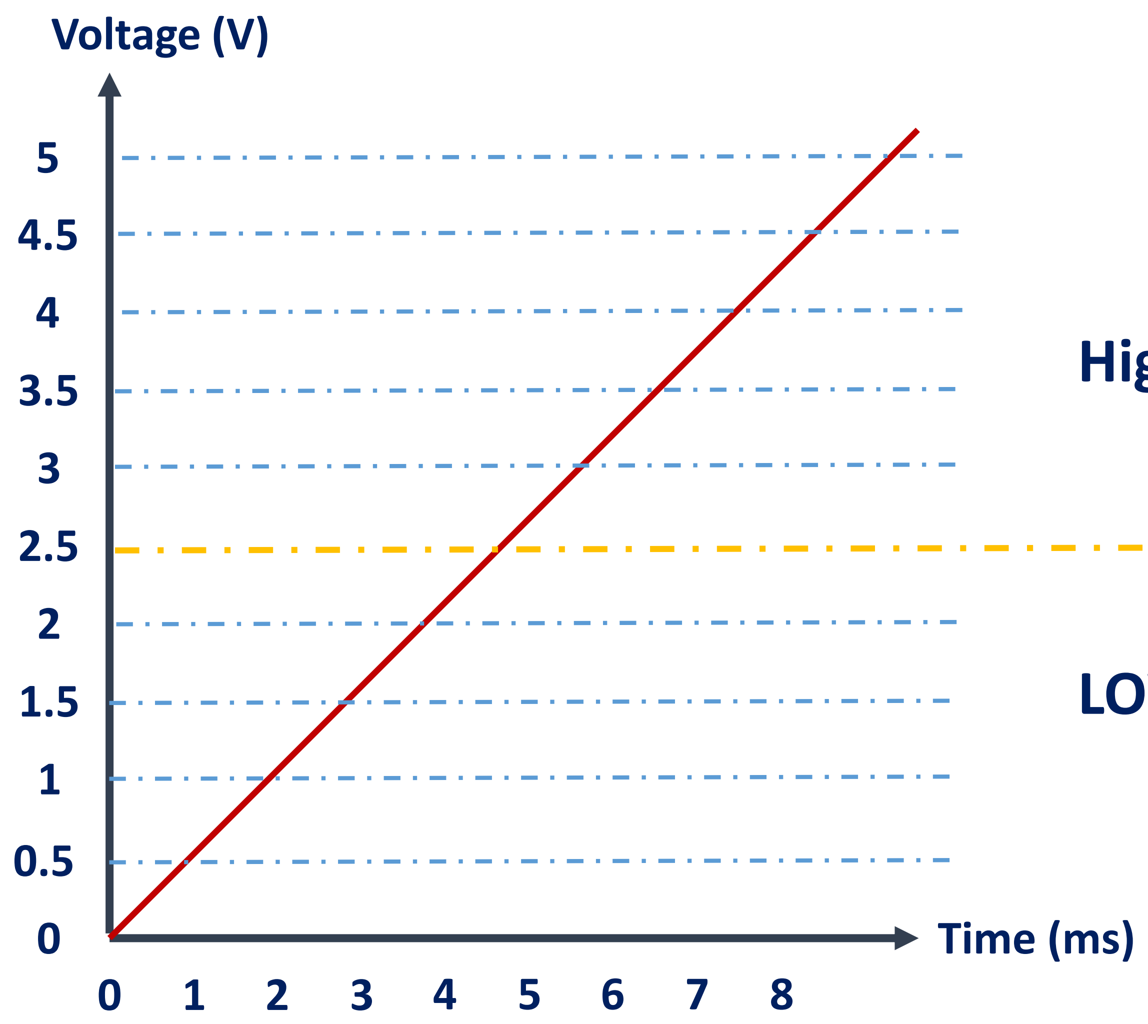
- **ADC:**
  - **Resolution:**

Step Size = $\frac{Vref}{2} = \frac{5\,v}{2} = 2.5$

$2^1 = 2$

ADC

1 Bit



High – Logic 1

LOW – Logic 0

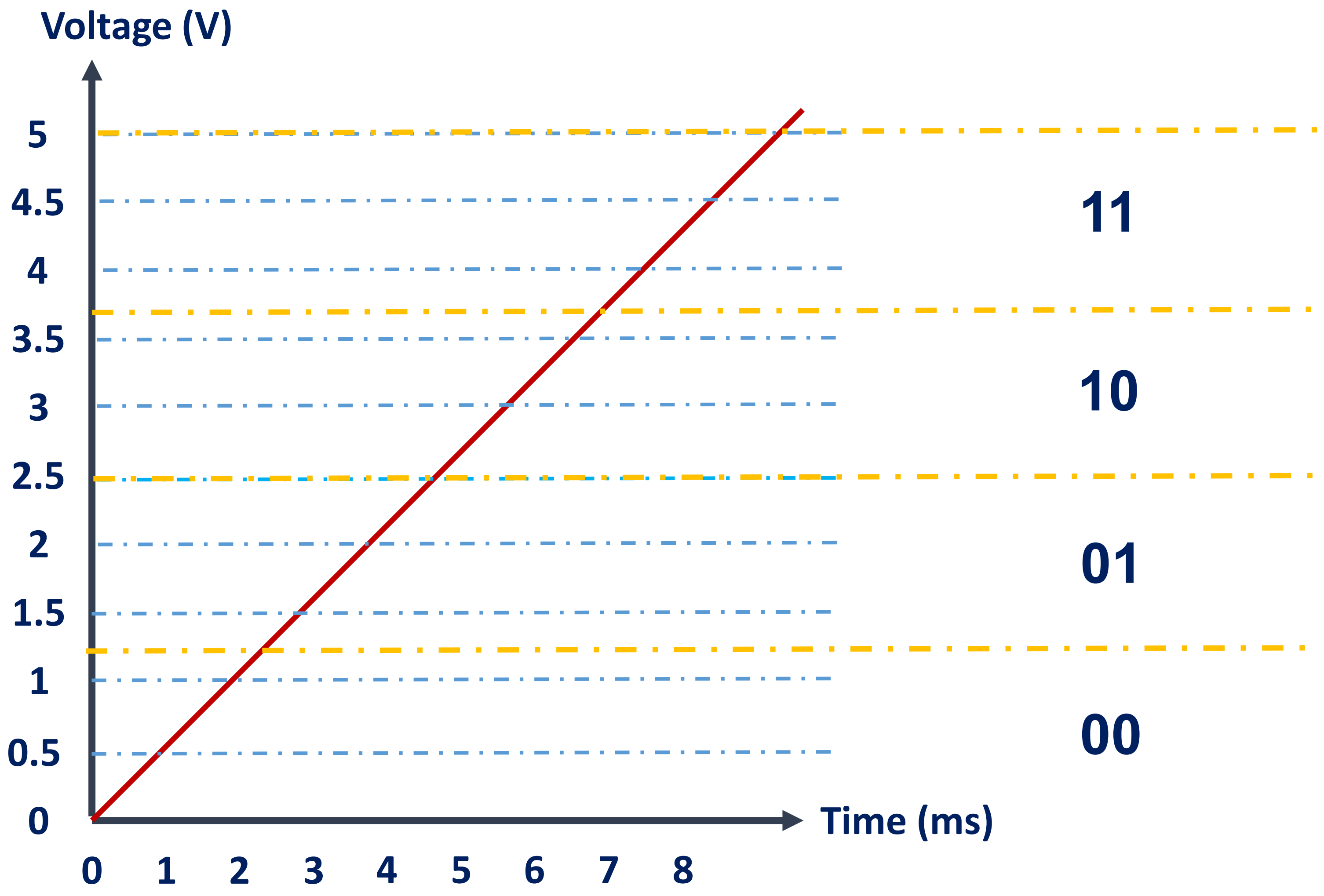
- **ADC:**
  - **Resolution:**

$$\frac{5\,v}{4} = 1.25v$$

$2^2 = 4$

ADC

2 Bit



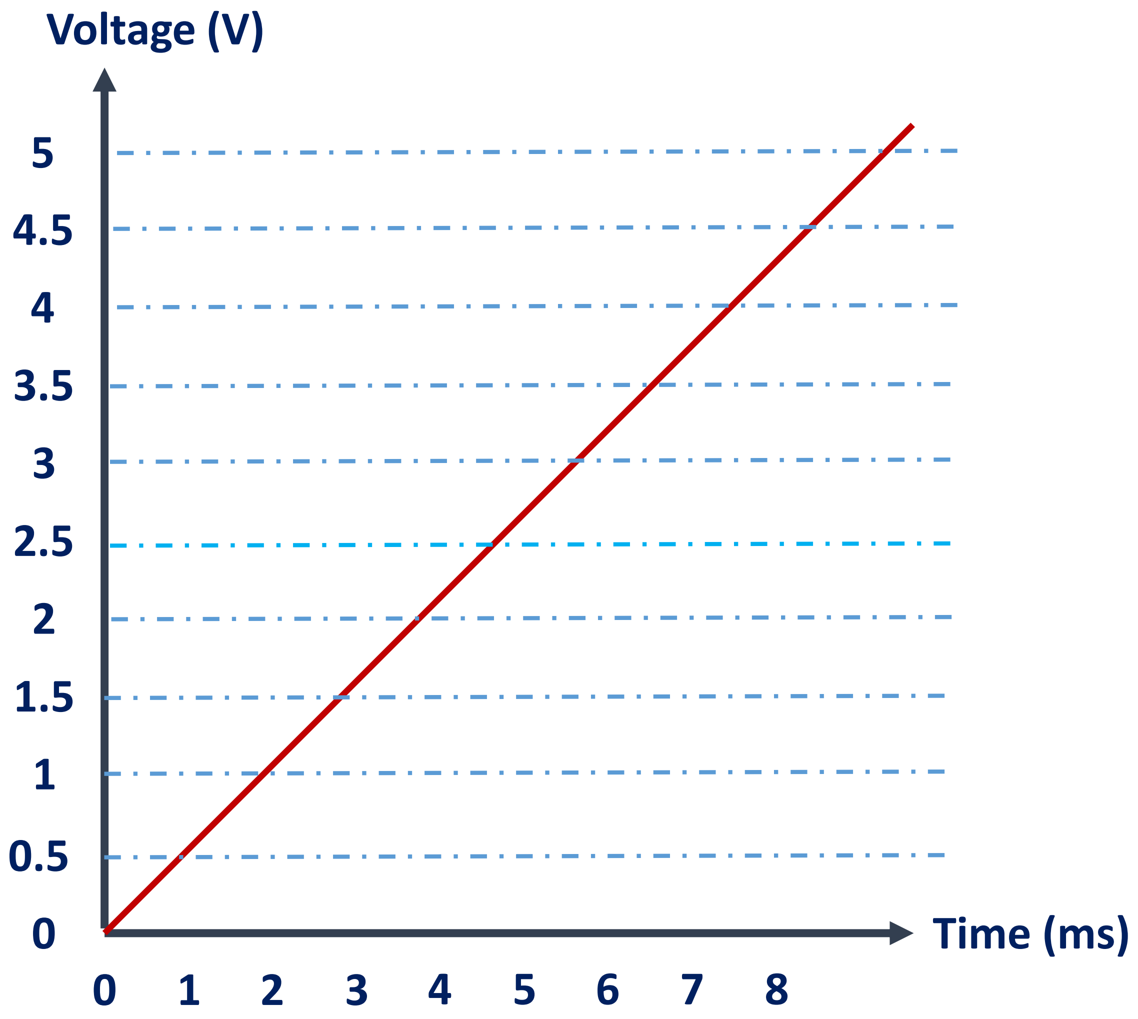
- **ADC:**
  - **Resolution:**

$$\frac{5\,v}{8} = 0.625$$

$2^3 = 8$

ADC

3 Bit



111
110
101
100
011
010
001
000

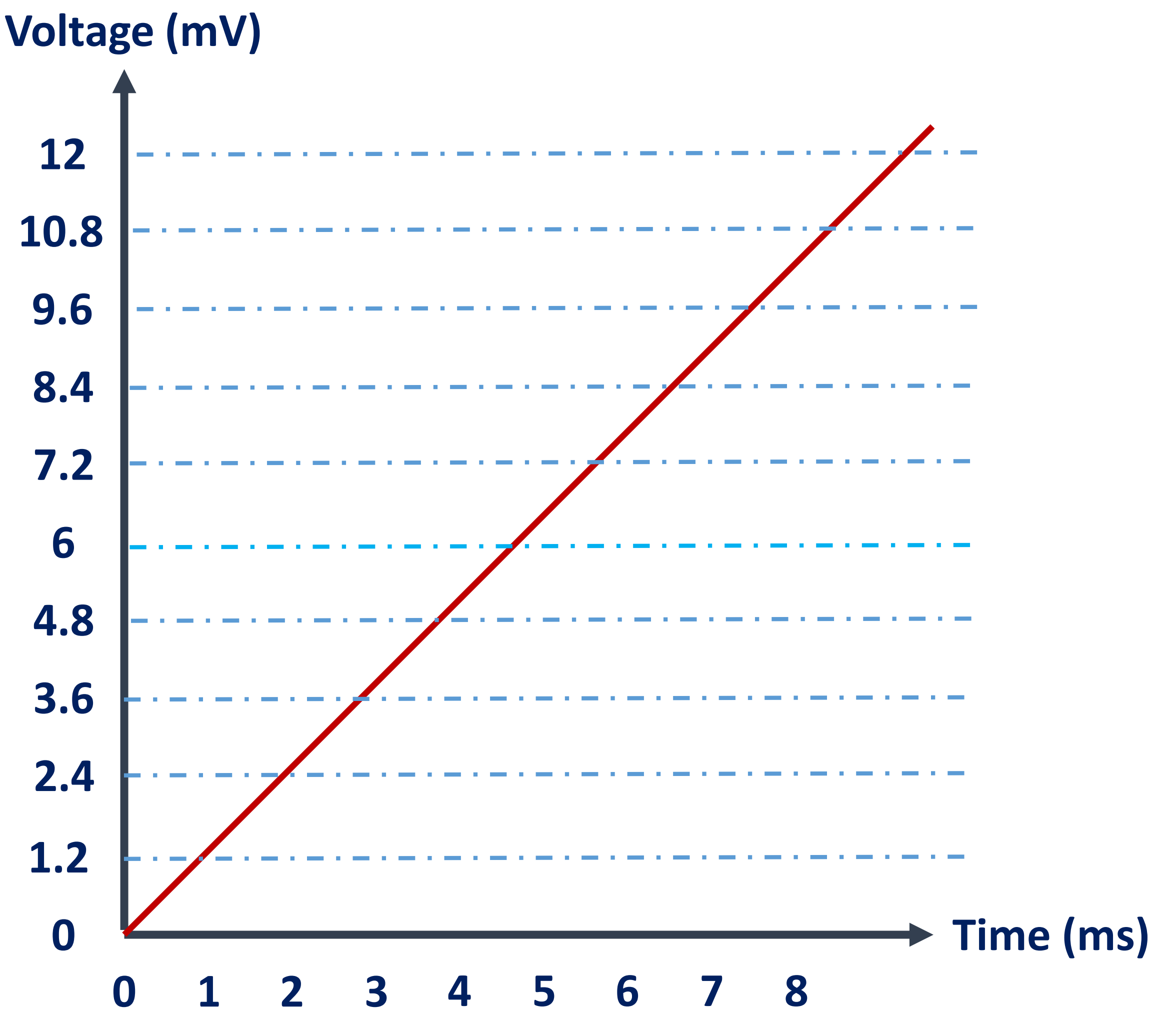
- **ADC:**
  - **Resolution:**

Step Size = $\frac{5\,v}{4096} = 1.2\,mv$

2^{12} = 4096

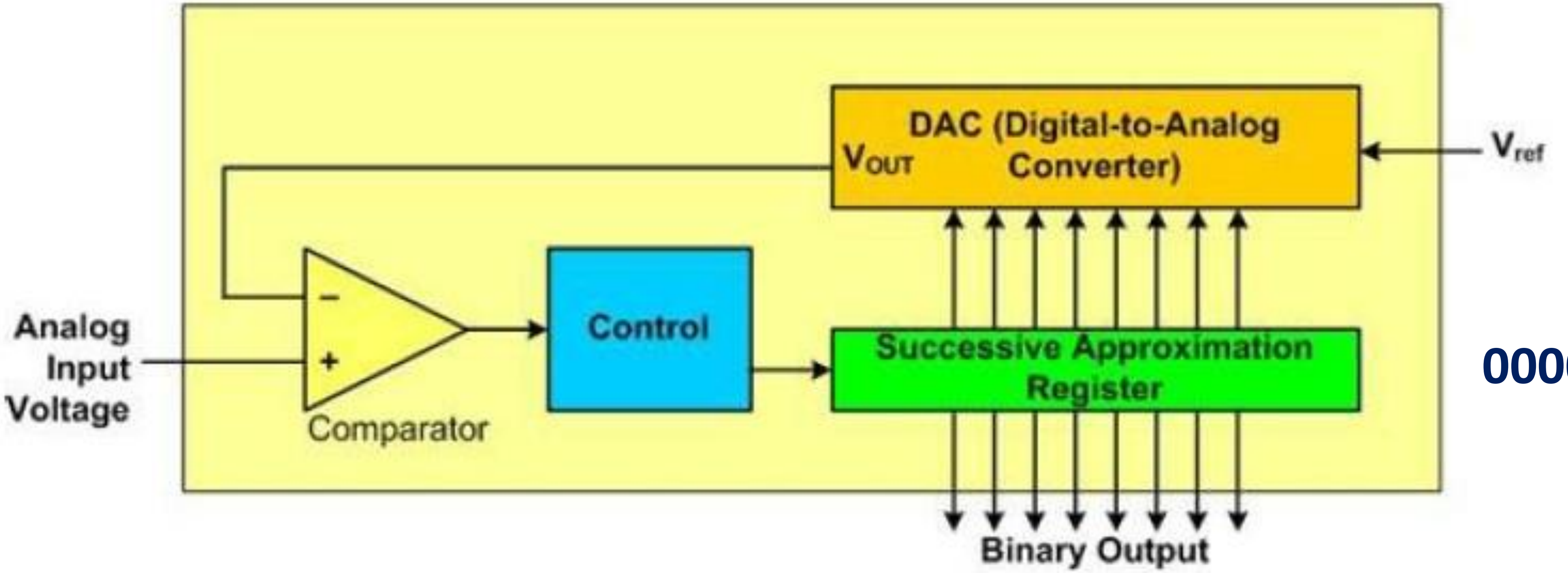
ADC

12 Bit



- **ADC:**
  - **Successive Approximation ADC:**

Assume
Step Size
10 mV



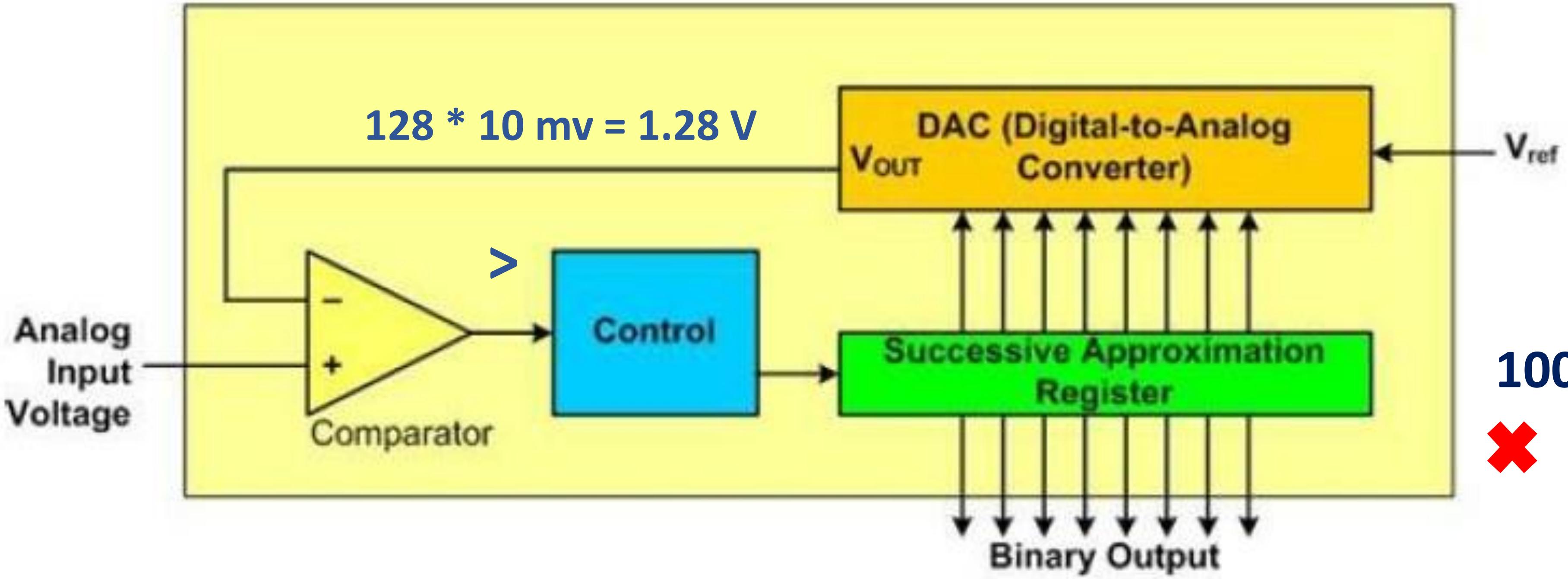
0

0000 0000

- **ADC:**
  - **Successive Approximation ADC:**

Assume
Step Size
10 mV

1 V

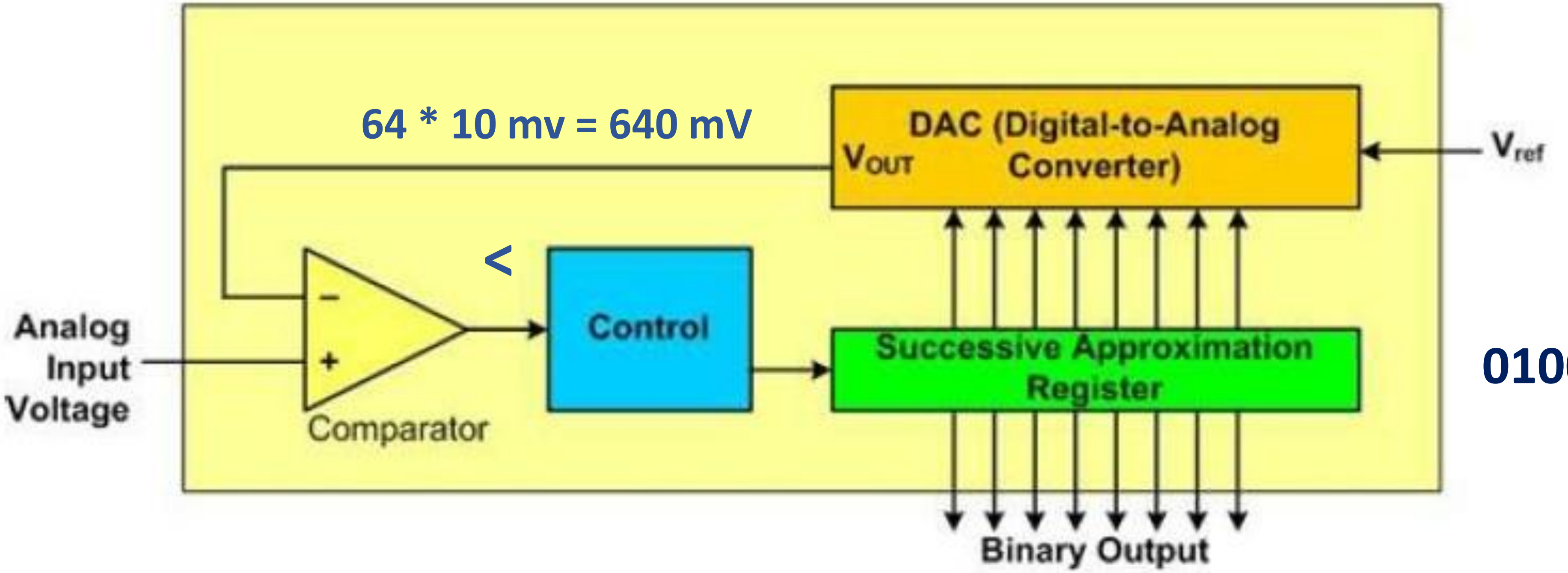


128

1000 0000

- **ADC:**
  - **Successive Approximation ADC:**

Assume
Step Size
10 mV



1 V

64

0100 0000

• ADC:

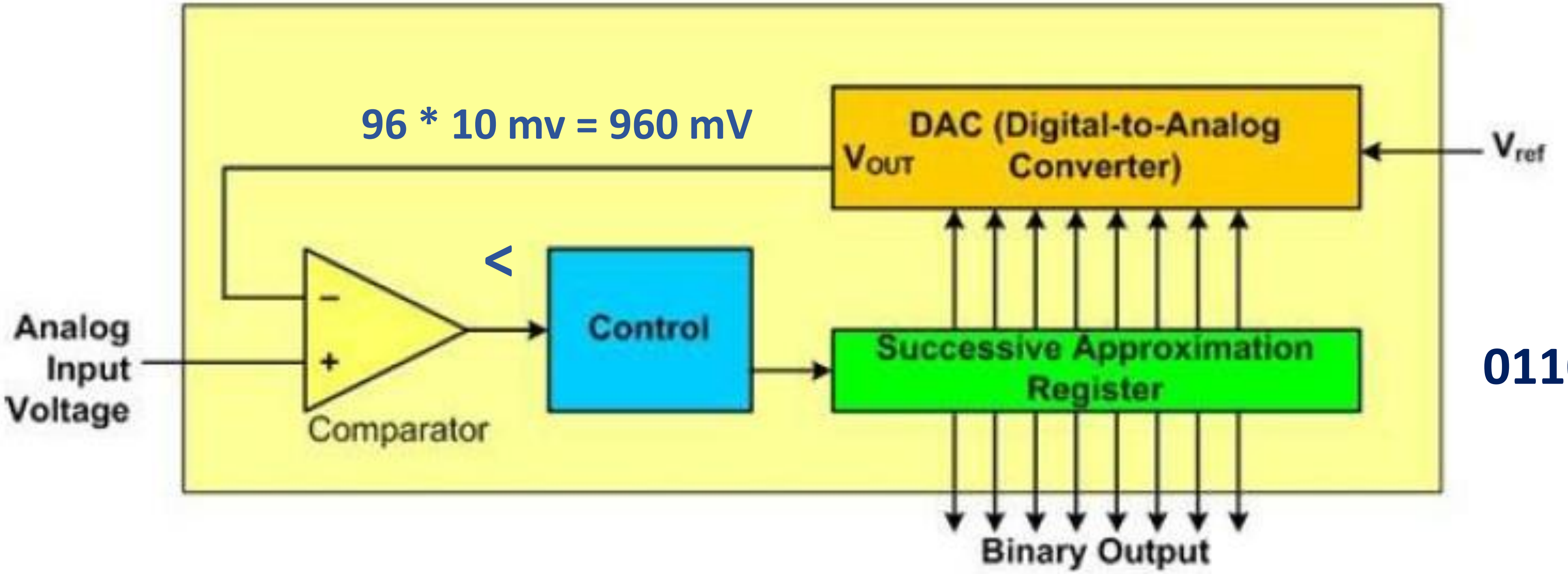
• Successive Approximation ADC:

Assume
Step Size
10 mV

96 * 10 mv = 960 mV

<

1 V

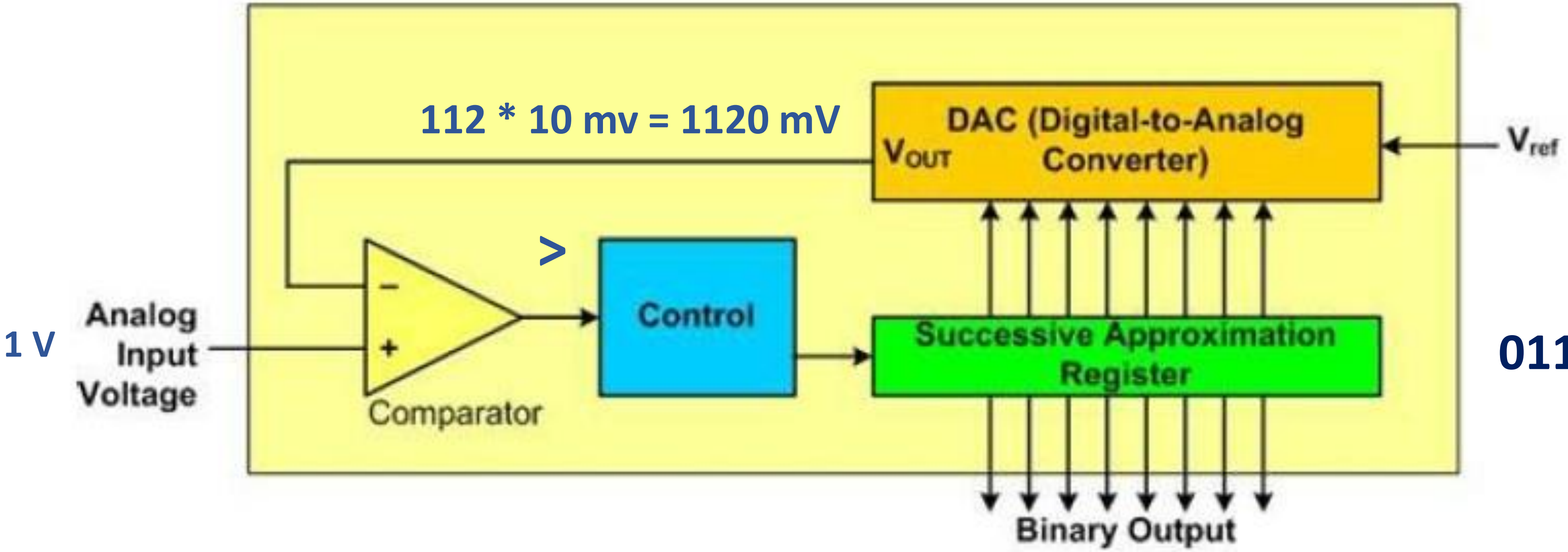


96

0110 0000

- **ADC:**
  - **Successive Approximation ADC:**

Assume
Step Size
10 mV



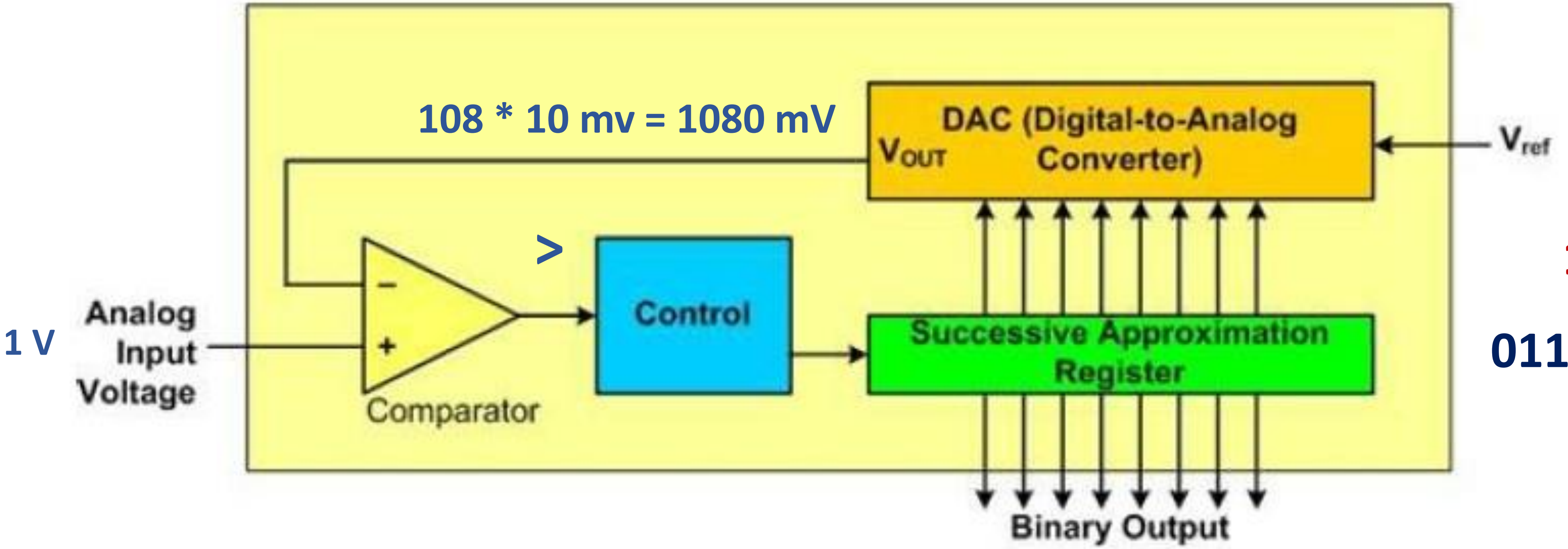
112

0111 0000

✖

- # ADC:
  - ## Successive Approximation ADC:

Assume
Step Size
10 mV

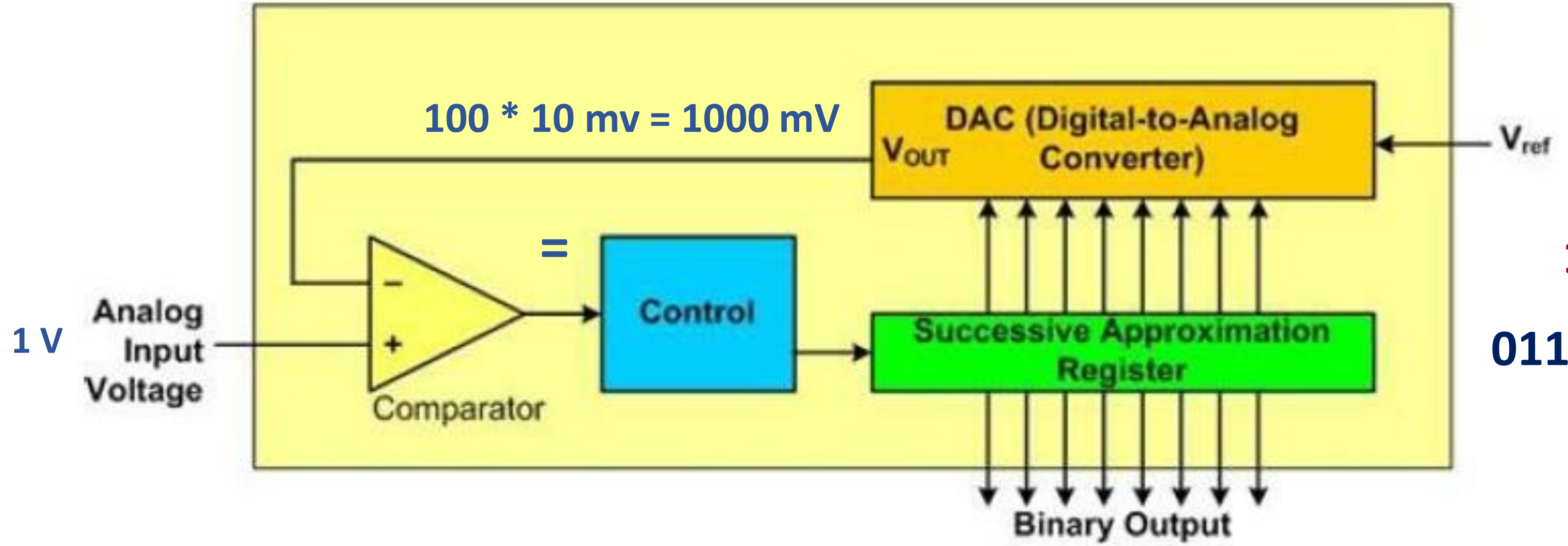


108

0110 1000

- **ADC:**
  - **Successive Approximation ADC:**

Assume
Step Size
10 mV

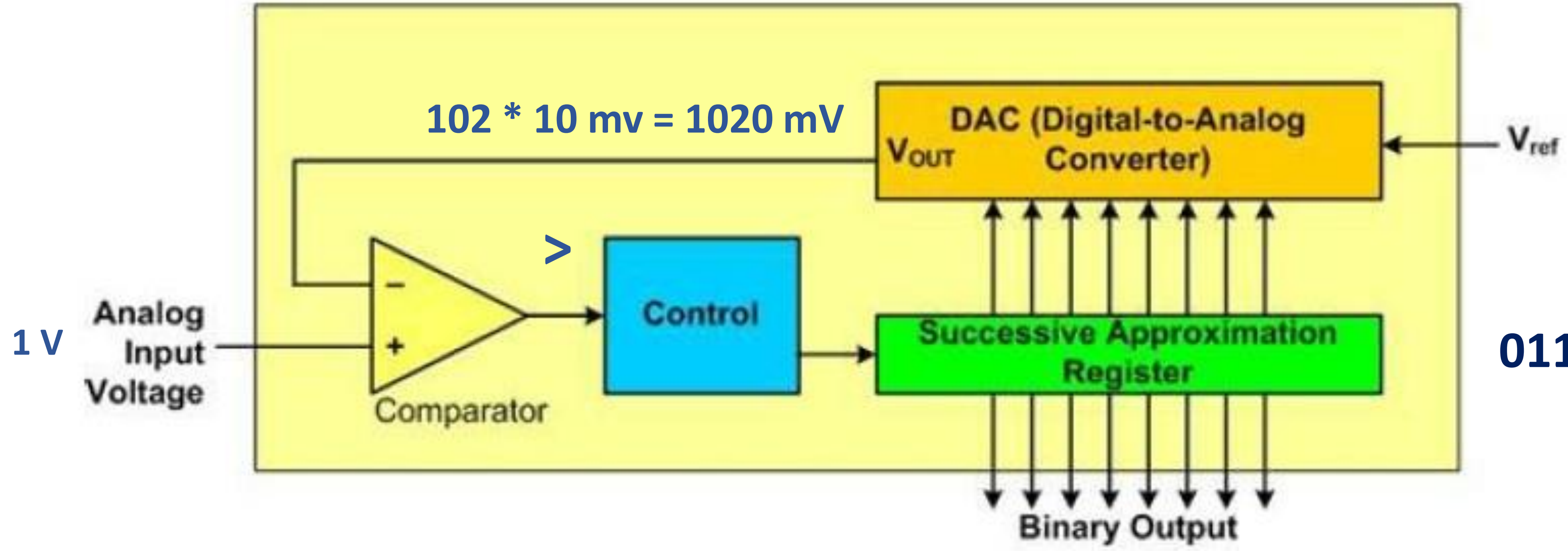


100

0110 0100

- # ADC:
  - ## Successive Approximation ADC:

Assume
Step Size
10 mV



102

0110 0110

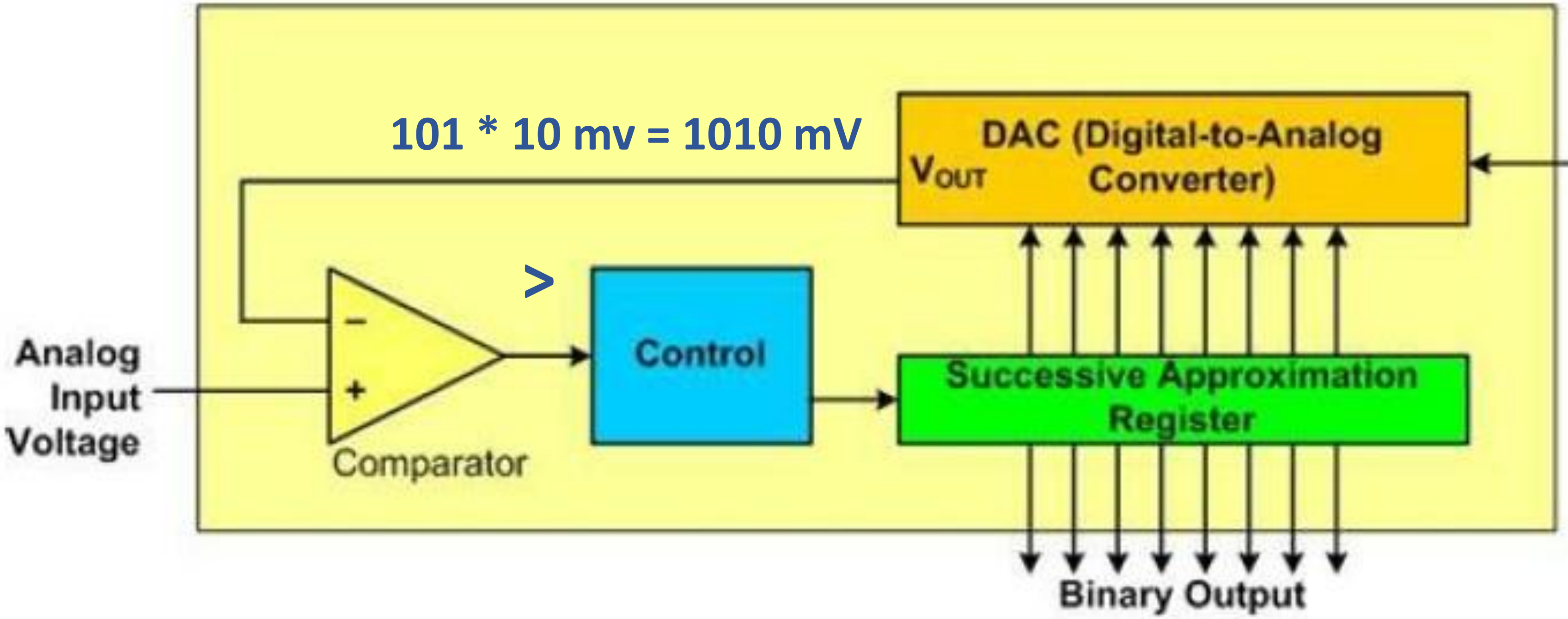
- # ADC:
  - ## Successive Approximation ADC:

Assume
Step Size
10 mV

101 * 10 mv = 1010 mV

>

1 V



101

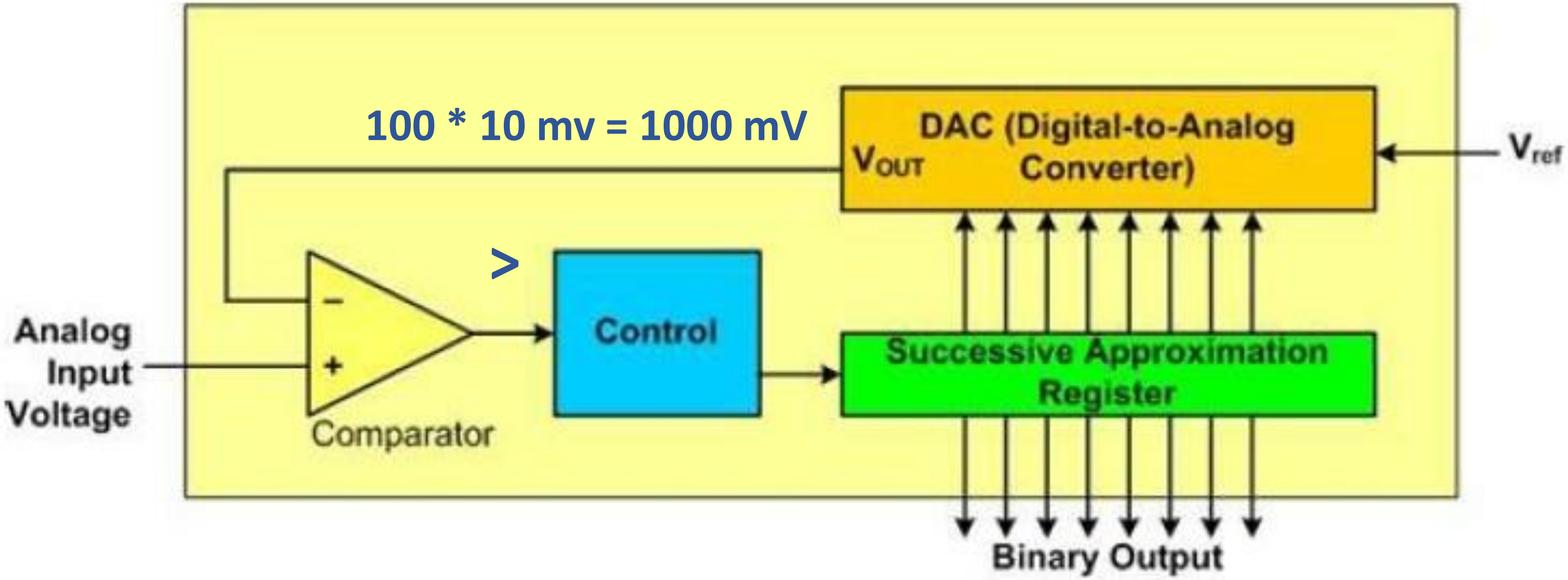
0110 0101

• ADC:

• Successive Approximation ADC:

Assume
Step Size
10 mV

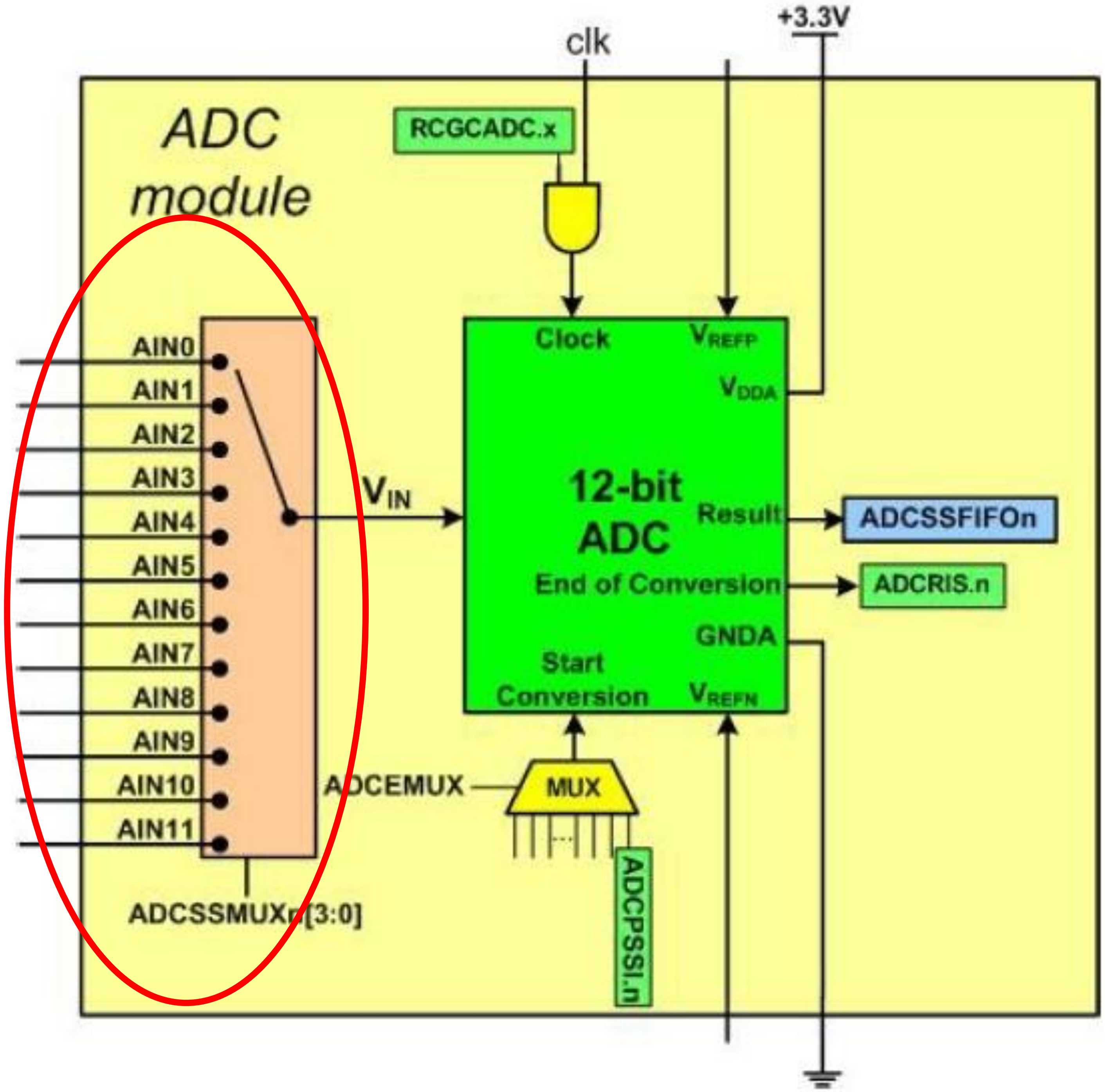
1 V



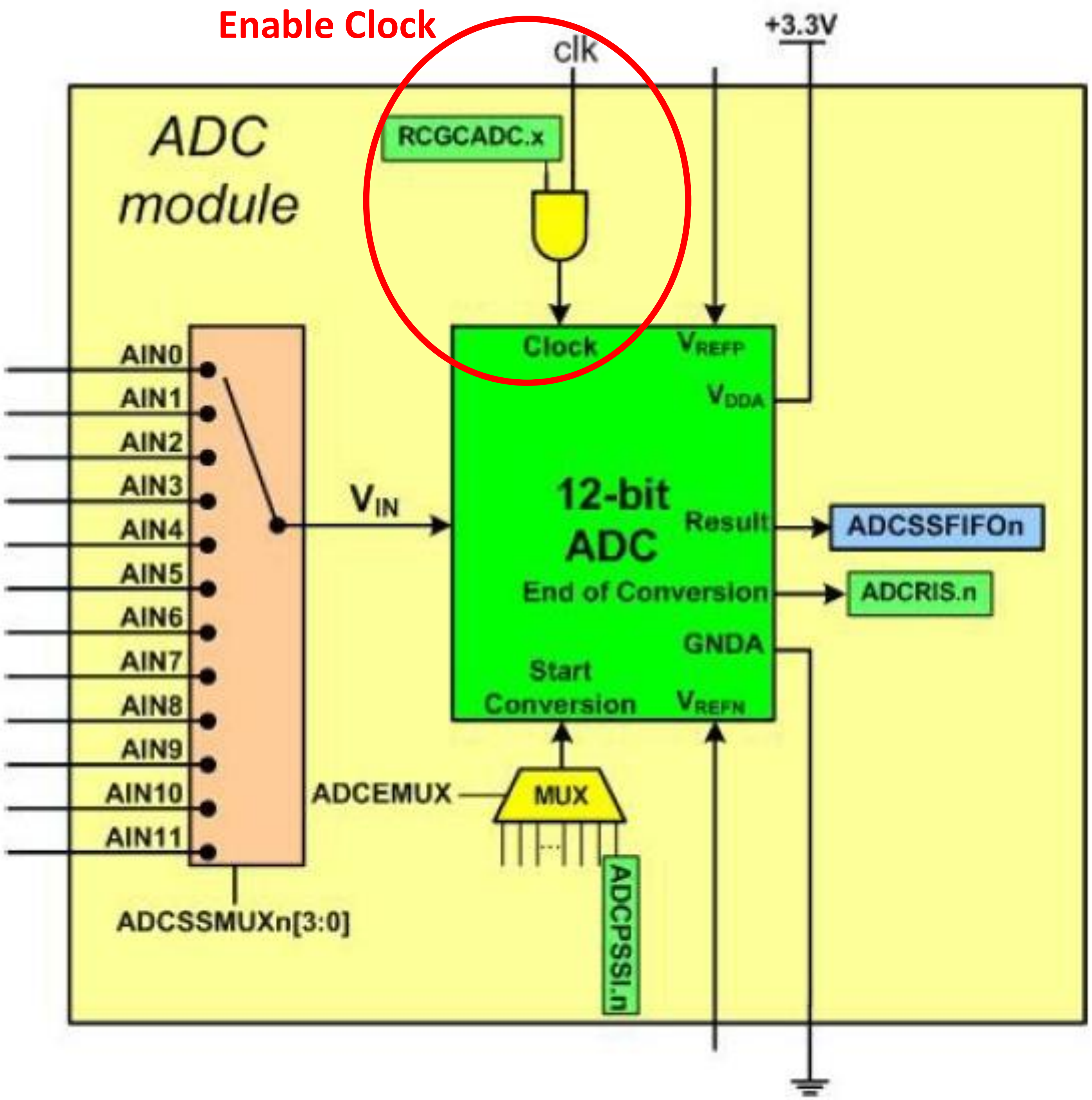
0110 0100

- **ADC:**

Channels

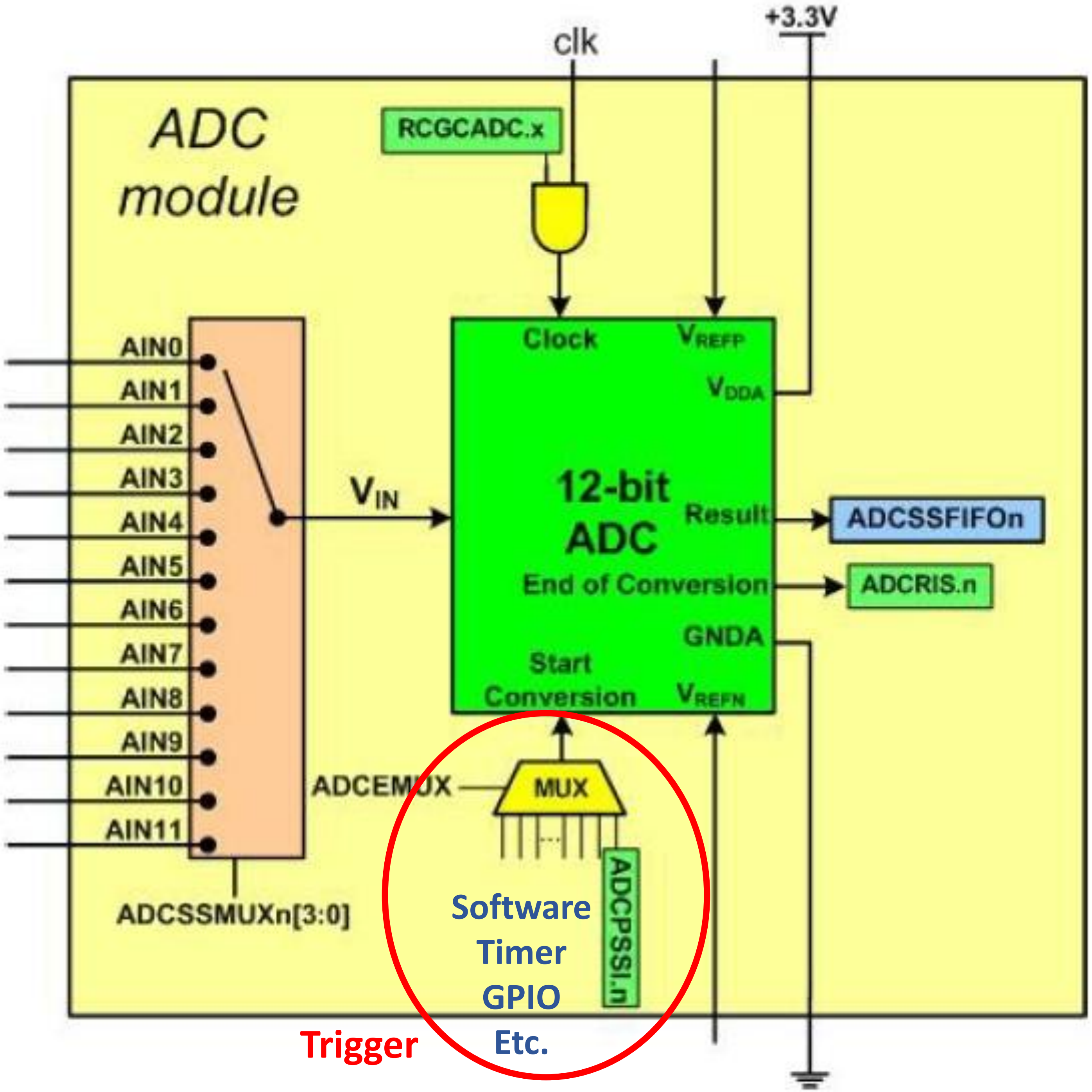


- **ADC:**



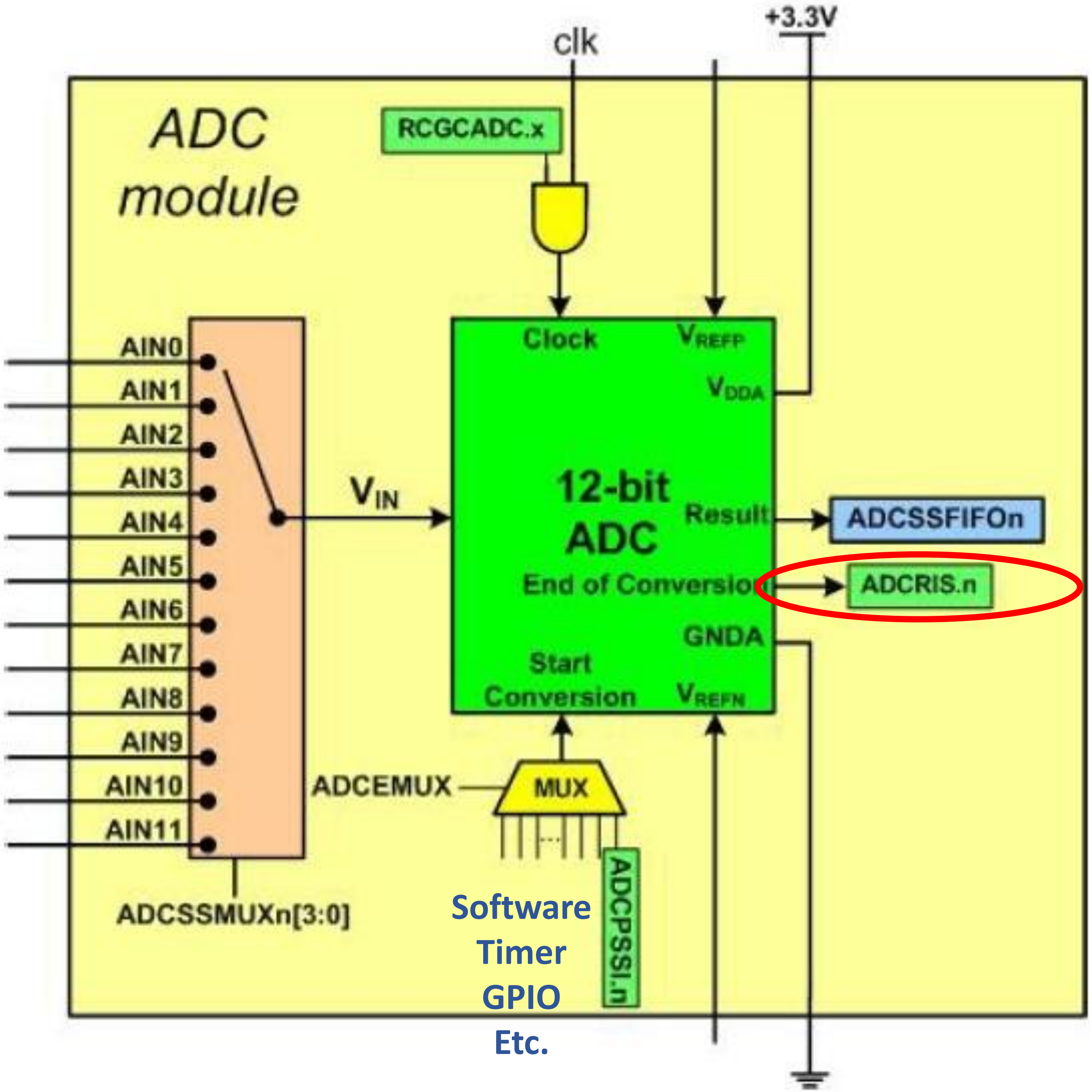
Enable Clock
clk
+3.3V
ADC module
RCGCADC.x
Clock
VREFP
VDDA
AIN0
AIN1
AIN2
AIN3
AIN4
AIN5
AIN6
AIN7
AIN8
AIN9
AIN10
AIN11
VIN
12-bit ADC
Result
ADCSSFIFOn
End of Conversion
ADCRIS.n
GNDA
Start Conversion
VREFN
ADCEMUX
MUX
ADCPSSI.n
ADCSSMUXn[3:0]

- **ADC:**



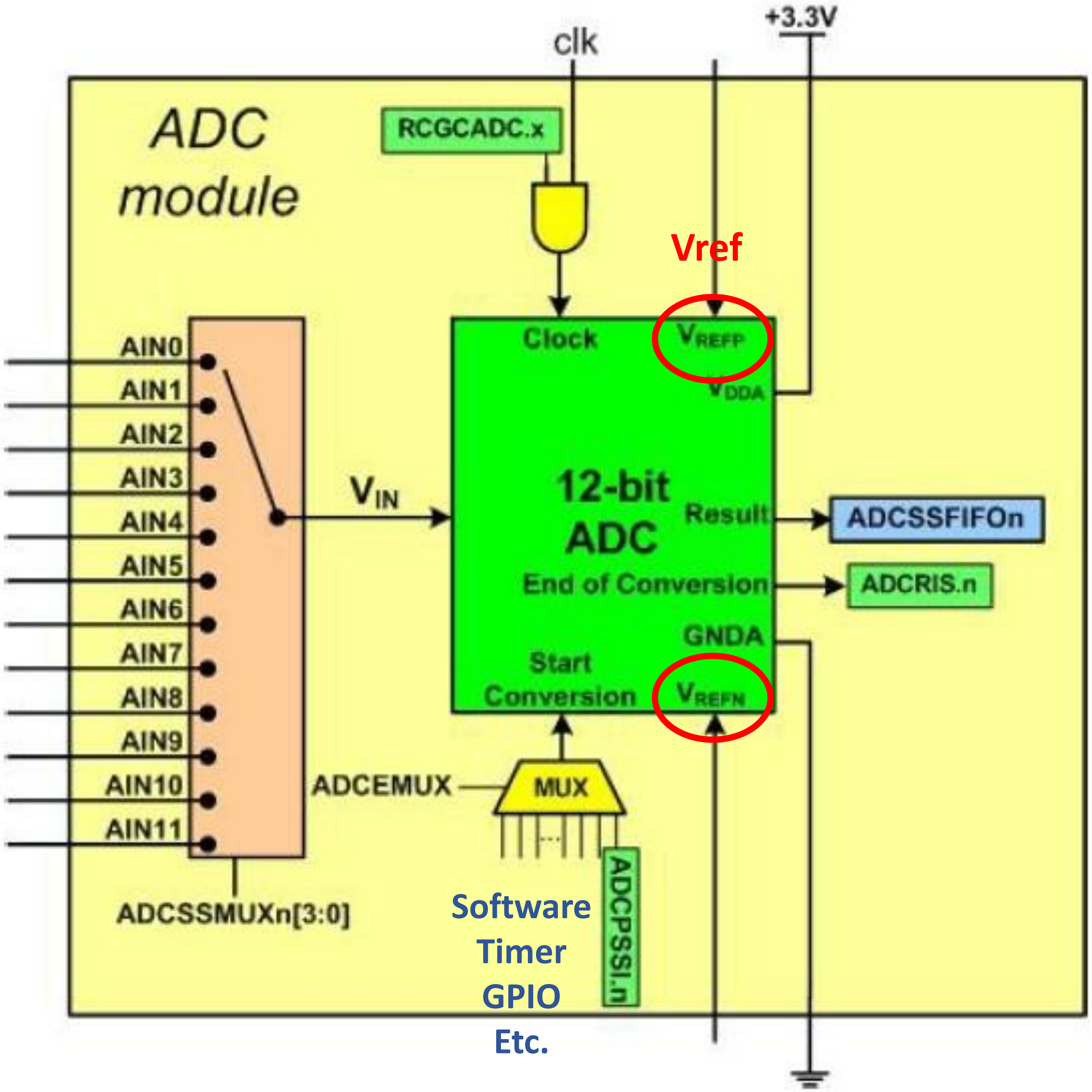
ADC module
clk
+3.3V
RCGCADC.x
Clock
VREFP
VDDA
12-bit ADC
Result
End of Conversion
GNDA
Start Conversion
VREFN
ADCSSFIFOn
ADCRIS.n
AIN0
AIN1
AIN2
AIN3
AIN4
AIN5
AIN6
AIN7
AIN8
AIN9
AIN10
AIN11
VIN
ADCEMUX
MUX
ADCPSSI.n
ADCSSMUXn[3:0]
Software
Timer
GPIO
Etc.
Trigger

- **ADC:**



Poll This Register
or use Interrupt

• **ADC:**

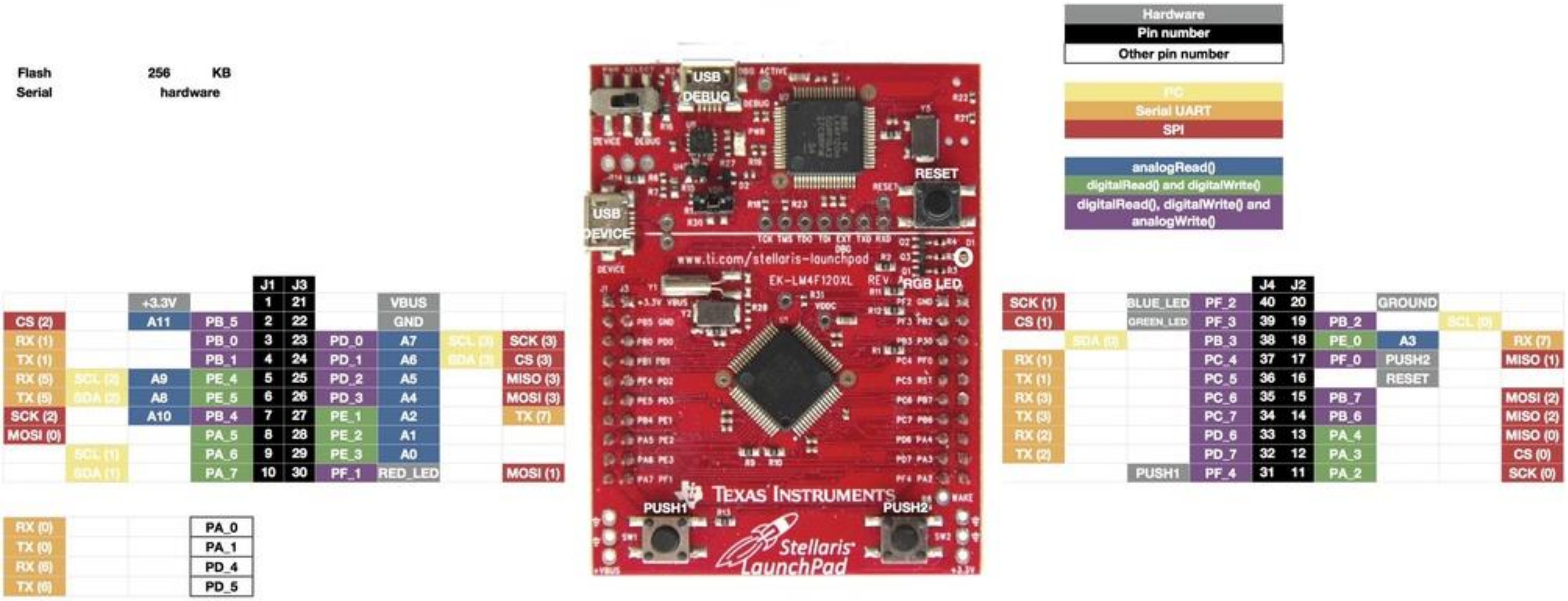


+3.3V
clk
ADC module
RCGCADC.x
Vref
Clock
VREFP
VDDA
AIN0
AIN1
AIN2
AIN3
AIN4
AIN5
AIN6
AIN7
AIN8
AIN9
AIN10
AIN11
VIN
12-bit ADC
Result
ADCSSFIFOn
End of Conversion
ADCRIS.n
GNDA
Start Conversion
VREFN
ADCEMUX
MUX
ADCPSSI.n
ADCSSMUXn[3:0]
Software
Timer
GPIO
Etc.

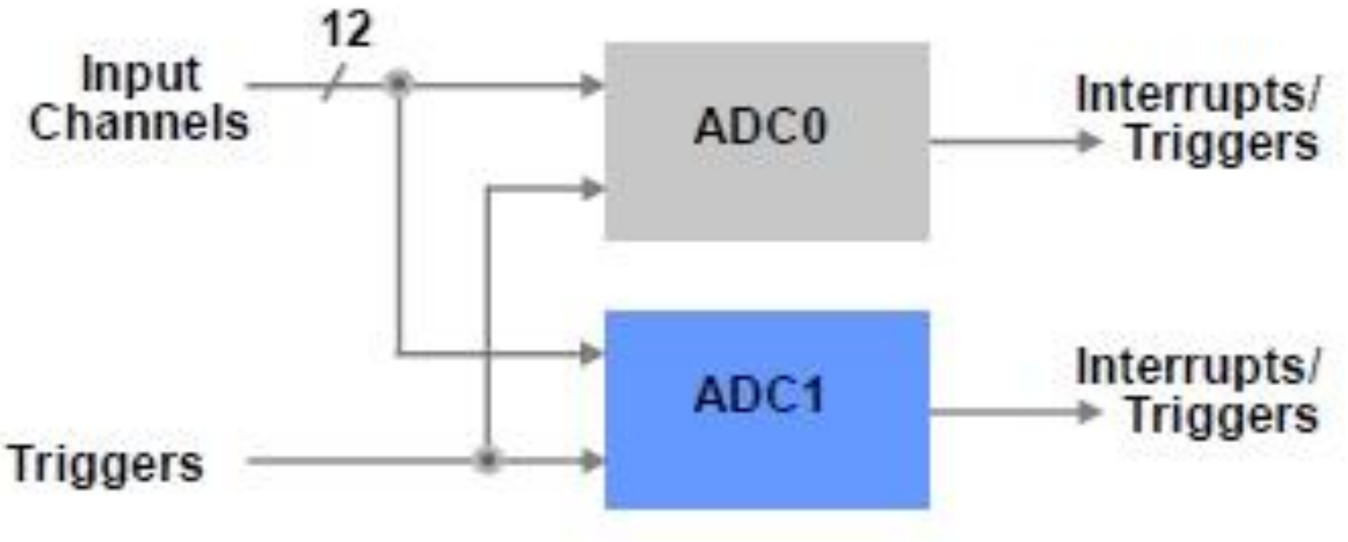
TM4C ADC Specs

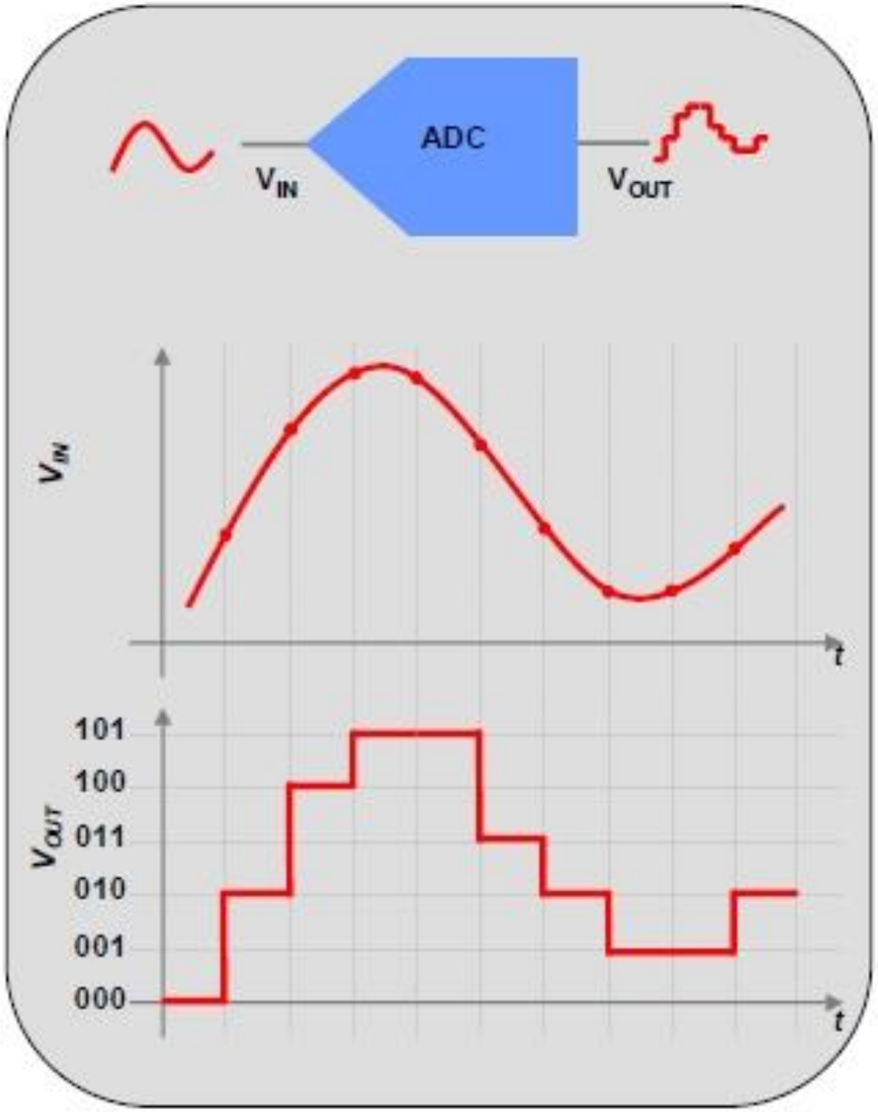
- Reference voltages:
  - VREFP – tied to 3.3V on the Launchpad.
  - VREFN – tied to ground on the Launchpad.
- Range & Resolution
  - Range = **0 → 4095**
  - Resolution = 3.3V / 4096 = **8.05 mV** (p. 803)
- Max Sampling Speed = 1 million samples/second
- Hardware Averaging
  - Averages 4 samples in hardware during conversion
- Differential Sampling
  - Sample the difference between two inputs
- Internal Temp Sensor
  - Can be sampled for chip temp

• ADC:

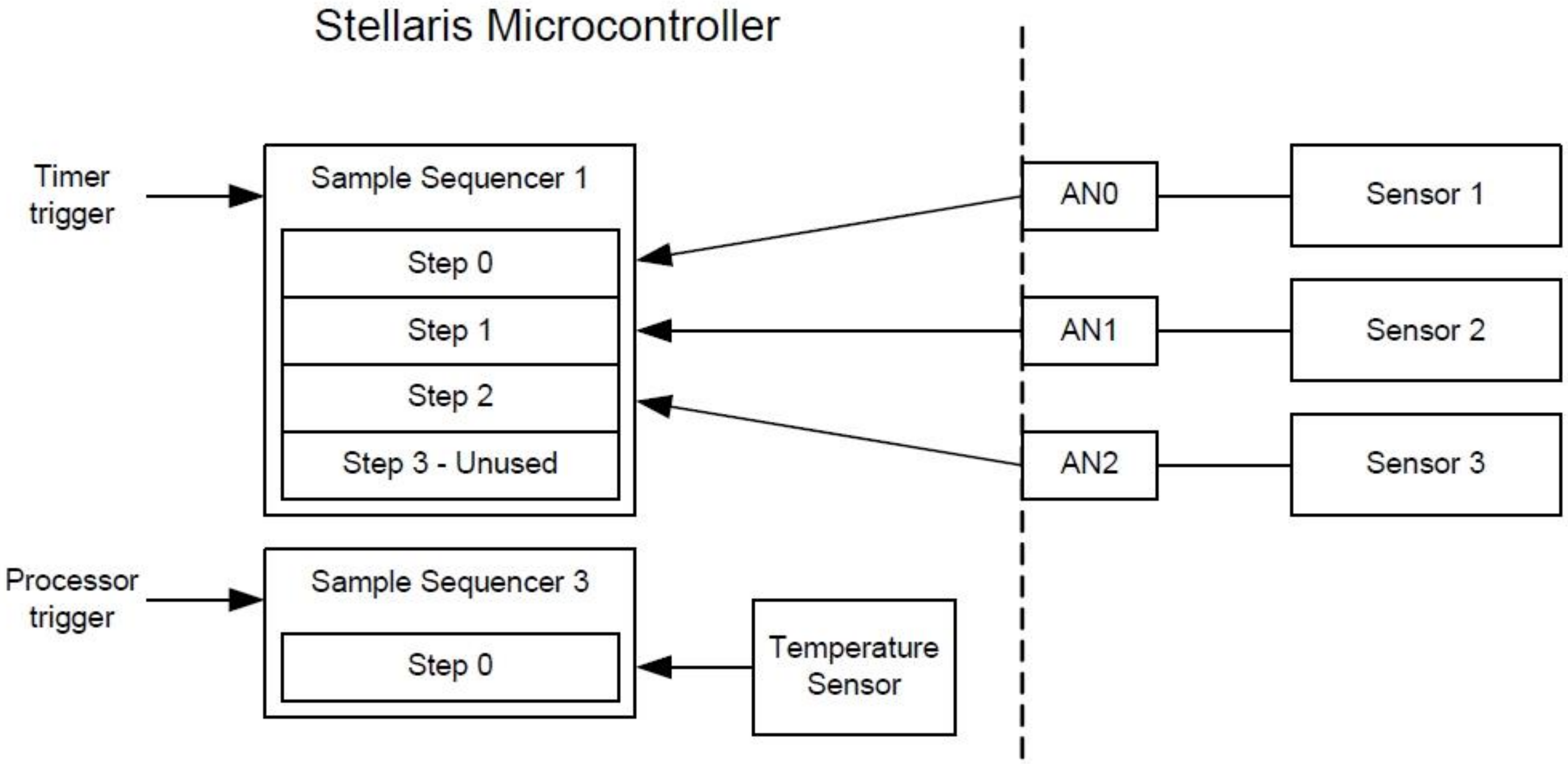

Flash 256 KB
Serial hardware
Hardware
Pin number
Other pin number
I²C
Serial UART
SPI
analogRead()
digitalRead() and digitalWrite()
digitalRead(), digitalWrite() and analogWrite()
J1 J3
+3.3V 1 21 VBUS
CS (2) A11 PB_5 2 22 GND
RX (1) PB_0 3 23 PD_0 A7 SCL (3) SCK (3)
TX (1) PB_1 4 24 PD_1 A6 SDA (3) CS (3)
RX (5) SCL (2) A9 PE_4 5 25 PD_2 A5 MISO (3)
TX (5) SDA (2) A8 PE_5 6 26 PD_3 A4 MOSI (3)
SCK (2) A10 PB_4 7 27 PE_1 A2 TX (7)
MOSI (0) PA_5 8 28 PE_2 A1
SCL (1) PA_6 9 29 PE_3 A0
SDA (1) PA_7 10 30 PF_1 RED_LED MOSI (1)
RX (0) PA_0
TX (0) PA_1
RX (6) PD_4
TX (6) PD_5
USB DEBUG
USB DEVICE
RESET
RGB LED
www.ti.com/stellaris-launchpad
EK-LM4F120XL
PUSH1
PUSH2
TEXAS INSTRUMENTS
Stellaris LaunchPad
J4 J2
SCK (1) BLUE_LED PF_2 40 20 GROUND
CS (1) GREEN_LED PF_3 39 19 PB_2 SCL (0)
SDA (0) PB_3 38 18 PE_0 A3 RX (7)
RX (1) PC_4 37 17 PF_0 PUSH2 MISO (1)
TX (1) PC_5 36 16 RESET
RX (3) PC_6 35 15 PB_7 MOSI (2)
TX (3) PC_7 34 14 PB_6 MISO (2)
RX (2) PD_6 33 13 PA_4 MISO (0)
TX (2) PD_7 32 12 PA_3 CS (0)
PUSH1 PF_4 31 11 PA_2 SCK (0)

- **ADC:**

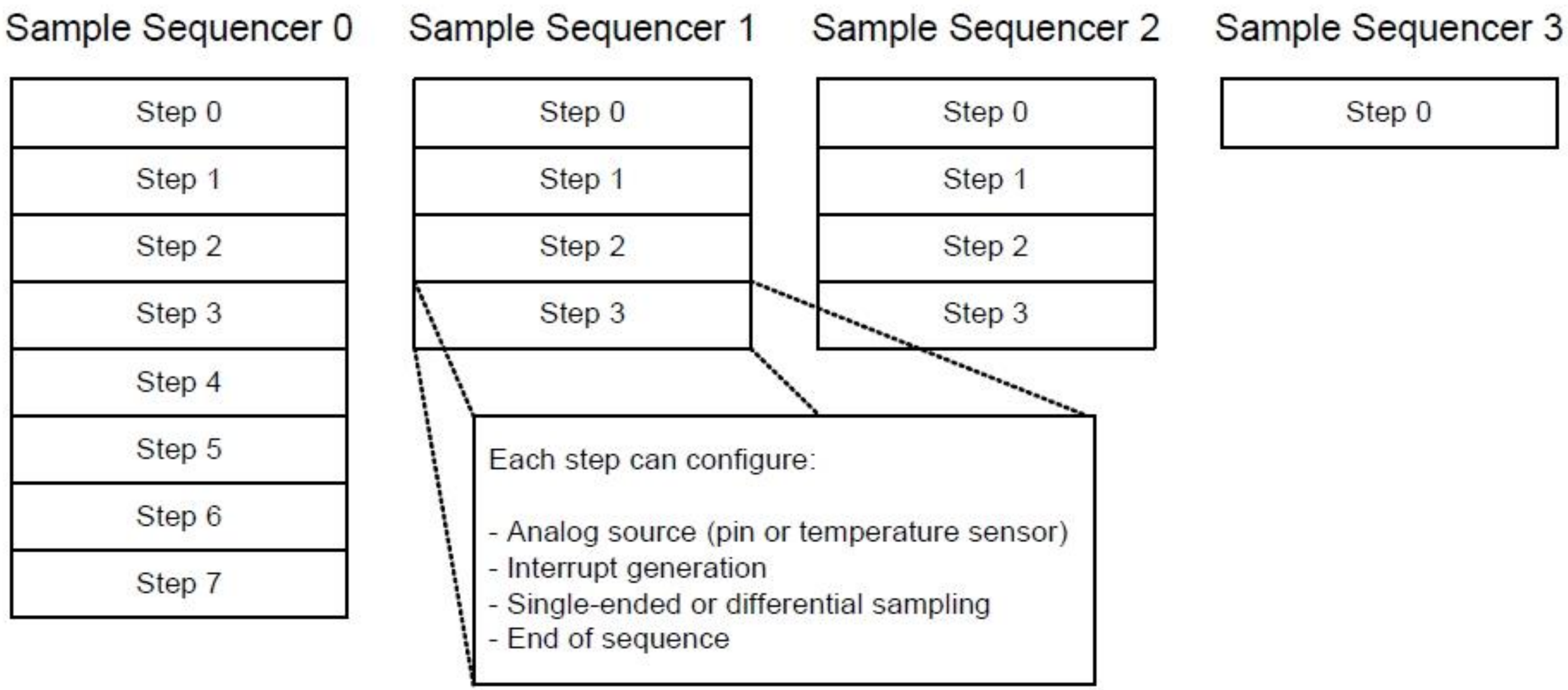

12
Input Channels
ADC0
Interrupts/ Triggers
ADC1
Interrupts/ Triggers
Triggers


ADC
V_{IN}
V_{OUT}
V_{IN}
t
101
100
011
010
001
000
V_{OUT}
t

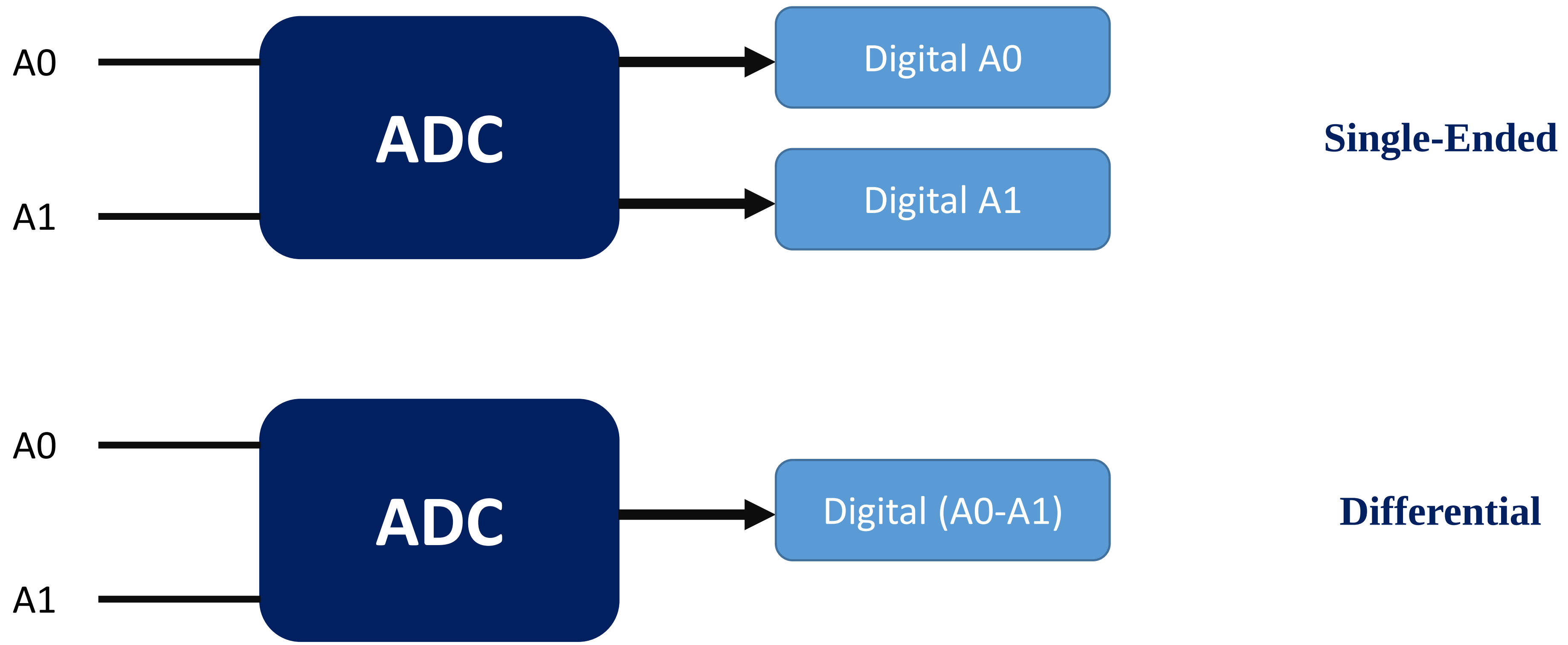
- **ADC:**
  - **Sample Sequencer:**



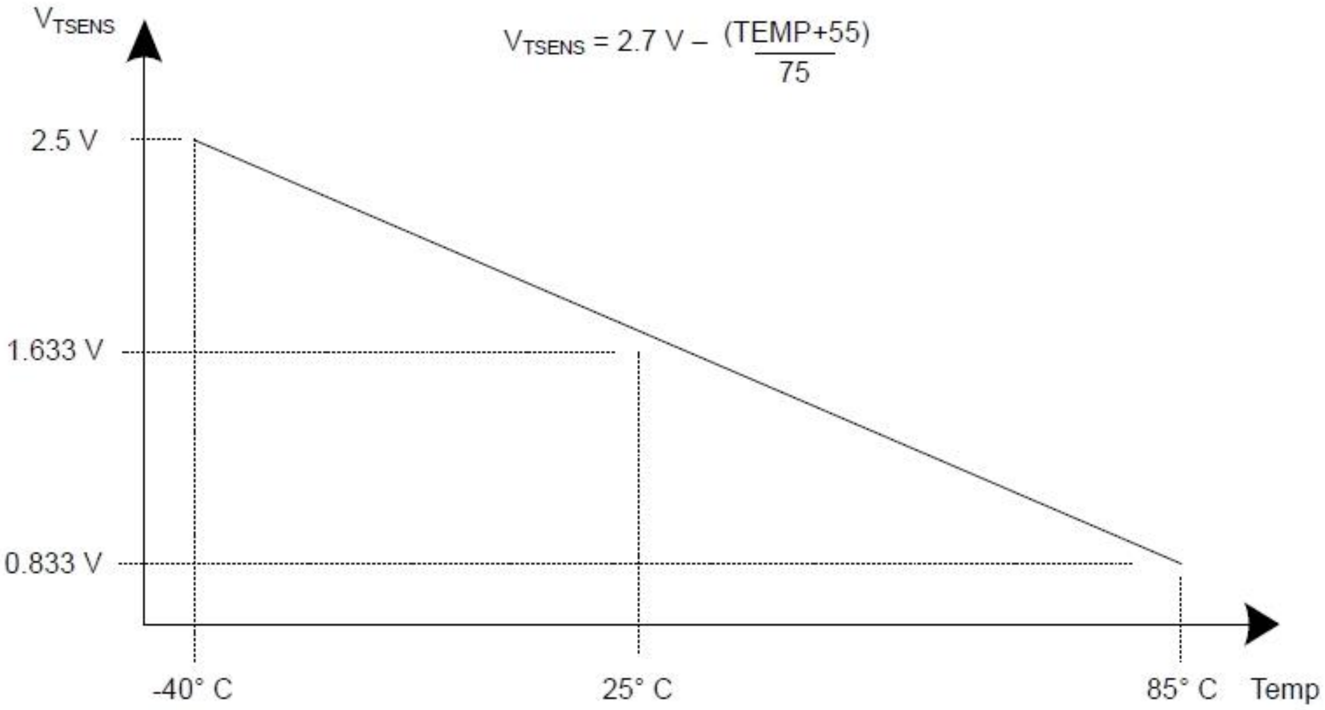
- **ADC:**
  - **Sample Sequencer:**



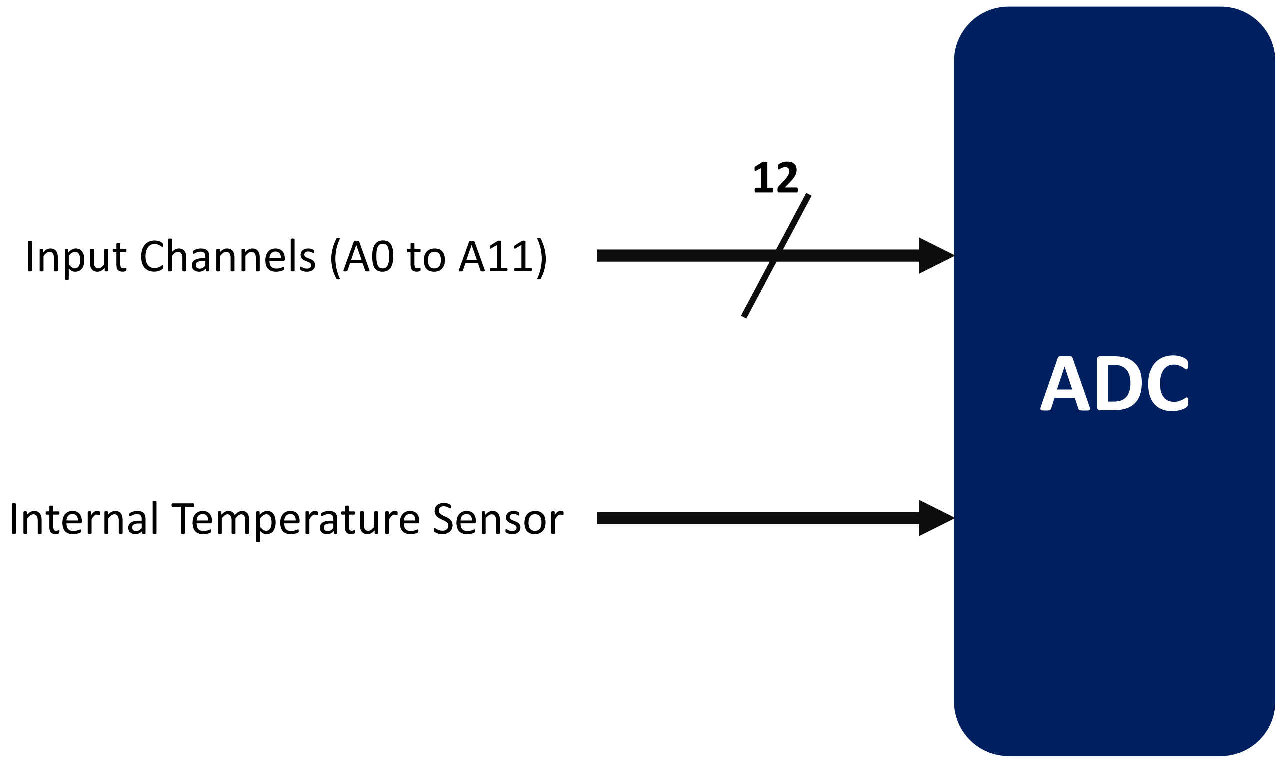
- **ADC:**
  - **Differential versus Single-Ended:**

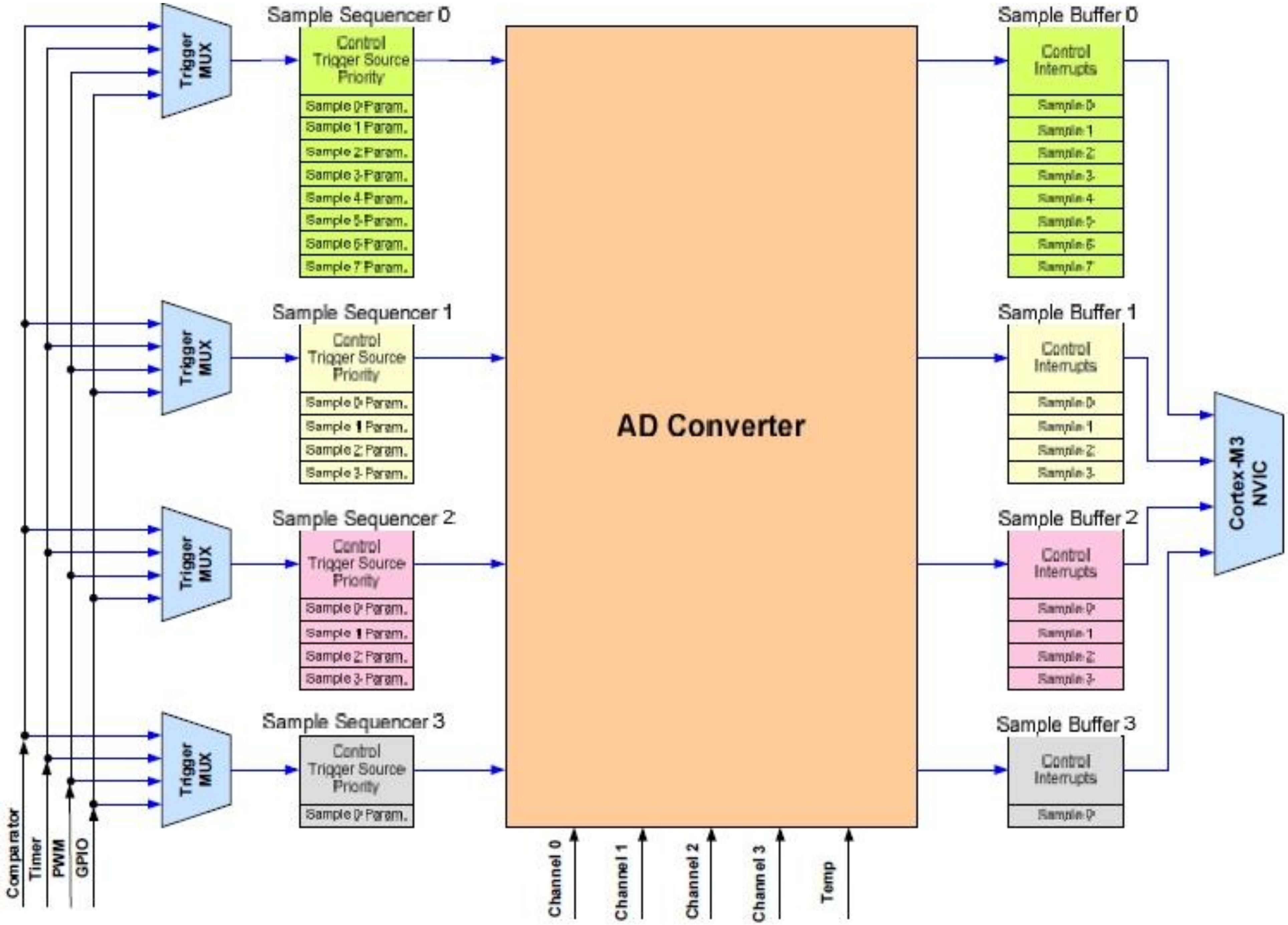


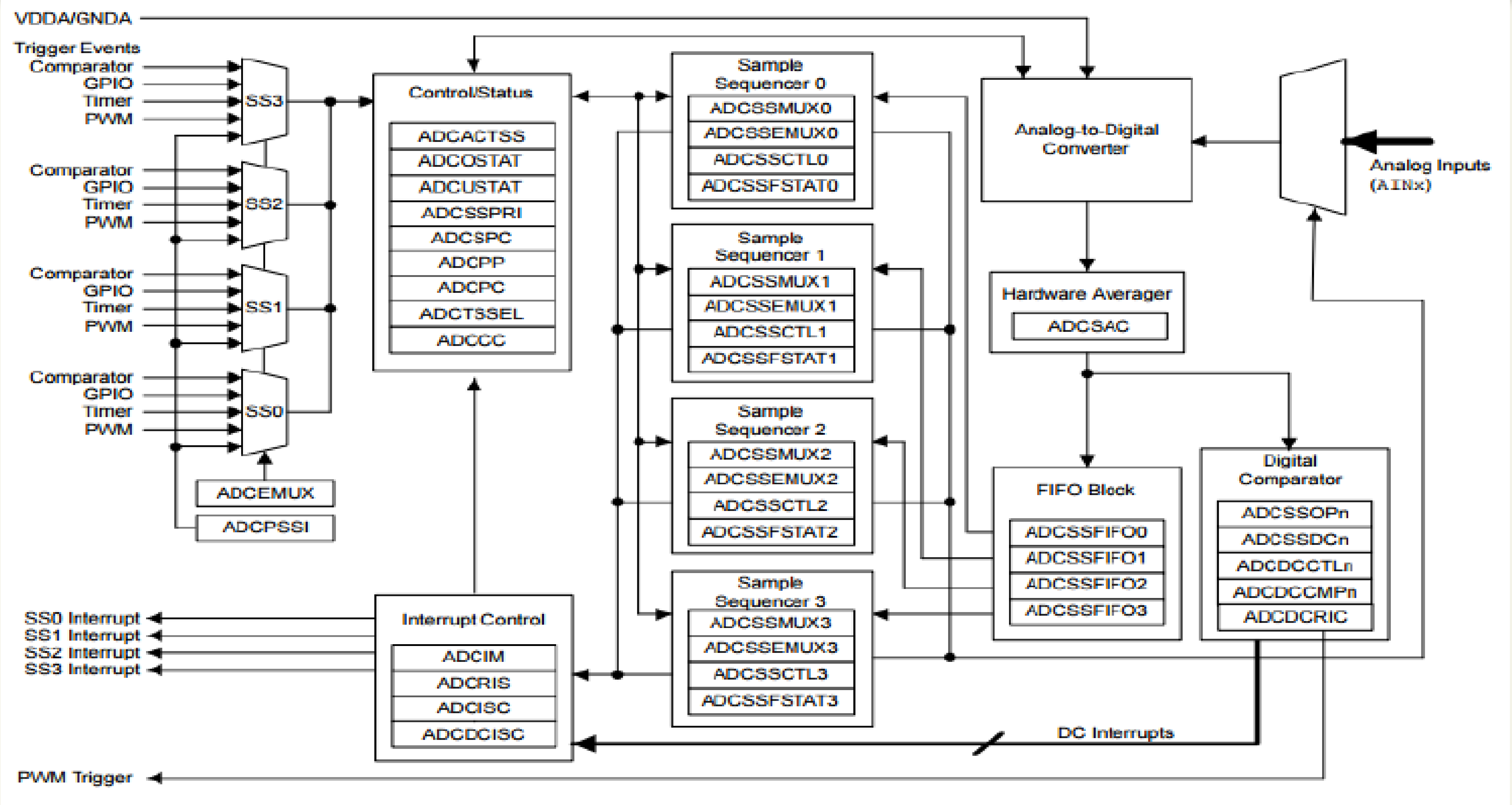
- **ADC:**
  - **Internal Temperature Sensor:**



- **ADC:**
  - **ADC Input:**




Sample Sequencer 0
Trigger MUX
Control
Trigger Source
Priority
Sample 0 Param.
Sample 1 Param.
Sample 2 Param.
Sample 3 Param.
Sample 4 Param.
Sample 5 Param.
Sample 6 Param.
Sample 7 Param.
Sample Sequencer 1
Sample Sequencer 2
Sample Sequencer 3
AD Converter
Sample Buffer 0
Control
Interrupts
Sample 0
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
Sample 6
Sample 7
Sample Buffer 1
Sample Buffer 2
Sample Buffer 3
Cortex-M3
NVIC
Comparator
Timer
PWM
GPIO
Channel 0
Channel 1
Channel 2
Channel 3
Temp



Datasheet Page:801

Example 3: TIVAWareADC

Table 7.18. API functions used to configure and handle sample sequencers.

API Function	Parameters	Description
void ADCSequenceConfigure(uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t ui32Trigger, uint32_t ui32Priority)	ui32Base is the base address of the ADC module. ui32SequenceNum is the sample sequencer number. ui32Trigger is the trigger source that initiates the sample sequence, it must be ADC_TRIGGER_ values. ui32Priority is the relative priority of the sample sequencer.	Configure the initiation criteria for a sample sequencer. The trigger condition and priority are set. ADC_TRIGGER_ values include: ADC_TRIGGER_PROCESSOR - A trigger generated by processor, via the ADCProcessorTrigger() function. ADC_TRIGGER_COMP0 - A trigger generated by the 1st analog comparator. ADC_TRIGGER_COMP1 - A trigger generated by the 2nd analog comparator. ADC_TRIGGER_COMP2 - A trigger generated by the 3rd analog comparator. ADC_TRIGGER_EXTERNAL - A trigger generated by an input from the Port B4 pin. ADC_TRIGGER_TIMER - A trigger generated by a timer ADC_TRIGGER_PWMn - A trigger generated by the nth PWM generator. ADC_TRIGGER_ALWAYS - A trigger is always asserted.

Table 7.18. API functions used to configure and handle sample sequencers.

void ADCSequenceStepConfigure(uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t ui32Step, uint32_t ui32Config)	ui32Base is the base address of the ADC module. ui32SequenceNum is the sample sequencer number. ui32Step is the sample to be configured. ui32Config is configuration of this sample; must be a logical OR of ADC_CTL_TS, ADC_CTL_IE, ADC_CTL_END, ADC_CTL_D, (ADC_CTL_CH0 ~ADC_CTL_CH11), (ADC_CTL_CMP0~ADC_CTL_CMP7).	Configure the ADC for one sample in a sequencer. The ADC can be configured by ui32Config for: Single-ended or differential operation (ADC_CTL_D bit selects differential mode when set) The channel to be sampled can be chosen (ADC_CTL_CH0 ~ ADC_CTL_CH11 values) The internal temperature sensor can be selected (ADC_CTL_TS bit). This step can be defined as the last in the sequence (ADC_CTL_END bit) It can be configured to cause an interrupt when the step is complete (ADC_CTL_IE bit).

Table 7.18. API functions used to configure and handle sample sequencers.

void ADCSequenceEnable(uint32_t ui32Base, uint32_t ui32SequenceNum)	ui32Base is the base address of the ADC module. ui32SequenceNum is the sample sequencer number.	Enable the specified sample sequencer to be captured when its trigger is detected. A sample sequencer must be configured before it is enabled.
void ADCSequenceDisable(uint32_t ui32Base, uint32_t ui32SequenceNum)	ui32Base is the base address of the ADC module. ui32SequenceNum is the sample sequencer number.	Disable a sample sequencer. A sample sequencer should be disabled before it is configured.
int32_t ADCSequenceDataGet(uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t *pui32Buffer)	ui32Base is the base address of the ADC module. ui32SequenceNum is the sample sequencer number. pui32Buffer is the address where the data is stored.	Get the captured data for a sample sequencer. This function copies data from the specified sample sequencer output FIFO to a memory resident buffer. The number of samples available in the hardware FIFO are copied into the buffer, which is assumed to be large enough to hold that many samples.

Table 7.19. API functions used to configure and control the processor trigger.

API Function	Parameters	Description
void ADCProcessorTrigger(uint32_t ui32Base, uint32_t ui32SequenceNum)	ui32Base is the base address of the ADC module. ui32SequenceNum is the sample sequencer number, with ADC_TRIGGER_WAIT or ADC_TRIGGER_SIGNAL optionally ORed into it.	Trigger a processor-initiated sample sequence if the sample sequencer trigger is configured to ADC_TRIGGER_PROCESSOR. If ADC_TRIGGER_WAIT is ORed into the sequencer number, the processor-initiated trigger is delayed until a later processor-initiated trigger to a different ADC module that specifies ADC_TRIGGER_SIGNAL, allowing multiple ADCs to start from a processor-initiated trigger in a synchronous manner.

Table 7.20. API functions used to configure and handle ADC interrupts.

API Function	Parameters	Description
void ADCIntRegister(uint32_t ui32Base, uint32_t ui32SequenceNum, void (*pfnHandler)(void))	`ui32Base` is the base address of the ADC module. `ui32SequenceNum` is the sample sequencer number. `pfnHandler` is a pointer to the function to be called when the ADC sample sequencer interrupt occurs.	Register an interrupt handler for an ADC interrupt. Set the handler to be called when a sample sequencer interrupt occurs. This function enables the global interrupt in the interrupt controller; the sequencer interrupt must be enabled with `ADCIntEnable()`. It is the interrupt handler's responsibility to clear the interrupt source via `ADCIntClear()`.
void ADCIntUnregister(uint32_t ui32Base, uint32_t ui32SequenceNum)	`ui32Base` is the base address of the ADC module. `ui32SequenceNum` is the sample sequencer number	Unregister the interrupt handler for an ADC interrupt. Unregister the interrupt handler. This function disables the global interrupt in the interrupt controller; the sequencer interrupt must be disabled via `ADCIntDisable()`.
void ADCIntEnable(uint32_t ui32Base, uint32_t ui32SequenceNum)	`ui32Base` is the base address of the ADC module. `ui32SequenceNum` is the sample sequencer number.	Enable the requested sample sequencer interrupt. Any outstanding interrupts are cleared before enabling the sample sequence interrupt.

Table 7.20. API functions used to configure and handle ADC interrupts.

API Function	Parameters	Description
void `ADCIntDisable(` uint32_t ui32Base, uint32_t ui32SequenceNum)	`ui32Base` is the base address of the ADC module. `ui32SequenceNum` is the sample sequencer number.	Disable the requested sample sequencer interrupt.
uint32_t `ADCIntStatus(` uint32_t ui32Base, uint32_t ui32SequenceNum, bool bMasked)	`ui32Base` is the base address of the ADC module. `ui32SequenceNum` is the sample sequence number. `bMasked` is false if the raw interrupt status is required and true if the masked interrupt status is required.	Gets the current interrupt status. This function returns the interrupt status for the specified sample sequencer. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.
void `ADCIntClear(` uint32_t ui32Base, uint32_t ui32SequenceNum)	`ui32Base` is the base address of the ADC module. `ui32SequenceNum` is the sample sequencer number.	Clear sample sequencer interrupt source. The specified sample sequencer interrupt is cleared, so that it no longer asserts. This function must be called in the interrupt handler to make the next interrupt to be triggered again in the future.

```c
//*****************************************************************************
// SDADCPoll.c - Main Application for ADC0 - SS0 – Channel 1
//*****************************************************************************
#include <stdint.h>
#include <stdbool.h>
#include "driverlib\adc.h"
#include "inc\hw_memmap.h"
#include "inc\hw_types.h"
#include "driverlib\sysctl.h"
#include "driverlib\gpio.h"
int main(void)
{
    uint32_t ui32Value, ssn = 0, sample = 0;
   // Enable clocks for GPIO Ports B & F and ADC0
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
   // Configure GPIO Ports B & F as output ports
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3|GPIO_PIN_2|
                                   GPIO_PIN_1|GPIO_PIN_0);
   // Configure ADC0 - SS0 – Channel 1 & the First Sample
   ADCSequenceDisable(ADC0_BASE, ssn); a
   ADCSequenceConfigure(ADC0_BASE, ssn, ADC_TRIGGER_PROCESSOR, 0);
   ADCSequenceStepConfigure(ADC0_BASE, ssn, sample, ADC_CTL_IE|ADC_CTL_END|
                                      ADC_CTL_CH1);
   ADCSequenceEnable(ADC0_BASE, ssn);
   while (1)
   {
       // Trigger the sample sequencer 0 – SS0.
       ADCProcessorTrigger(ADC0_BASE, ssn);
       // Wait until the sample sequence has completed
       while(!ADCIntStatus(ADC0_BASE, ssn, false)){}
       // Read the value from the ADC.
       ADCSequenceDataGet(ADC0_BASE, ssn, &ui32Value);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, ui32Value >> 8);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0,
                         ui32Value >> 8);
       ADCIntClear(ADC0_BASE, ssn);
   }
}
```

```c
//*****************************************************************************
// SDADCPoll.c - Main Application for ADC0 - SS0 – Channel 1
//*****************************************************************************
#include <stdint.h>
#include <stdbool.h>
#include "driverlib\adc.h"
#include "inc\hw_memmap.h"
#include "inc\hw_types.h"
#include "driverlib\sysctl.h"
#include "driverlib\gpio.h"
```

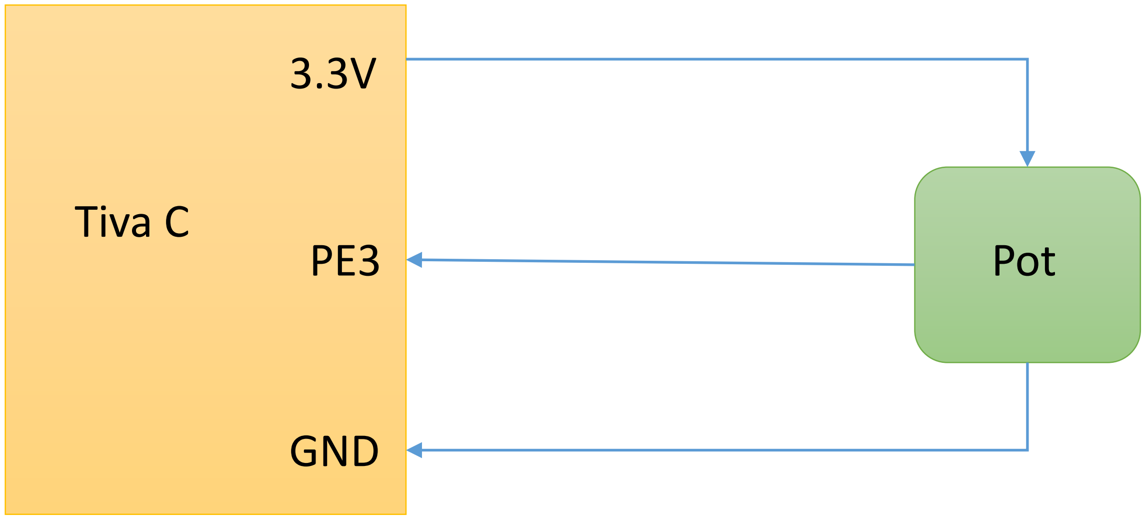
```
int main(void)
{
    uint32_t  ui32Value,  ssn = 0,  sample = 0;

   // Enable clocks for GPIO Ports B & F and ADC0
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

   // Configure GPIO Ports B & F as output ports
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,  GPIO_PIN_3|GPIO_PIN_2|
                                          GPIO_PIN_1|GPIO_PIN_0);
```

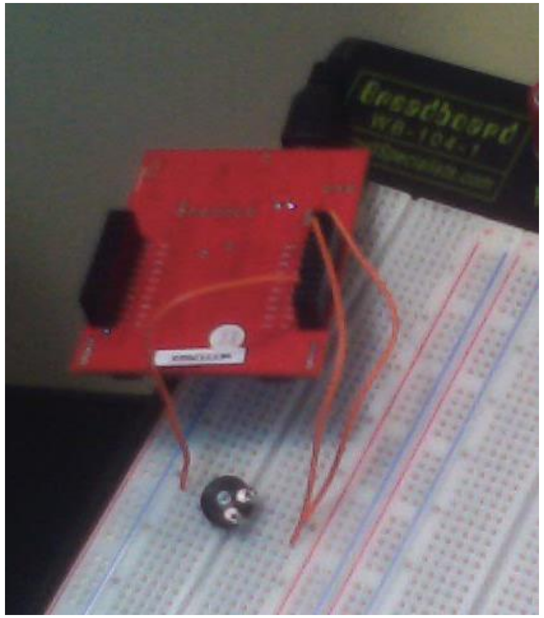
```
// Configure ADC0 - SS0 – Channel 1 & the First Sample
ADCSequenceDisable(ADC0_BASE,  ssn); a
ADCSequenceConfigure(ADC0_BASE, ssn, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, ssn, sample, ADC_CTL_IE|ADC_CTL_END|
                                   ADC_CTL_CH1);
ADCSequenceEnable(ADC0_BASE,  ssn);
```

```
    while (1)
    {
        // Trigger the sample sequencer 0 – SS0.
        ADCProcessorTrigger(ADC0_BASE, ssn);

        // Wait until the sample sequence has completed
        while(!ADCIntStatus(ADC0_BASE, ssn, false)){}

        // Read the value from the ADC.
        ADCSequenceDataGet(ADC0_BASE, ssn, &ui32Value);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, ui32Value >> 8);
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0,
                            ui32Value >> 8);
        ADCIntClear(ADC0_BASE, ssn);
    }
}
```

TIVA Ware Example 4 setup

It's actually that simple!





Example ADC Code

1. Set up clock for 40 MHz
```
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
```
2. Enable peripheral ADC0, then reset it to apply changes
```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
SysCtlPeripheralReset(SYSCTL_PERIPH_ADC0);
```
3. Disable ADC0 sequencer 3
```
ADCSequenceDisable(ADC0_BASE,3);
```
4. Configure ADC sequencer
```
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
```
5. Configure steps for sequences
```
ADCSequenceStepConfigure(ADC0_BASE, 3, 0,  ADC_CTL_CH0 /*AIN0*/ | ADC_CTL_IE | ADC_CTL_END);
```
6. Enable peripheral GPIO port E
```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
```

Example ADC Code

1. Activate PE3’s alternate function (AIN0)
   ```
   GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);
   ```
2. Set up interrupt handler for ADC0 sequencer 3
   ```
   ADCIntRegister(ADC0_BASE, 3, &ISR_ADC_Read);
   ```
3. Enable interrupts for ADC0 sequencer 3
   ```
   ADCIntEnable(ADC0_BASE, 3);
   ```
4. Re-enable ADC0 sequencer 3
   ```
   ADCSequenceEnable(ADC0_BASE, 3);
   ```
5. Enable global interrupts
   ```
   IntMasterEnable();
   ```
6. Post request for ADC conversion (in a loop)
   ```
   ADCProcessorTrigger(ADC0_BASE, 3);
   ```

Example ADC Code (ISR handler)

1. Clear interrupt flag for ADC0 Sequencer 3

```
ADCIntClear(ADC0_BASE, 3);
```

2. Wait until conversion is done

```
while(!ADCIntStatus(ADC0_BASE, 3, false)) {}
```

3. Read data, and process it

```
uint32_t result;
ADCSequenceDataGet(ADC0_BASE, 3, &result);
//Convert to voltage.
float voltage = result*.000805664;
printf("%04u\t%.02fv\n", result, voltage);
```

Libraries to include

- From the TivaWare peripheral drivers:

```
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_adc.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_gpio.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
```

- File path to TivaWare libraries and examples:
  - C:\ti\TivaWare_C_Series-2.1.1.71\examples\peripherals