# Laboratorio di Architetture e Programmazione dei Sistemi Elettronici Industriali

Prof. Luca Benini <luca.benini@unibo.it>

Simone Benatti <simone.benatti@unibo.it>

Filippo Casamassima<filippo.casamassima@unibo.it>

# #8 ADC

# ADC

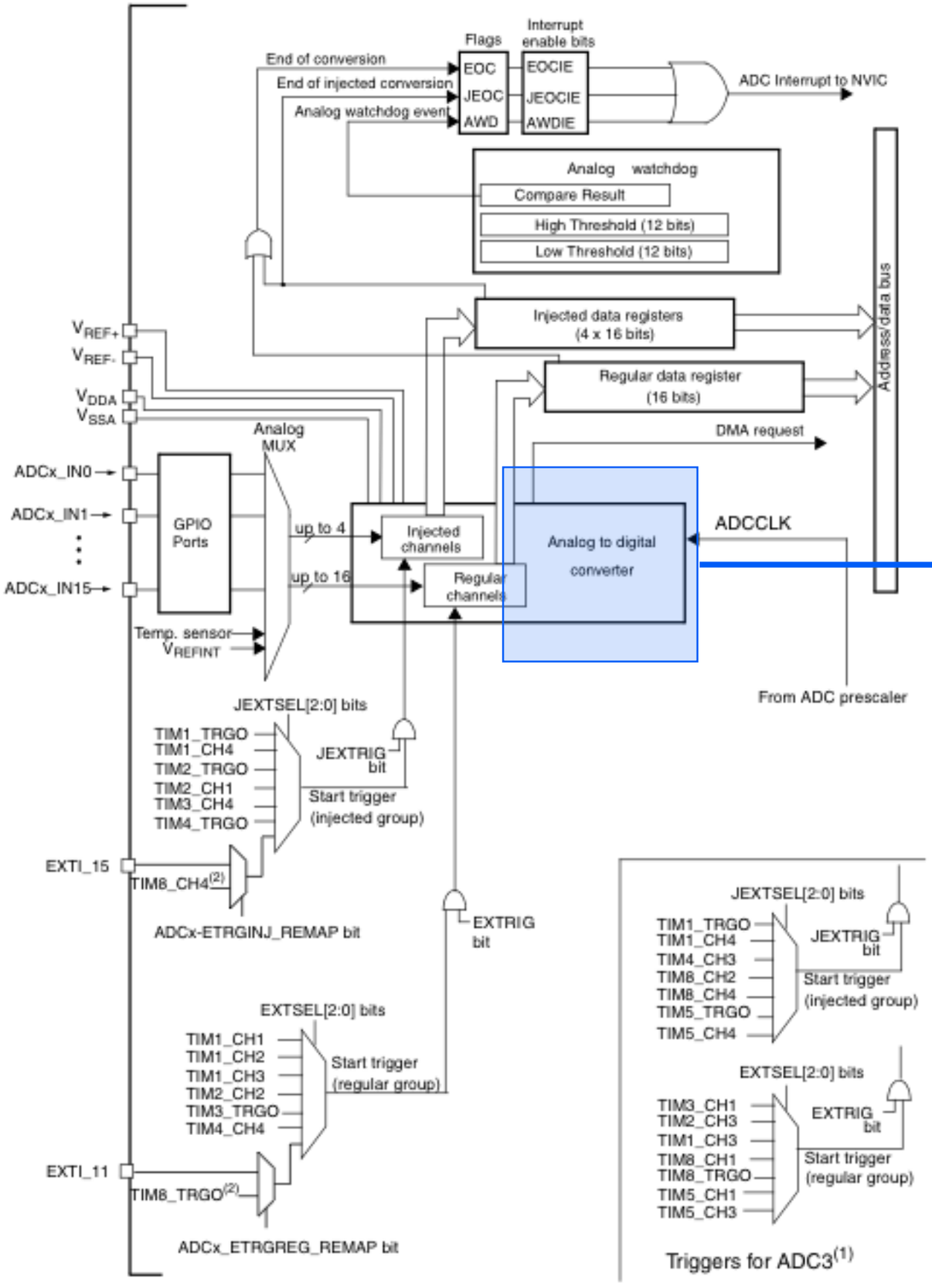
- The **12-bit ADC** is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it measure signals from **16 external and two internal sources**.

- A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.

- The **analog watchdog** feature allows the application to detect if the input voltage goes outside the user-defined high or low thresholds.

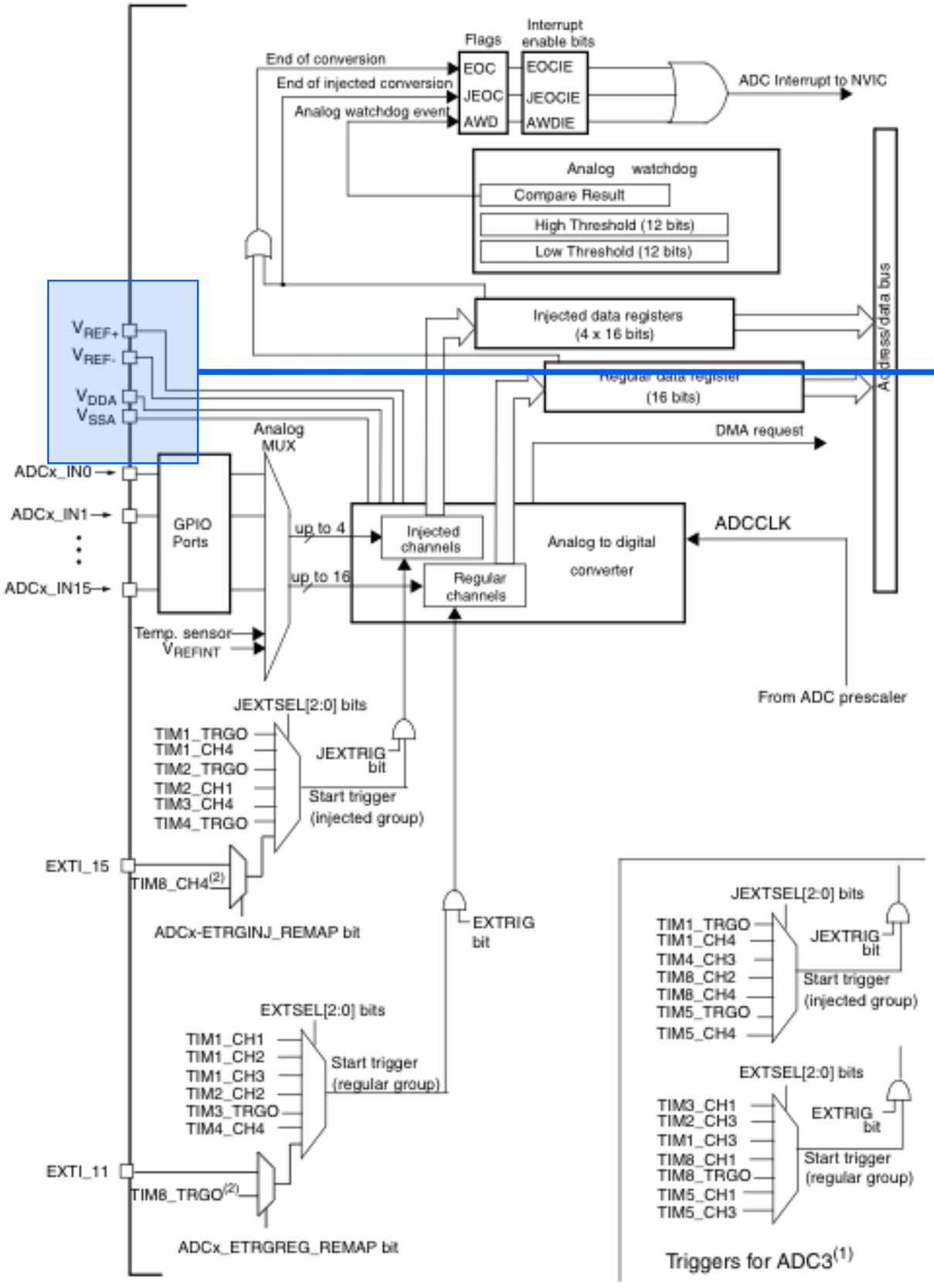
- The **ADC input clock** is generated from the PCLK2 clock divided by a prescaler and it must not exceed 14 MHz

# ADC



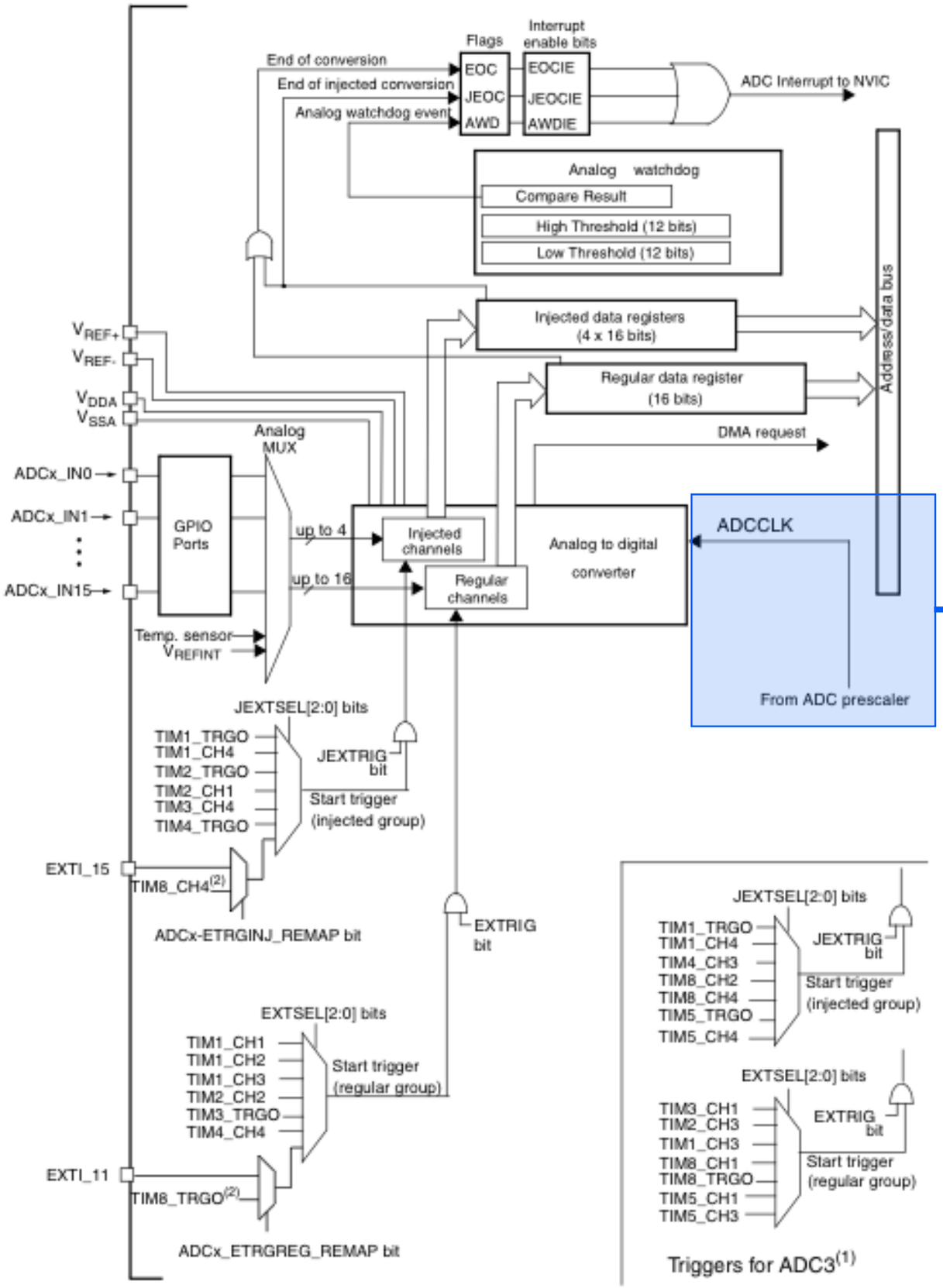

**12-bit** successive approximation **ADC**

# ADC




Analog references / supply

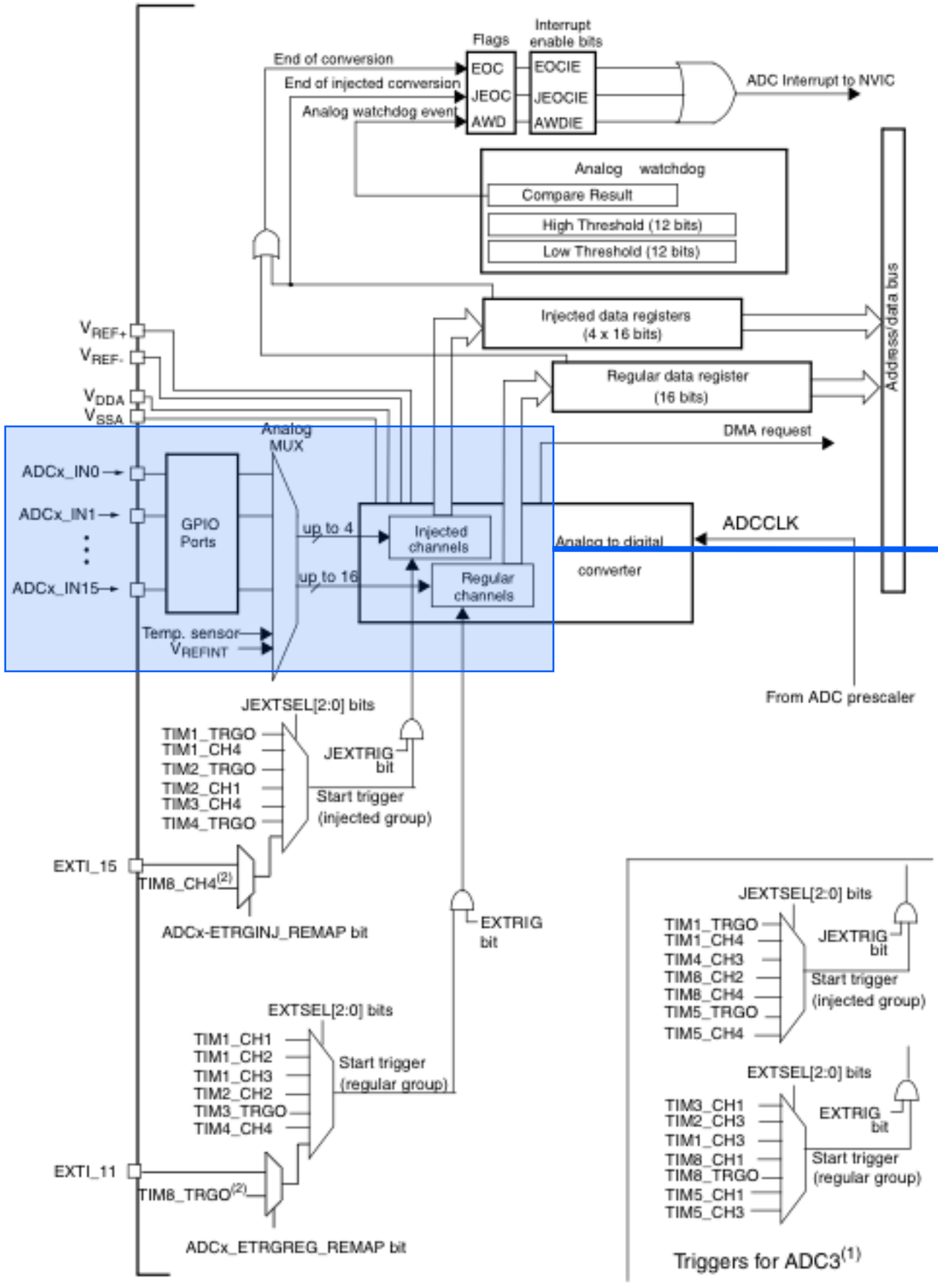
# ADC




The **ADCCLK clock** provided by the Clock Controller is synchronous with the **PCLK2** (APB2 clock). The RCC controller has a dedicated programmable **prescaler** for the ADC clock (**max 14MHz**)

If prescaler == 0 on the STM32 discovery **ADCCLK==24MHz**

# ADC

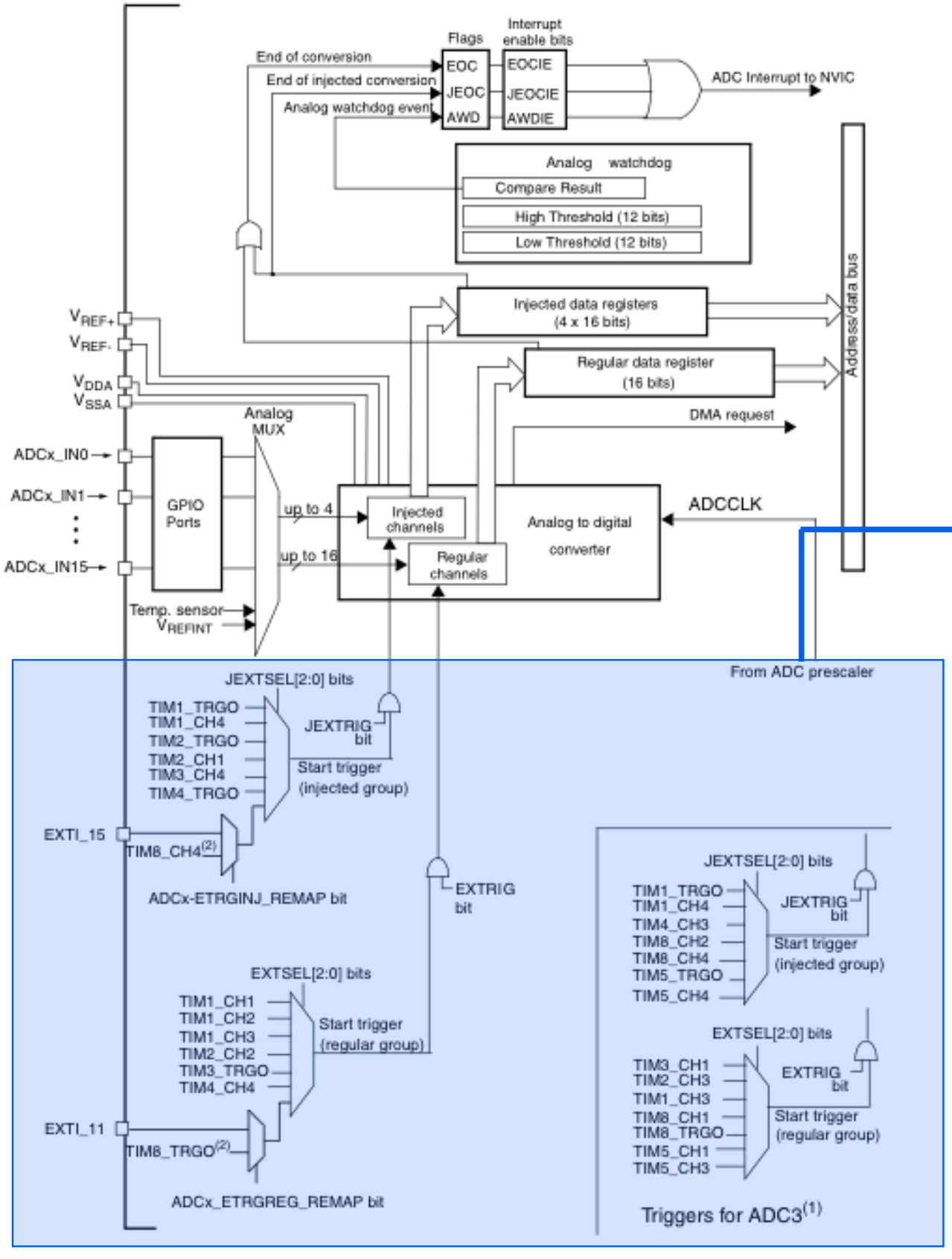



There are **16 multiplexed channels**. It is possible to organize the conversions in two groups: **regular** and **injected**. A group consists of a sequence of conversions which can be done on any channel and in any order.

The **Temperature sensor** is connected to channel ADCx_IN16 and the internal reference voltage VREFINT is connected to ADCx_IN17. These two **internal channels** can be selected and converted as injected or regular channels.

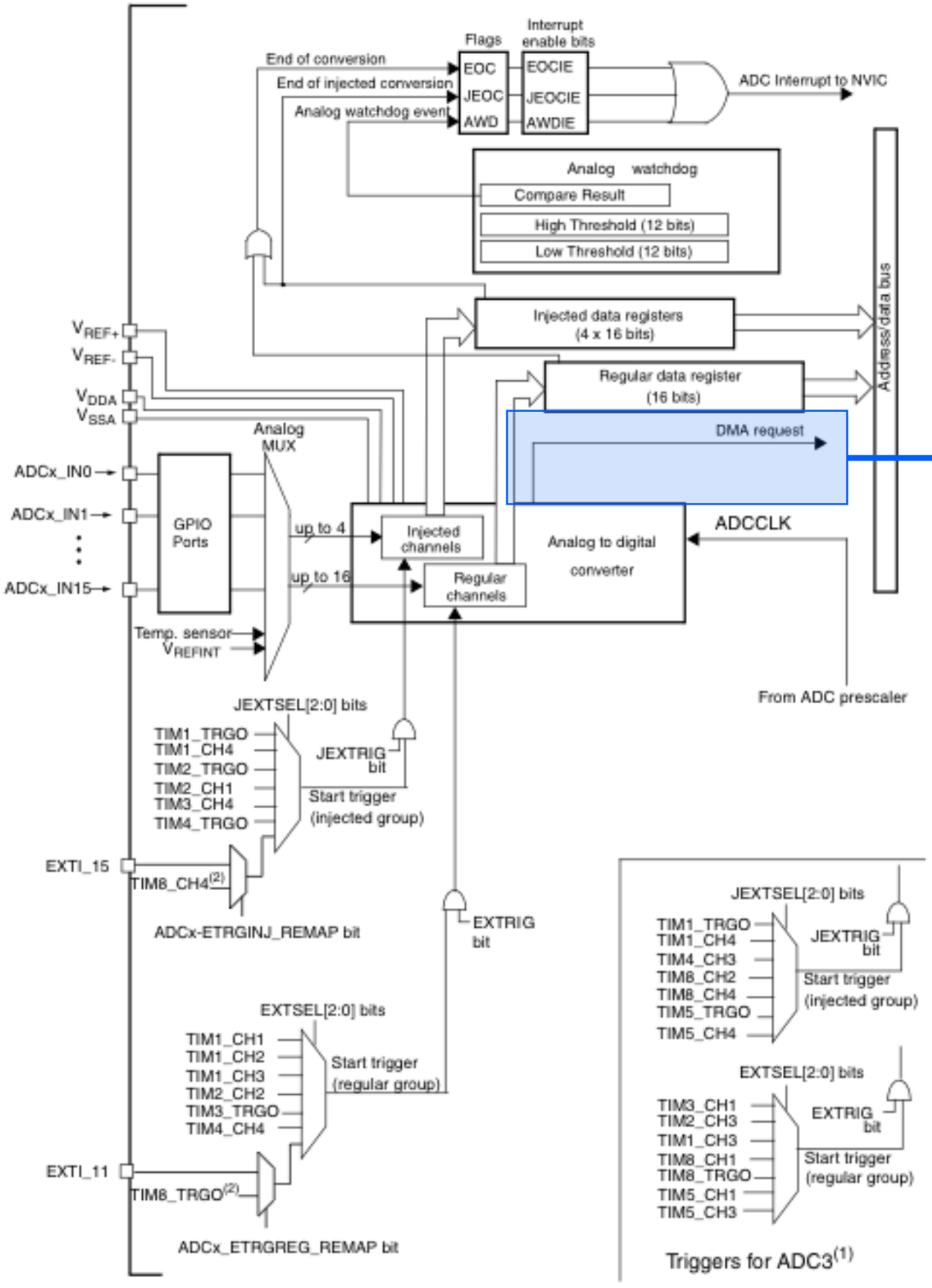
The recommended sampling time for the temperature sensor is 17.1 µs.

# ADC




Conversion can be **triggered** by an **external event** (e.g. timer capture, EXTI line)

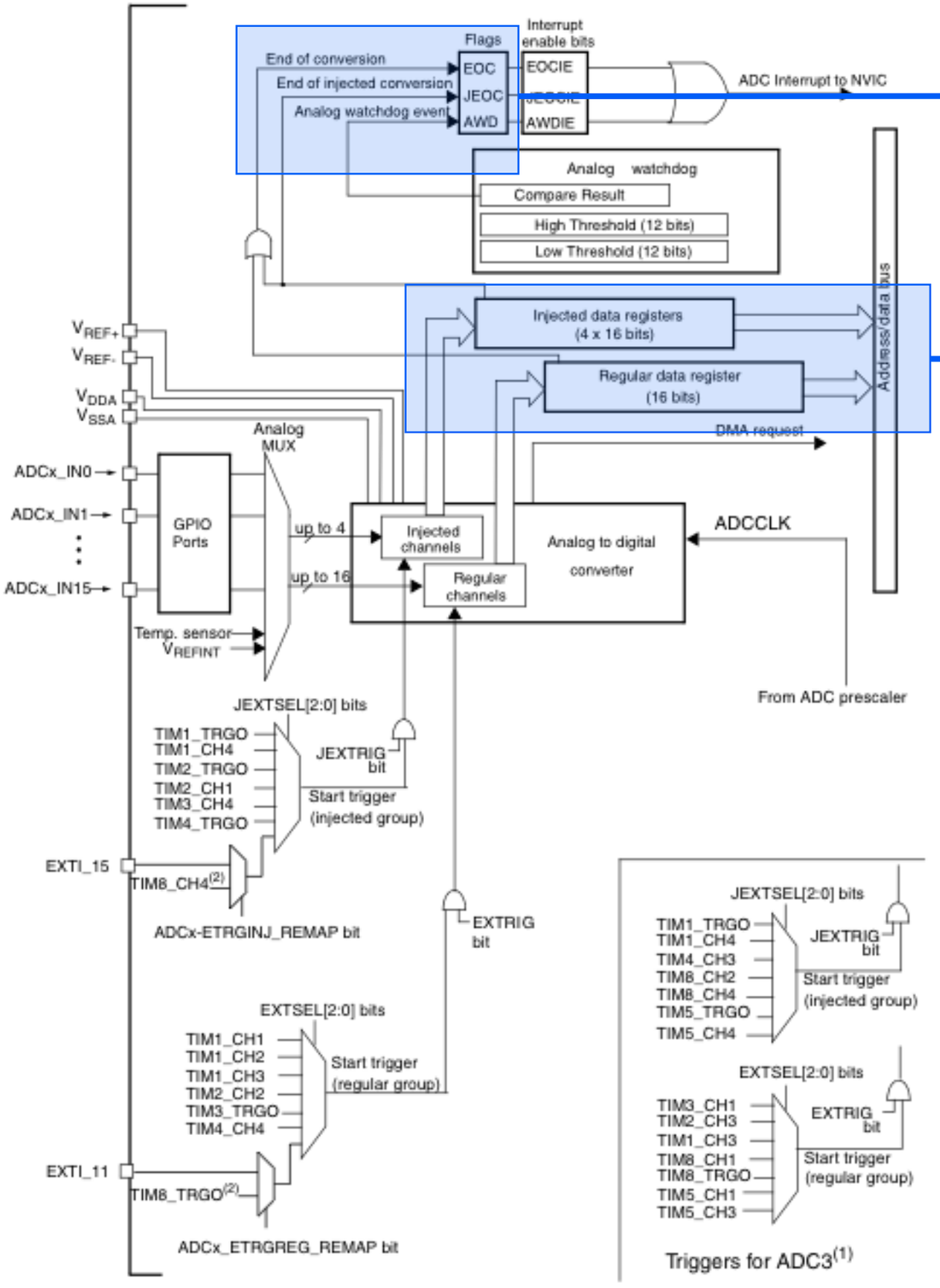
# ADC




Since converted regular channels value are stored in a unique data register, it is necessary to use **DMA** for conversion of more than one regular channel.

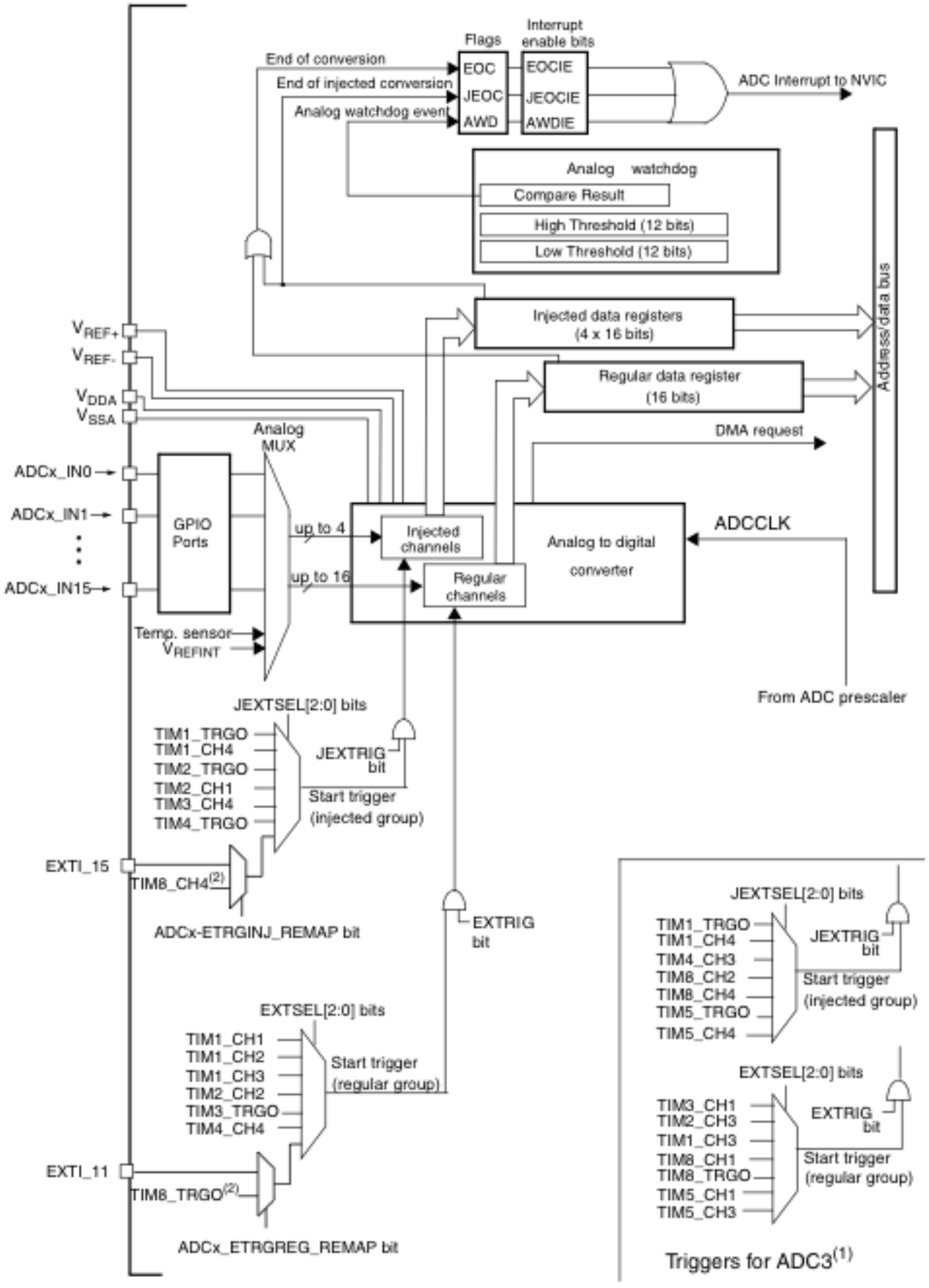
Only the end of conversion of a regular channel generates a **DMA request**, which allows the transfer of its converted data from the ADC_DR register to the destination location selected by the user.

# ADC




When the conversion is done, data is stored in 16 bits **registers** and the proper **flags** are set to indicate the **end of conversion**

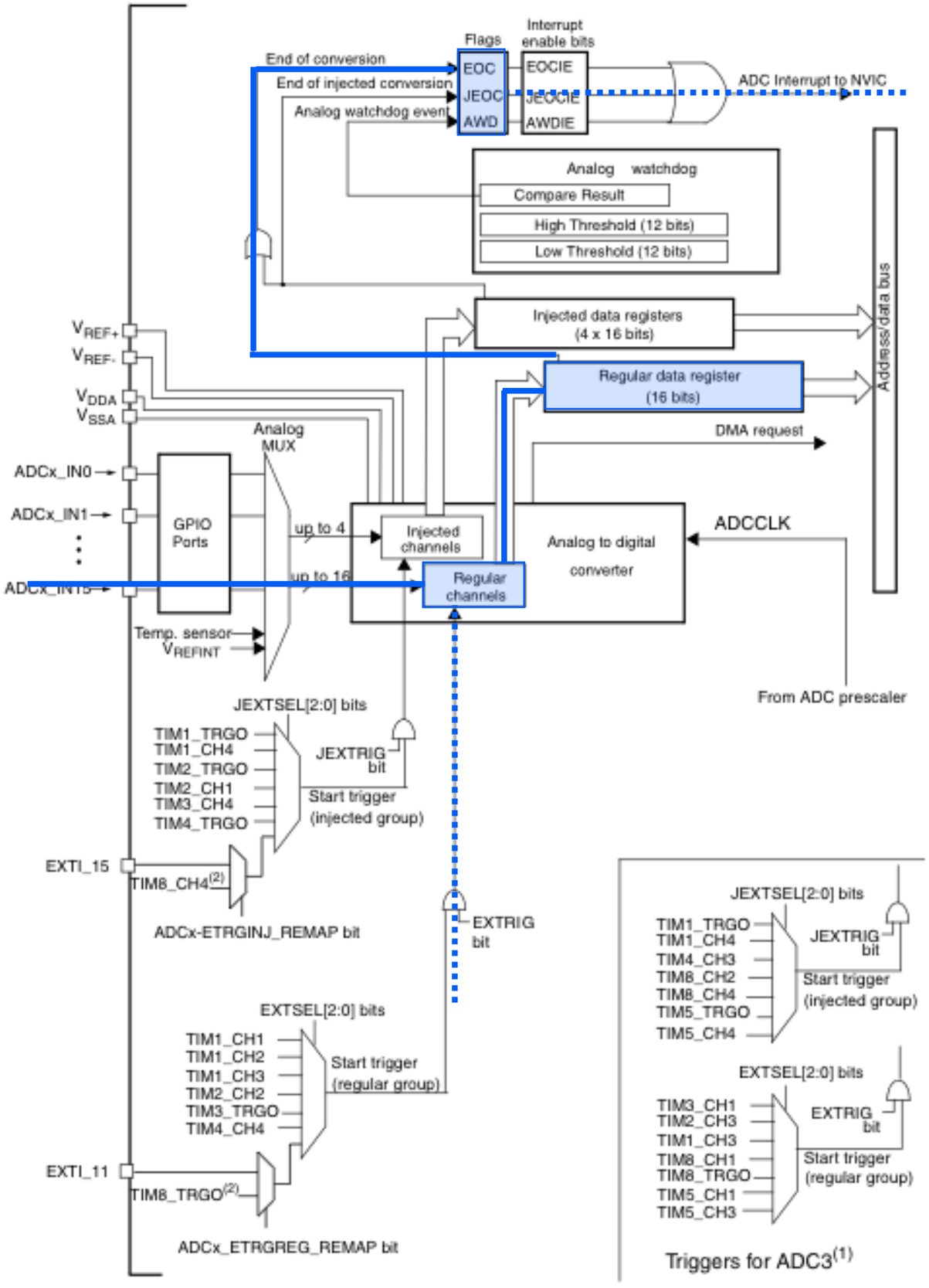
# ADC




**Calibration**

The ADC has an built-in **self calibration mode**. Calibration significantly **reduces accuracy errors** due to internal capacitor bank variations. During calibration, an error-correction code (digital word) is calculated for each capacitor, and during all subsequent conversions, the error contribution of each capacitor is removed using this code.

# ADC



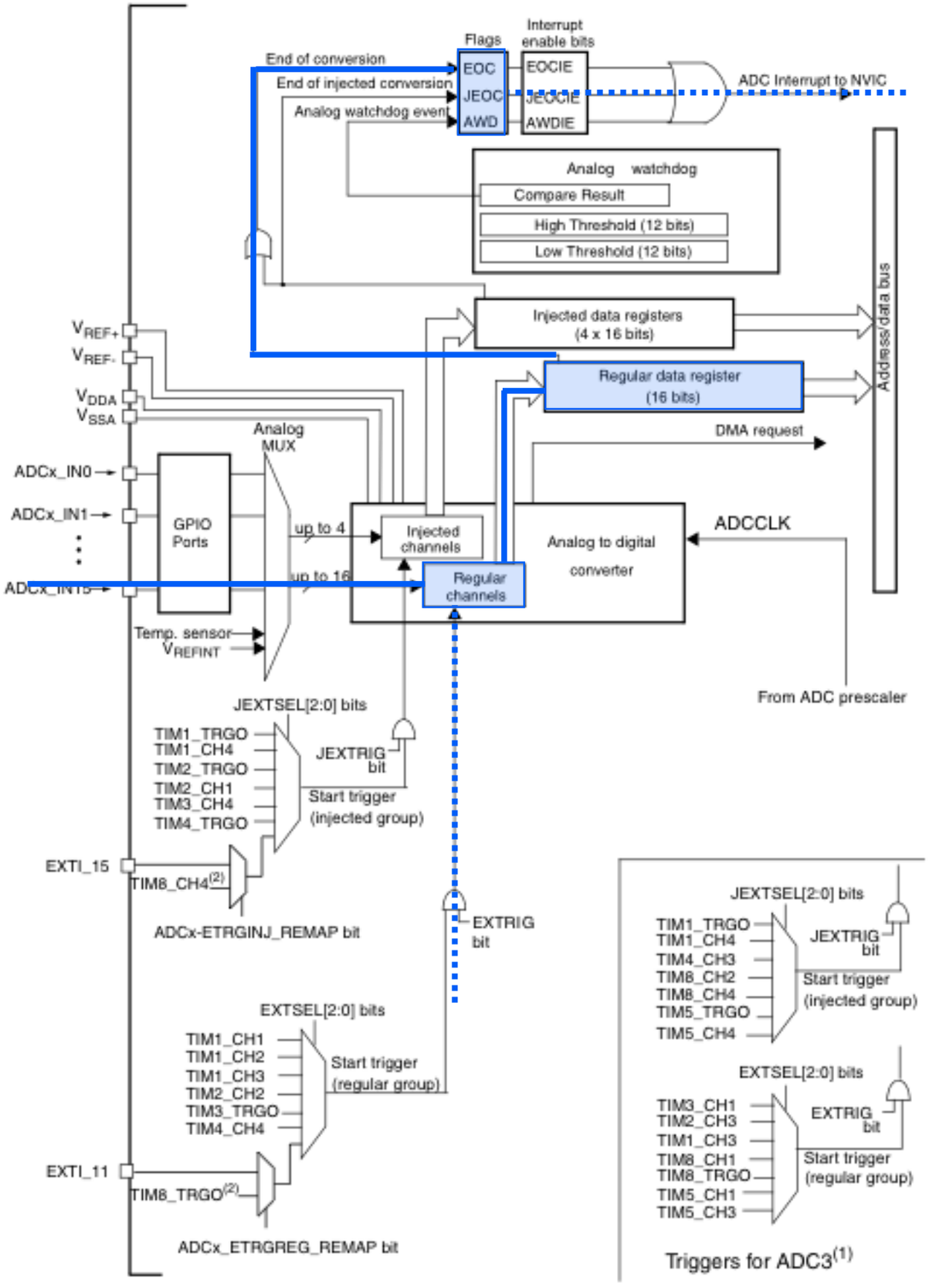

**Conversion Mode (single / continuos)**

1. The converted data is stored in the 16-bit register
2. The EOC (End Of Conversion) flag is set
3. An interrupt is eventually generated

- **<u>Single conversion mode</u>**: in Single conversion mode the ADC does one conversion.

- **<u>Continuous conversion mode</u>**: in continuous conversion mode ADC starts another conversion as soon as it finishes one.

# ADC

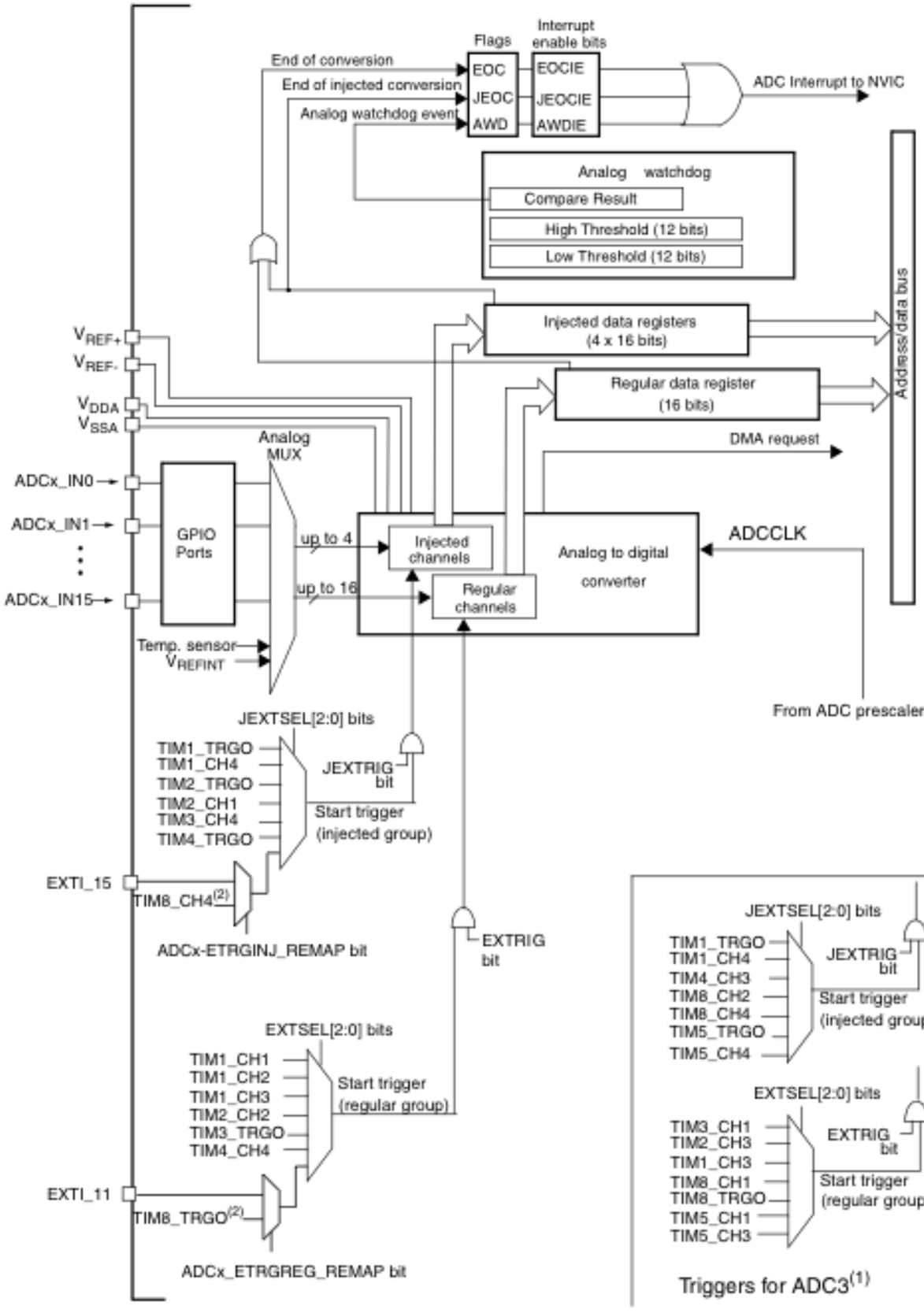



**Conversion Mode (scan / single)**

- **<u>Scan (multichannel)</u>**: This mode is used to scan a group of analog channels. A single conversion is performed for each channel of the group. After each end of conversion the next channel of the group is converted automatically.

- **<u>Single (one channel)</u>**: one single channel is used

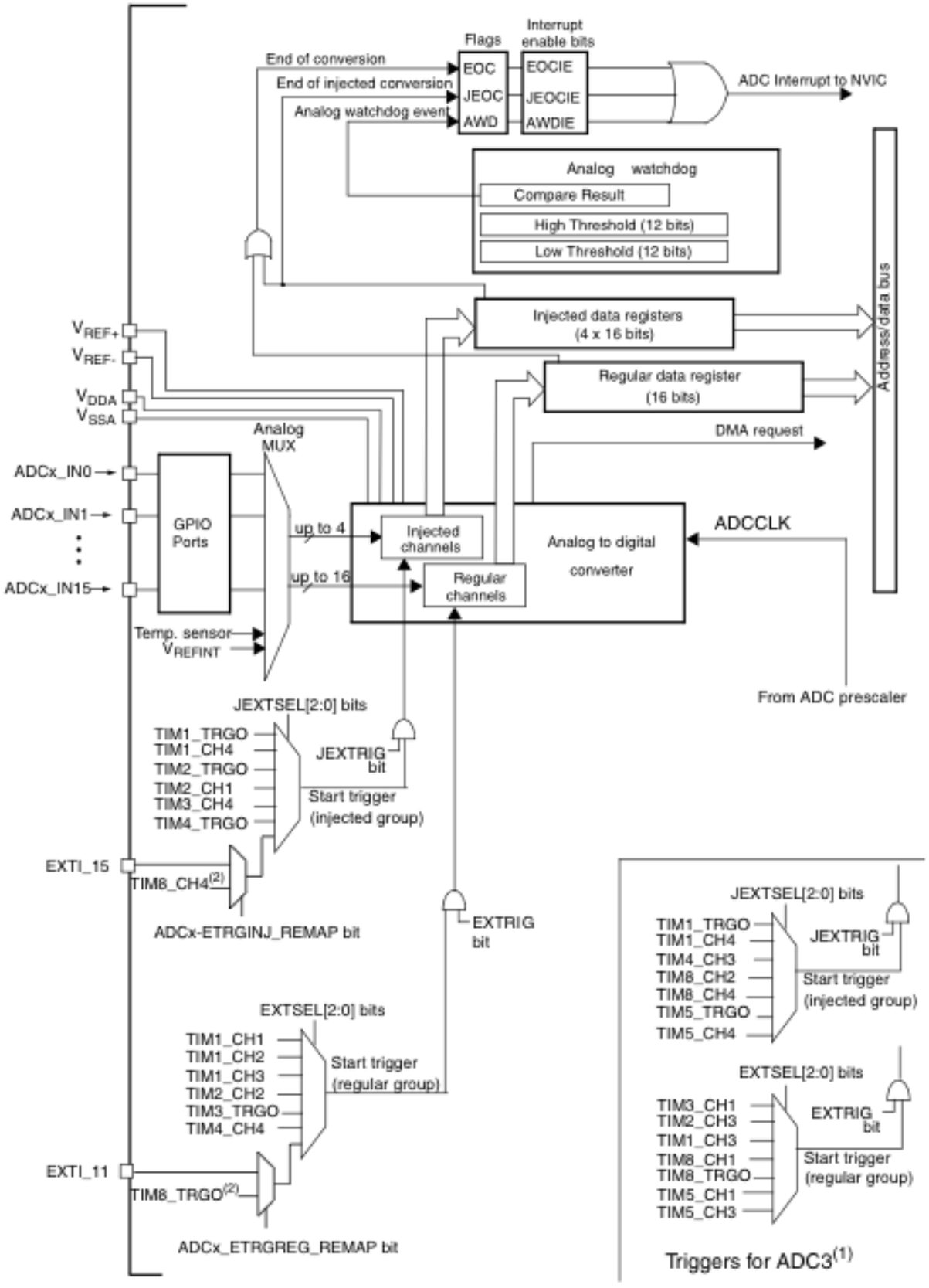
# ADC




**ADC mode (independent / dual)**

- **<u>Independent mode</u>**: one ADC is used
- **<u>Dual mode</u>**: In dual ADC mode the start of conversion is triggered alternately or simultaneously by the ADC1 master to the ADC2 slave

# ADC




**Channel-by-channel programmable sample time**

ADC samples the input voltage for a number of **ADCCLK cycles** which can be modified and each channel can be sampled with a **different sample time**.

The total conversion time is calculated as follows:

**Tconv = Sampling time + 12.5 cycles**

# ADC

- We want to use a **DMA** and **ADC** to sample the temperature from ADCx_IN16 input channel.

  - **Configure and enable DMA**

  1. **Configure and enable ADC**

     1. Configure the prescaler for ADCCLK
     2. Configure and init the ADC (conversion mode, sampling time, etc...)
     3. Calibrate ADC
     4. Starting the acquisition

# ADC

## 1. Configure and enable DMA

Address of the register where the result of the acquisition is saved

```
#define ADC1_DR_Address    ((uint32_t)0x4001244C)
DMA_InitTypeDef DMA_InitStructure;
__IO uint16_t ADCConvertedValue;

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

/* DMA1 channel1 configuration ----------------------------------------------*/
DMA_DeInit(DMA1_Channel1);
DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&ADCConvertedValue;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = 1;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
/* Enable DMA1 channel1 */
DMA_Cmd(DMA1_Channel1, ENABLE);
```

# ADC

## 1. Configure and enable DMA

```
#define ADC1_DR_Address    ((uint32_t)0x4001244C)
DMA_InitTypeDef DMA_InitStructure;
__IO uint16_t ADCConvertedValue;

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

/* DMA1 channel1 configuration ----------------------------------------------*/
DMA_DeInit(DMA1_Channel1);
DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&ADCConvertedValue;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = 1;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
/* Enable DMA1 channel1 */
DMA_Cmd(DMA1_Channel1, ENABLE);
```

We read 16bits from ADC and save 16bits in the ADCConvertedValue variable

## 2. Configure and enable ADC

We set the prescaler to have **ADCCLK==12MHz**

```
ADC_InitTypeDef ADC_InitStructure;

RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
```

ADC in **independent mode**

```
/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

# ADC

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
```

ADC in **continuous scan mode**

```
/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

# ADC

## 2. Configure and enable ADC

```
RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);

/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

no **trigger**

# ADC

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
```

one **single channel**

```
/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
```

```
/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

The recommended **sampling time** for the temperature sensor is **17.1 µs**.

With **ADCCLK==12MHz** we have **12 cycles in 1 µs**.

So the **sampling time in cycles** is 12 * 17.1 == **206 cycles**

We set the sampling time to **239.5 cycles**

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
```

```
/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

Enabling the temperature sensor

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);

/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

Enabling DMA and ADC

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);

/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

**Calibration** phase

## 2. Configure and enable ADC

```
ADC_InitTypeDef ADC_InitStructure;


RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* ADC1 configuration */
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);

/* Enable the temperature sensor */
ADC_TempSensorVrefintCmd(ENABLE);
/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
/* Enable ADC1 reset calibaration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibaration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
/* Start ADC1 Software Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

**Start conversion**

# ADC (**code**)

## stm32f10x_conf.h

...

```
#include "stm32f10x_adc.h"
/* #include "stm32f10x_bkp.h" */
/* #include "stm32f10x_can.h" */
/* #include "stm32f10x_crc.h" */
/* #include "stm32f10x_dac.h" */
/* #include "stm32f10x_dbgmcu.h" */
#include "stm32f10x_dma.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_flash.h"
/* #include "stm32f10x_fsmc.h" */
#include "stm32f10x_gpio.h"
/* #include "stm32f10x_i2c.h" */
/* #include "stm32f10x_iwdg.h" */
#include "stm32f10x_pwr.h"
#include "stm32f10x_rcc.h"
/* #include "stm32f10x_rtc.h" */
/* #include "stm32f10x_sdio.h" */
/* #include "stm32f10x_spi.h" */
#include "stm32f10x_tim.h"
/* #include "stm32f10x_usart.h" */
/* #include "stm32f10x_wwdg.h" */
```

...

# ADC (**code**)

## main.c

```
#include "stm32f10x.h"
#define ADC1_DR_Address    ((uint32_t)0x4001244C)
ADC_InitTypeDef ADC_InitStructure;
DMA_InitTypeDef DMA_InitStructure;
__IO uint16_t ADCConvertedValue;
void RCC_Configuration(void);
void GPIO_Configuration(void);
int main(void)
{
      RCC_Configuration();
      /* DMA1 channel1 configuration ----------------------------------------------*/
      DMA_DeInit(DMA1_Channel1);
      DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
      DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&ADCConvertedValue;
      DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
      DMA_InitStructure.DMA_BufferSize = 1;
      DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
      DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
      DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
      DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
      DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
      DMA_InitStructure.DMA_Priority = DMA_Priority_High;
      DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
      DMA_Init(DMA1_Channel1, &DMA_InitStructure);
      /* Enable DMA1 channel1 */
      DMA_Cmd(DMA1_Channel1, ENABLE);
```

...

# ADC (**code**)

## main.c

```
...
    /* ADC1 configuration ------------------------------------------------------*/
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);
    /* ADC1 regular channel14 configuration */
    ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
    ADC_TempSensorVrefintCmd(ENABLE);
    /* Enable ADC1 DMA */
    ADC_DMACmd(ADC1, ENABLE);
    /* Enable ADC1 */
    ADC_Cmd(ADC1, ENABLE);
    /* Enable ADC1 reset calibaration register */
    ADC_ResetCalibration(ADC1);
    /* Check the end of ADC1 reset calibration register */
    while(ADC_GetResetCalibrationStatus(ADC1));
    /* Start ADC1 calibaration */
    ADC_StartCalibration(ADC1);
    /* Check the end of ADC1 calibration */
    while(ADC_GetCalibrationStatus(ADC1));
    /* Start ADC1 Software Conversion */
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    while (1);
}
```

# ADC (**code**)

## main.c

```
/**
    * @brief  Configures the different system clocks.
    * @param  None
    * @retval None
*/
void RCC_Configuration(void)
{
      /* ADCCLK = PCLK2/2 */
      RCC_ADCCLKConfig(RCC_PCLK2_Div2);
      /* Enable peripheral clocks ------------------------------------------------*/
      /* Enable DMA1 clock */
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
      /* Enable ADC1 and GPIOC clock */
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
      }
```

# ADC (**exercises and questions**)

1. **What the read value is? What does it mean?**

2. **Search the conversion formula in the manual. How many degrees are we measuring? How is it possible?**

2. **Write a program to monitor the temperature for 10 min with a sampling time of 30 sec, writing data in SRAM memory using DMA.**

3. **Write a program to turn on a LED if the temperature is above a certain threshold and turn off again if the temperature falls below the threshold.**

**Table 64. TS characteristics**

| Symbol | Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| $T_L$ | $V_{SENSE}$ linearity with temperature | | ±1 | ±2 | °C |
| Avg_Slope | Average slope | 4.0 | 4.3 | 4.6 | mV/°C |
| $V_{25}$ | Voltage at 25 °C | 1.34 | 1.43 | 1.52 | V |
| $t_{START}$(1) | Startup time | 4 | | 10 | µs |
| $T_{S\_temp}$(2)(1) | ADC sampling time when reading the temperature | | | 17.1 | µs |

1. Guaranteed by design, not tested in production.
2. Shortest sampling time can be determined in the application by multiple iterations.

# #9 Project

int sprintf ( char * str, const char * format, ... );

# Project

The project consist in writing the code for a device able to transmit maximum and average value of a waveform connected to a pin of the microcontroller

Project description:

- When the STM32 Discovery is turned on the green LED starts blinking at 2 Hz.
- When the user presses the button the signal is sampled using the ADC and the data is stored in SRAM memory buffer using DMA.
- The memory buffer is used to compute maximum, minimum and average value
- The blinking frequency of the blue LED is set according to signal amplitude (peak to peak value, use maixmum – minimum) when amplitude is below 100 mv blink frequency is 0.1Hz, when above 3V frequency is 20Hz
- If the user press again the button the second LED is turned off and the acquisition is stopped.
- Each time the maximum, minimum and average value are extracted, a string containing data is sent trough serial port.

Tips and project requirements:

- Convert the ADC value in Volt and use “sprintf ( char * str, const char * format, ... );” to convert ADC value to a string
- The memory buffer has to be the size of 100 elements, chose arbitrary waveform, with a frequency around kHz
- The acquisition can be performed using the channel14 of ADC1 (corresponding to the Pin4 of the GPIOC that has to be configured in INPUT mode)