ADC Tutorial

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 19 multiplexed channels. The A/D conversion of the channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored into a leftor right-aligned 16-bit data register.

ADC main features

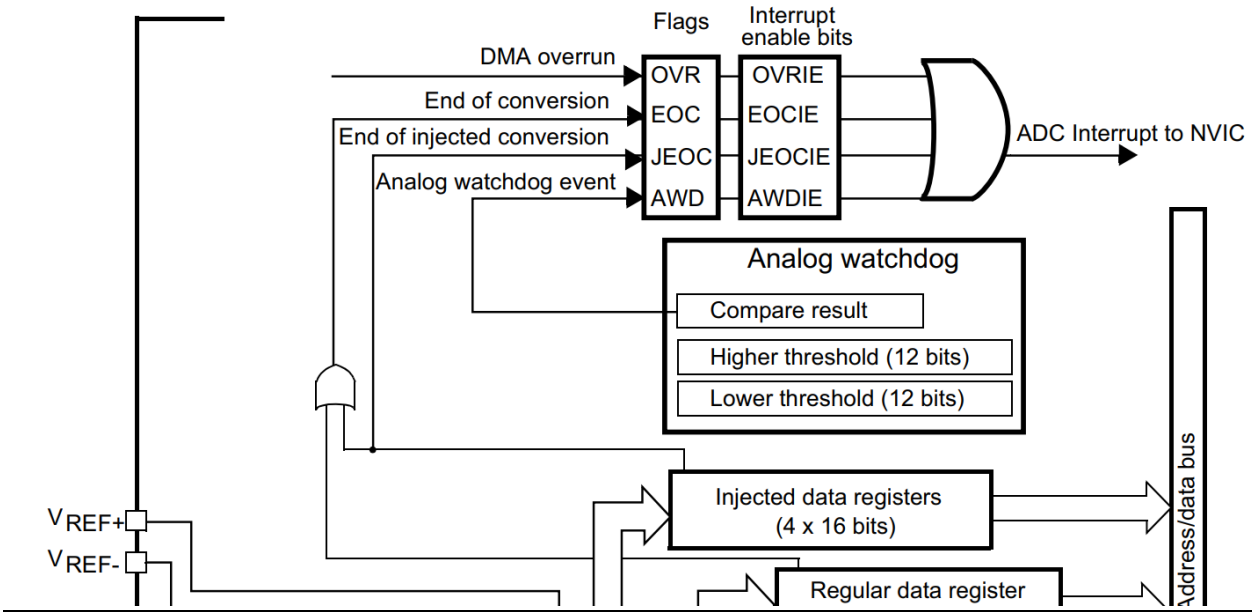
12-bit, 10-bit, 8-bit or 6-bit configurable resolution

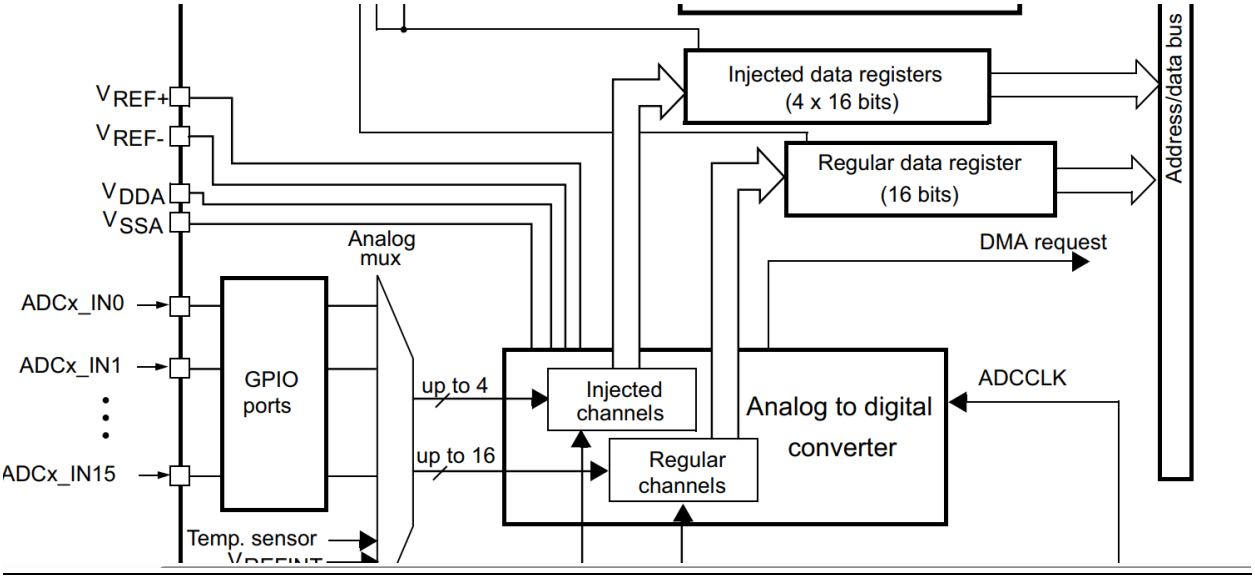
Interrupt generation at the end of conversion, end of injected conversion, and in case of analog watchdog or overrun events

Single and continuous conversion modes

Scan mode for automatic conversion of channel 0 to channel ‘n’ • Data alignment with in-built data coherency

Channel-wise programmable sampling time





ADC can take analog voltage given from sensors and convert it into its digital equivalent based on the following formula.

We are going to configure the ADC1 with interrupt mode and take the services of NVIC also in this tutorial.

The channels we are going to take for ADC sampling is PA0 in analog mode.

The following steps can be considered for ADC initialization and can be clubbed in ADC_Init() function.

1. Enable PA0 in analog mode by giving GPIOA->MODER |=0x03;
2. Enable the clock the for ADC1 on its relevant APB2ENR.
3. These bits on ADC1->SQR1 is made zero which are written by software to define the total number of conversions in the regular channel conversion sequence.
4. Make the ADC ON on ADC_CR2 register.

The following steps can be clubbed to enable interrupt for ADC and name it as ADC_EN_Interrupt().

1. Enable EOC(end of conversion) interrupt enable to enable the end of conversion interrupt on `ADC1->CR1`.
2. Enable the NVIC channel for this ADC1 by giving `NVIC_EnableIRQ(ADC_IRQn).`

The following steps can be clubbed together to give commands to ADC1 to start the conversion and name it as ADC_SOC();

1. Enable CONTINUOUS bit on ADC->CR2;
2. Enable SWSTART bit on ADC->CR2

The following steps can be clubbed together to read ADC values

1. Read EOC bit on ADC1->SR register.
2. Once it is high go to collect data from ADC1->DR register into a variable

Implement a ADC_IRQHandler having the following things in it.

1. Check for the arrival of EOC signal on ADC1->SR as follows
   `if((ADC1->SR & ADC_SR_EOC) != 0)`
2. Clear the EOC flag.
3. Collect the converted value in a variable
4. Print it thru printf
5. Call some for loop delay
6. End of the ADC handler.

So my main() would look as follows

Int main()

{

```
      clockSpeed_PLL();

    UART2_Init(); // this is a function for USART2_Transmit import it
from USART Tutorial for reference.
    adc_init(); //
    ADC_EN_Interrupt(); // this function is for enabling interrupt
    ADC_SOC();

    while(1){

    }
}

int __io_putchar(int ch){
    tx_send(ch); // USART2_Transmit function
    for(uint32_t i=0;i<1600000;i++);
    return ch;
}
```