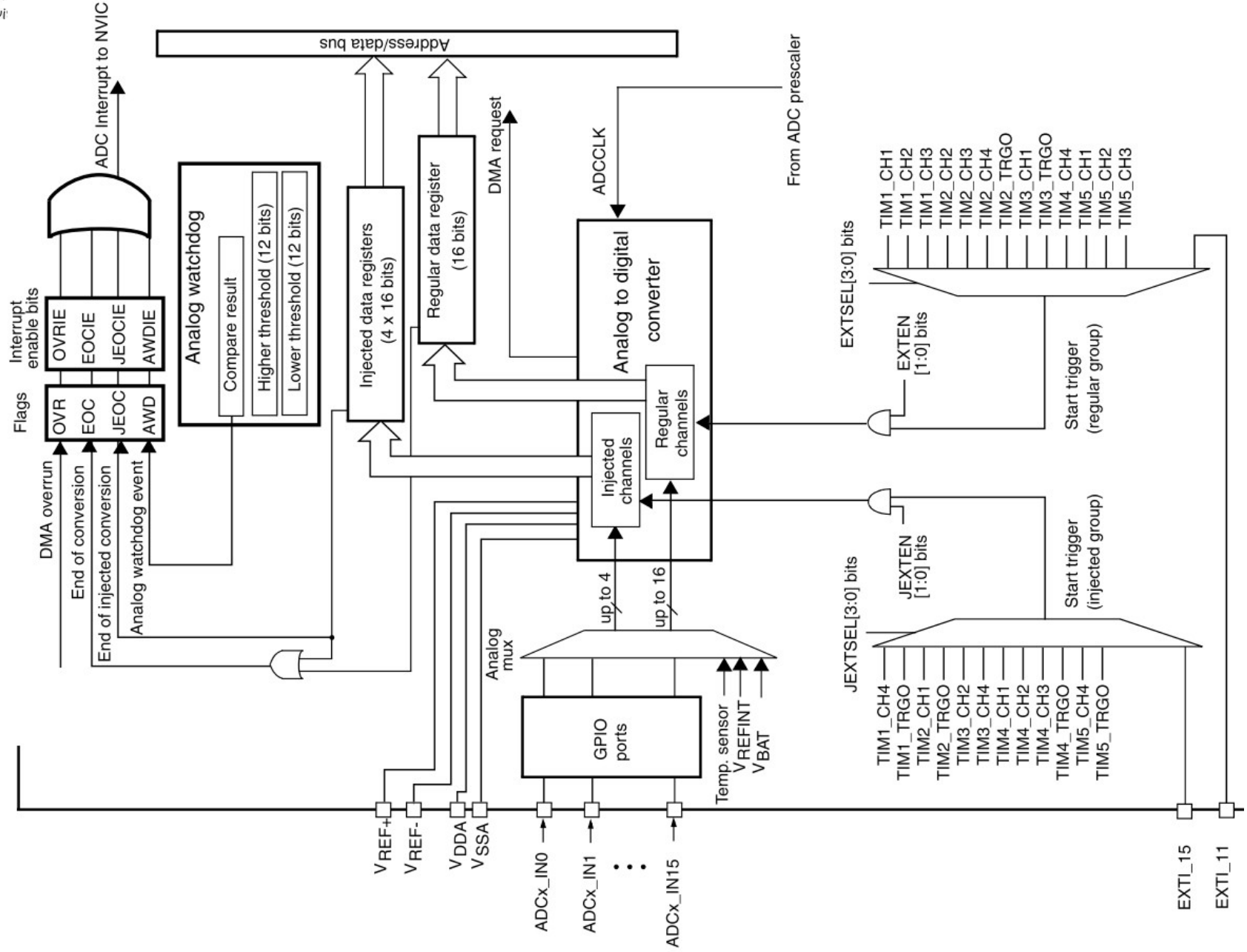
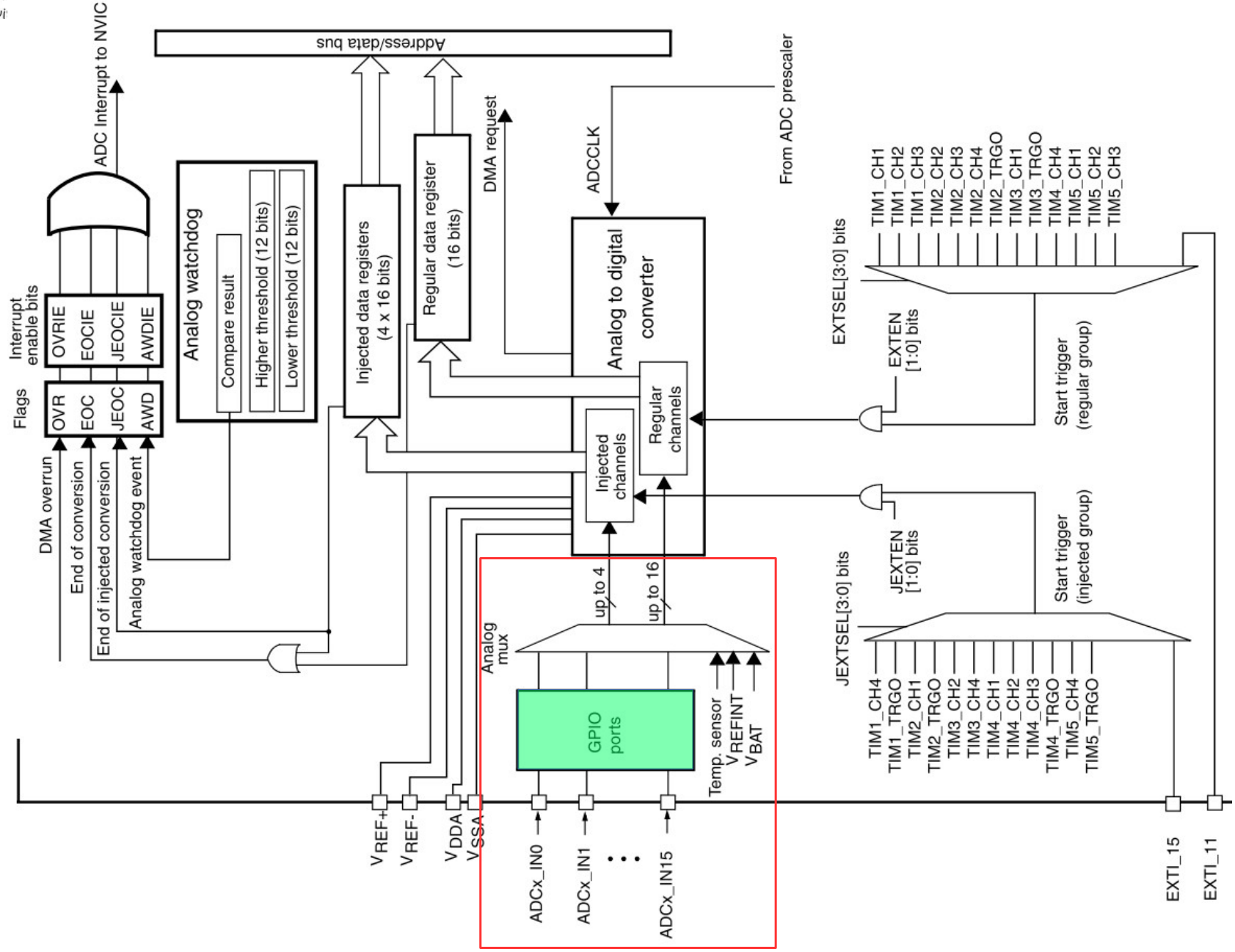
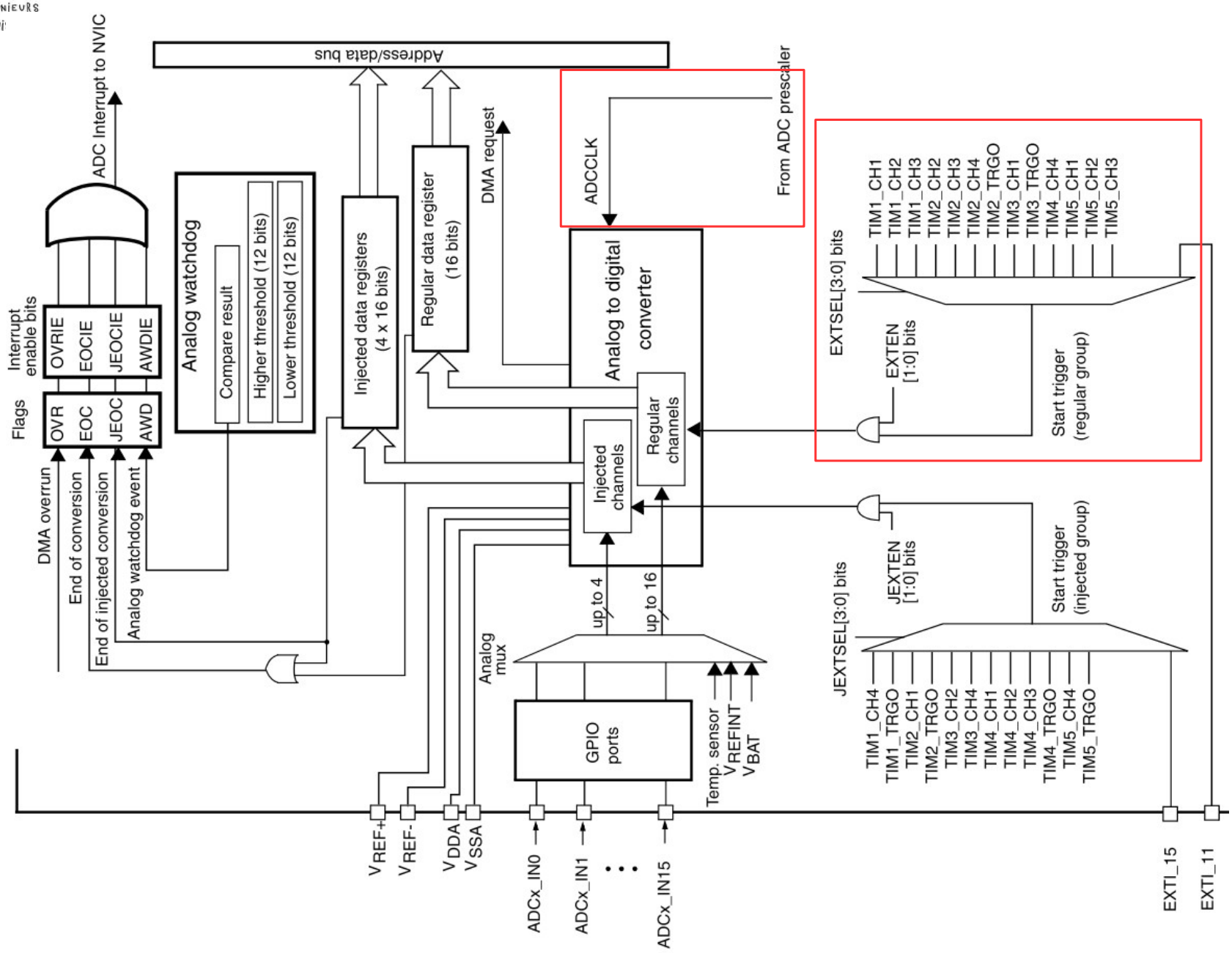
Annexe TD ADC



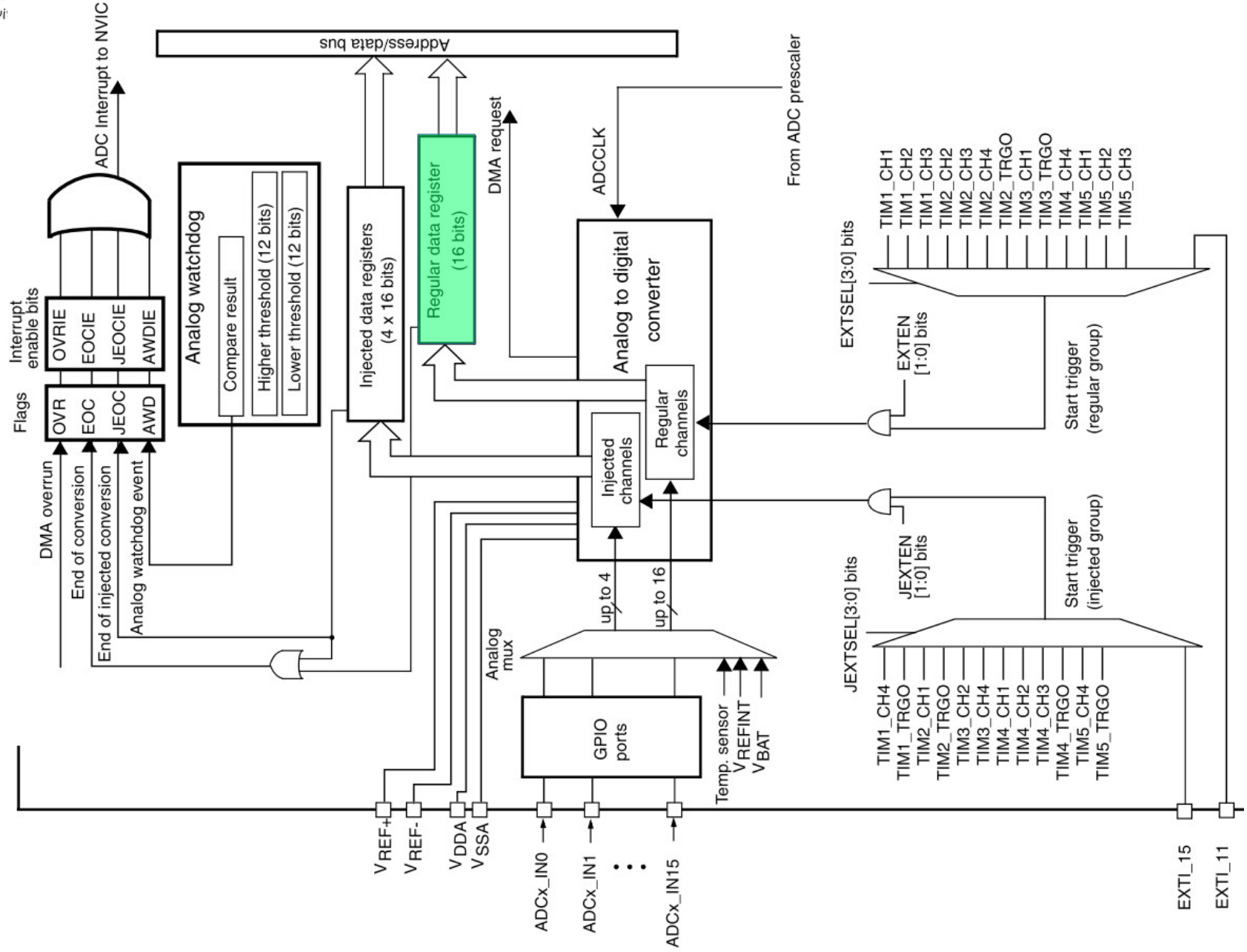
Annexe TD ADC



Annexe TD ADC



Annexe TD ADC



Annexe TD ADC

```
int adc_init(ADC_t *adc, uint32_t mode, OnSample cb)
{
    IRQn_t    irq_number;
    uint32_t  irq_priority;

    if (adc == _ADC1) {



        // configure pins
        io_configure(ADC1_GPIO_PORT, ADC1_GPIO_PINS, PIN_MODE_ANALOG, NULL);

        irq_number = ADC_IRQn;
        irq_priority = ADC1_IRQ_PRIORITY;
    } else { return -1; }

    for (int presc_cfg = 0; presc_cfg ≤ 3; presc_cfg++) {
        // set up ADCCLK prescaler config (2,4,6 or 8) so that
        // Fadcclk < 30MHz
        if (sysclks.apb2_freq / (2*(presc_cfg+1)) < ADCCLK_MAX) {









            // return ADCCLK
            adc_freq = sysclks.apb2_freq / (2*(presc_cfg + 1));
            return (int)adc_freq;
        }
    }

    return -1;
}
```

Annexe TD ADC

```
int adc_init(ADC_t *adc, uint32_t mode, OnSample cb)
{
    IRQn_t    irq_number;
    uint32_t  irq_priority;

    if (adc ==  ADC1) {
        // enable ADC clocking
        _RCC->APB2ENR |= 1<<8;

        // configure pins
        io_configure(ADC1_GPIO_PORT, ADC1_GPIO_PINS, PIN_MODE_ANALOG, NULL);

        irq_number = ADC_IRQn;
        irq_priority = ADC1_IRQ_PRIORITY;
    } else { return -1; }

    for (int presc_cfg = 0; presc_cfg ≤ 3; presc_cfg++) {
        // set up ADCCLK prescaler config (2,4,6 or 8) so that
        // Fadcclk < 30MHz
        if (sysclks.apb2_freq / (2*(presc_cfg+1)) < ADCCLK_MAX) {









            // return ADCCLK
            adc_freq = sysclks.apb2_freq / (2*(presc_cfg + 1));
            return (int)adc_freq;
        }
    }

    return -1;
}
```

Annexe TD ADC

```
int adc_init(ADC_t *adc, uint32_t mode, OnSample cb)
{
    IRQn_t    irq_number;
    uint32_t  irq_priority;

    if (adc == _ADC1) {
        // enable ADC clocking
        _RCC->APB2ENR |= 1<<8;

        // configure pins
        io_configure(ADC1_GPIO_PORT, ADC1_GPIO_PINS, PIN_MODE_ANALOG, NULL);

        irq_number = ADC_IRQn;
        irq_priority = ADC1_IRQ_PRIORITY;
    } else { return -1; }

    for (int presc_cfg = 0; presc_cfg ≤ 3; presc_cfg++) {
        // set up ADCCLK prescaler config (2,4,6 or 8) so that
        // Fadcclk < 30MHz
        if (sysclks.apb2_freq / (2*(presc_cfg+1)) < ADCCLK_MAX) {
            ADC->CCR = presc_cfg << 16;




            // return ADCCLK
            adc_freq = sysclks.apb2_freq / (2*(presc_cfg + 1));
            return (int)adc_freq;
        }
    }

    return -1;
}
```

Annexe TD ADC

```
int adc_init(ADC_t *adc, uint32_t mode, OnSample cb)
{
    IRQn_t    irq_number;
    uint32_t  irq_priority;

    if (adc == _ADC1) {
        // enable ADC clocking
        _RCC->APB2ENR |= 1<<8;

        // configure pins
        io_configure(ADC1_GPIO_PORT, ADC1_GPIO_PINS, PIN_MODE_ANALOG, NULL);

        irq_number = ADC_IRQn;
        irq_priority = ADC1_IRQ_PRIORITY;
    } else { return -1; }

    for (int presc_cfg = 0; presc_cfg ≤ 3; presc_cfg++) {
        // set up ADCCLK prescaler config (2,4,6 or 8) so that
        // Fadcclk < 30MHz
        if (sysclks.apb2_freq / (2*(presc_cfg+1)) < ADCCLK_MAX) {
            ADC->CCR = presc_cfg << 16;

            // Scan mode (1<<8) : convert inputs selected in ADC_SQRx
            adc->CR1 = 1 << 8 | (mode & (3 << 24));



            // return ADCCLK
            adc_freq = sysclks.apb2_freq / (2*(presc_cfg + 1));
            return (int)adc_freq;
        }
    }

    return -1;
}
```

Annexe TD ADC

```
int adc_init(ADC_t *adc, uint32_t mode, OnSample cb)
{
    IRQn_t    irq_number;
    uint32_t  irq_priority;

    if (adc == _ADC1) {
        // enable ADC clocking
        _RCC->APB2ENR |= 1<<8;

        // configure pins
        io_configure(ADC1_GPIO_PORT, ADC1_GPIO_PINS, PIN_MODE_ANALOG, NULL);

        irq_number = ADC_IRQn;
        irq_priority = ADC1_IRQ_PRIORITY;
    } else { return -1; }

    for (int presc_cfg = 0; presc_cfg ≤ 3; presc_cfg++) {
        // set up ADCCLK prescaler config (2,4,6 or 8) so that
        // Fadcclk < 30MHz
        if (sysclks.apb2_freq / (2*(presc_cfg+1)) < ADCCLK_MAX) {
            ADC->CCR = presc_cfg << 16;

            // Scan mode (1<<8) : convert inputs selected in ADC_SQRx
            adc->CR1 = 1 << 8 | (mode & (3 << 24));

            // EOCS (1<<10) : bit set (and interrupt requested) after each channel conversion
            // ADON (1<<0)  : ADC on
            adc->CR2 = 1<<10 | 1;

            // return ADCCLK
            adc_freq = sysclks.apb2_freq / (2*(presc_cfg + 1));
            return (int)adc_freq;
        }
    }

    return -1;
}
```

Annexe TD ADC

```
#define ADC_CHANNEL_TEMP                        16
#define ADC_CHANNEL_VREFINT                     17
#define ADC_CHANNEL_VBAT                        18


uint16_t adc_channel_sample(ADC_t *adc, uint32_t channel)
{
    if (channel > 18) return 0;

    n_to_convert = 1;

    /* special internal channels */
    if (channel == ADC_CHANNEL_TEMP) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VREFINT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VBAT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (1<<22);
    }









}
```

Annexe TD ADC

```
#define ADC_CHANNEL_TEMP                        16
#define ADC_CHANNEL_VREFINT                     17
#define ADC_CHANNEL_VBAT                        18


uint16_t adc_channel_sample(ADC_t *adc, uint32_t channel)
{
    if (channel > 18) return 0;

    n_to_convert = 1;

    /* special internal channels */
    if (channel == ADC_CHANNEL_TEMP) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VREFINT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VBAT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (1<<22);
    }

    adc->SQR[0] = (n_to_convert - 1) << 20;

    adc->SQR[1] = 0;

    adc->SQR[2] = channel;




}
```

Annexe TD ADC

```
#define ADC_CHANNEL_TEMP                        16
#define ADC_CHANNEL_VREFINT                     17
#define ADC_CHANNEL_VBAT                        18


uint16_t adc_channel_sample(ADC_t *adc, uint32_t channel)
{
    if (channel > 18) return 0;

    n_to_convert = 1;

    /* special internal channels */
    if (channel == ADC_CHANNEL_TEMP) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VREFINT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VBAT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (1<<22);
    }

    adc->SQR[0] = (n_to_convert - 1) << 20;

    adc->SQR[1] = 0;

    adc->SQR[2] = channel;

    adc->CR2 |= 1<<30;



}
```

Annexe TD ADC

```
#define ADC_CHANNEL_TEMP                        16
#define ADC_CHANNEL_VREFINT                     17
#define ADC_CHANNEL_VBAT                        18


uint16_t adc_channel_sample(ADC_t *adc, uint32_t channel)
{
    if (channel > 18) return 0;

    n_to_convert = 1;

    /* special internal channels */
    if (channel == ADC_CHANNEL_TEMP) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VREFINT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VBAT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (1<<22);
    }

    adc->SQR[0] = (n_to_convert - 1) << 20;

    adc->SQR[1] = 0;

    adc->SQR[2] = channel;

    adc->CR2 |= 1<<30;

    while (!(adc->SR & ADC_SR_EOC));

}
```

Annexe TD ADC

```
#define ADC_CHANNEL_TEMP                        16
#define ADC_CHANNEL_VREFINT                     17
#define ADC_CHANNEL_VBAT                        18


uint16_t adc_channel_sample(ADC_t *adc, uint32_t channel)
{
    if (channel > 18) return 0;

    n_to_convert = 1;

    /* special internal channels */
    if (channel == ADC_CHANNEL_TEMP) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VREFINT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (2<<22);
    } else if (channel == ADC_CHANNEL_VBAT) {
        ADC->CCR = (ADC->CCR & ~(3<<22)) | (1<<22);
    }

    adc->SQR[0] = (n_to_convert - 1) << 20;

    adc->SQR[1] = 0;

    adc->SQR[2] = channel;

    adc->CR2 |= 1<<30;


    while (!(adc->SR & ADC_SR_EOC));

    return (uint32_t)(adc->DR);
}
```