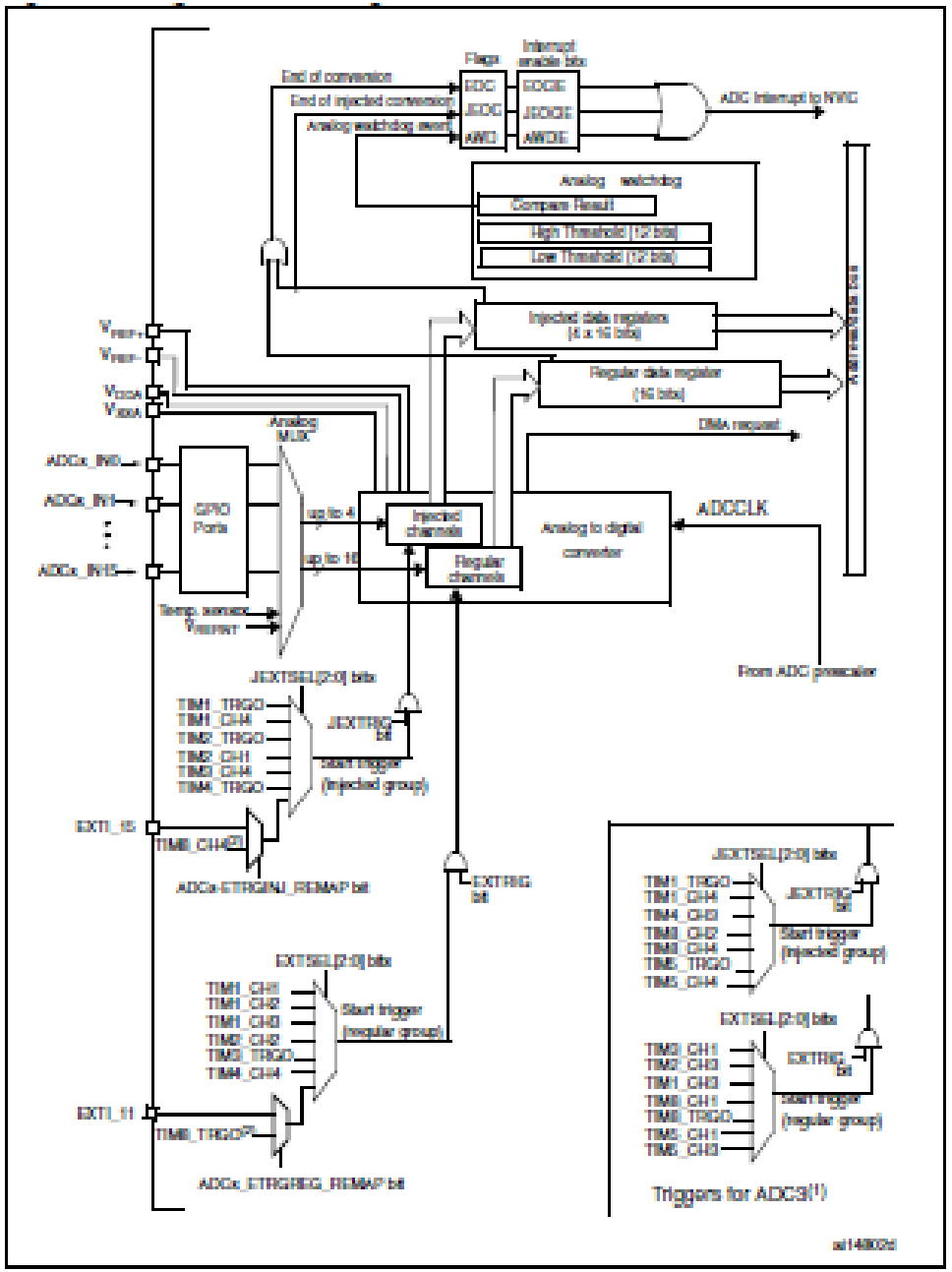
Analog to Digital Converter(ADC)
and
Digital to Analog Converter (DAC)

ARM Cortex ADC support

- 12 bit ADC is successive approximation
- 18 multiplexed channel (16 external and 2 internal)
- The result is stored in a left-aligned or right aligned 16 bit register
- Analog watchdog feature allows the application to detect if input voltage goes outside user defined high or low threshold
- ADC input clock is generated from PCLK2

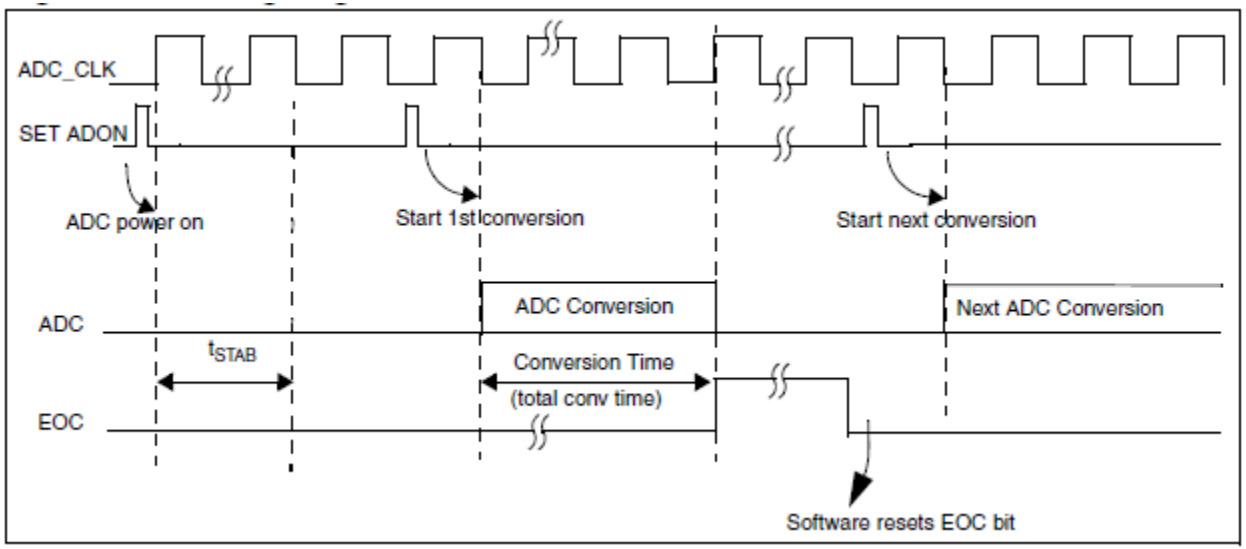
Single ADC block diagram



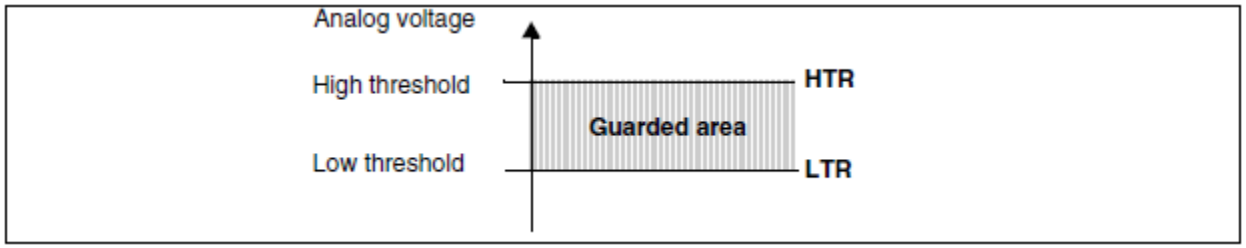
ADC pins

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, $2.4\ V \le V_{REF+} \le V_{DDA}$
V_{DDA} (1)	Input, analog supply	Analog power supply equal to V_{DD} and $2.4\ V \le V_{DDA} \le 3.6\ V$
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, $V_{REF-} = V_{SSA}$
V_{SSA} (1)	Input, analog supply ground	Ground for analog power supply equal to V_{SS}
ADCx_IN[15:0]	Analog signals	16 analog channels

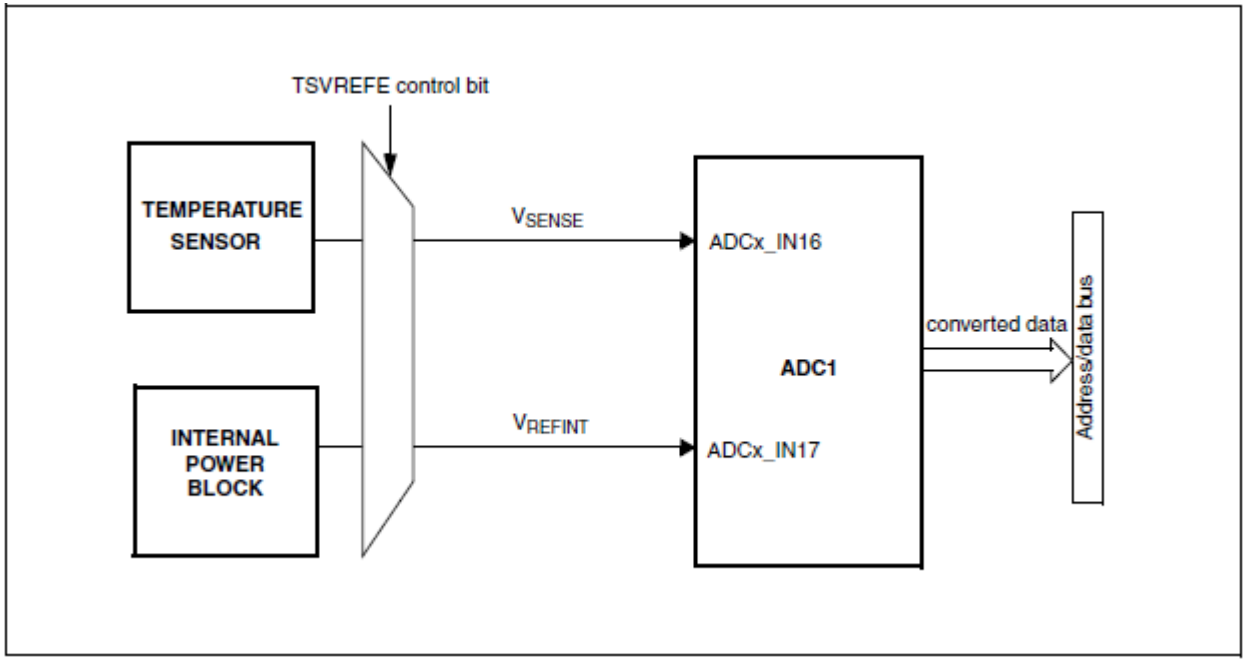
Timing diagram



Analog watchdog



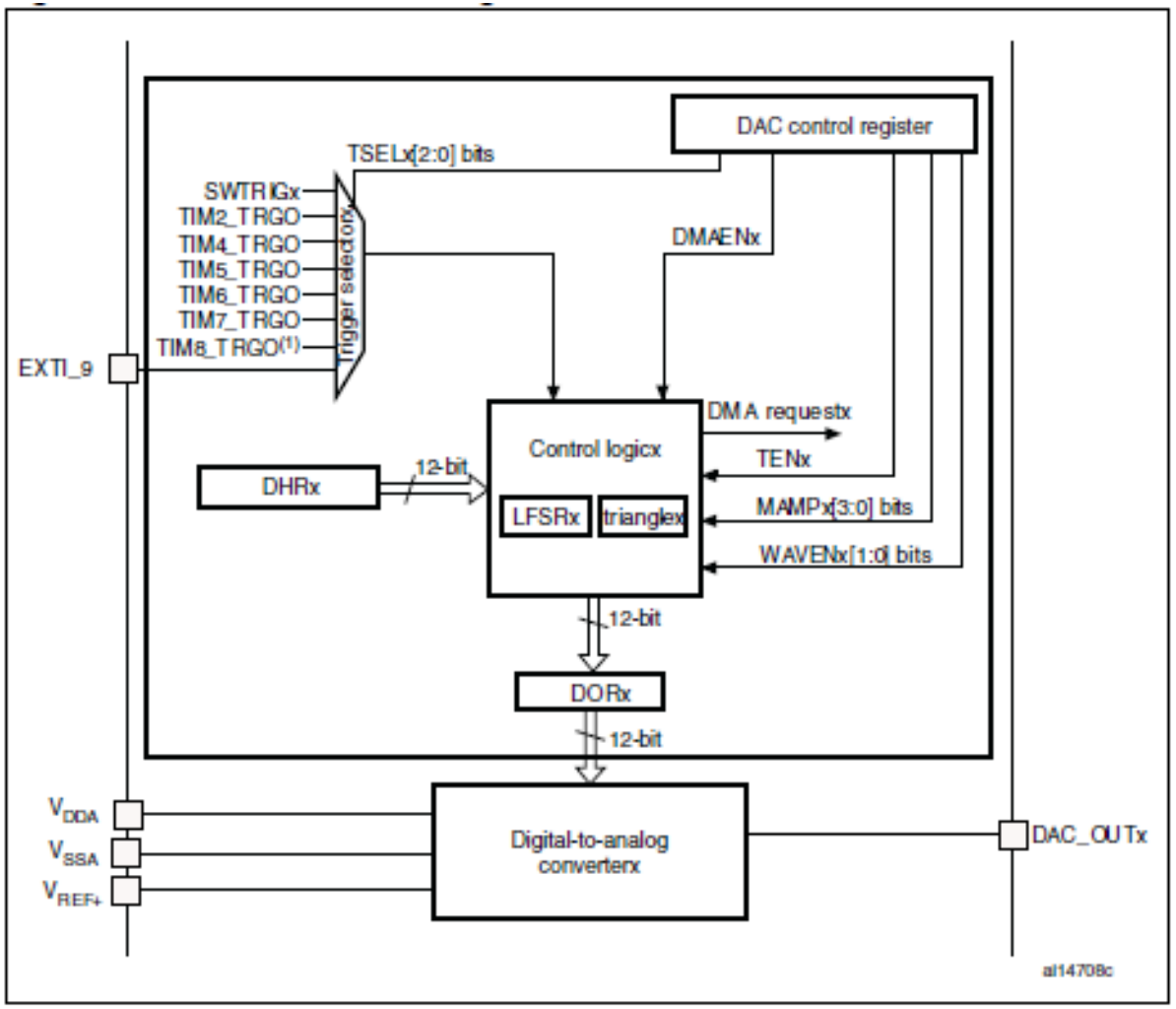
Temperature sensor and Vref diagram



Digital to Analog converter

- Can be configured to 8 or 12 bit mode
- The data could be left or right aligned
- DAC has two output channels
  - Both output can be independently or simultaneously

DAC channel block diagram



DAC pins

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the DAC, $2.4\ V \leq V_{REF+} \leq V_{DDA}$ (3.3 V)
V_{DDA}	Input, analog supply	Analog power supply
V_{SSA}	Input, analog supply ground	Ground for analog power supply
DAC_OUTx	Analog output signal	DAC channelx analog output

Example of the code

```
int main(void){
    // System Clocks Configuration
    RCC_Configuration();

    // NVIC Configuration
    NVIC_Configuration();

    // Configure the GPIO for ADC1
    ADC1GPIOInit();

    // Init  ADC1
    ADC1Init();

    // Confugure the GPIO for LEDs
    LEDsGPIOInit();

    VoltageMeter();
}
```

```
void ADC1GPIOInit(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    // Enable ADC1 clock
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    // Config PA1 as analog input
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Speed = (GPIOSpeed_TypeDef)0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init (GPIOA, &GPIO_InitStructure);
}
```

```
void ADC1Init(void){
    ADC_InitTypeDef ADC_InitStructure;

    /* ADC1 is configured as follow:
       - Single channel, single conversion mode
       - Output data aligned left */
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left;
    ADC_InitStructure.ADC_NbrOfChannel = 1;

    // Do it from scratch
    ADC_DeInit(ADC1);
    ADC_Init(ADC1, &ADC_InitStructure);
```

```
    // Enable the ADC
    ADC_Cmd(ADC1, ENABLE);

    // ADC calibration
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1) == SET);
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1) == SET);
    // Configure channel
    ADC_RegularChannelConfig(ADC1, 1, 1, ADC_SampleTime_55Cycles5);
}
```

```
void LEDsGPIOInit(void){
    GPIO_InitTypeDef GPIO_InitStructure;
    // Initial LED PB[8..15]
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin =       GPIO_Pin_8 |
                                GPIO_Pin_9 |GPIO_Pin_10 | GPIO_Pin_11 |
          GPIO_Pin_12 | GPIO_Pin_13 |GPIO_Pin_14 |GPIO_Pin_15;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

```
void VoltageMeter(void){
    u16 currentReading = 0;
    u8 first8bits = 0;
    while(1){
        currentReading = GetADC1Channel1();
        // Get only higher byte
        first8bits = currentReading >> 8;
        OutputLEDs(first8bits);
    }
}

u16 GetADC1Channel1(void){
    // Configure channel
    ADC_RegularChannelConfig(ADC1, 1, 1, ADC_SampleTime_55Cycles5);
    // Start the conversion
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    // Wait until conversion completion
    while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
    // Get the conversion value
    return ADC_GetConversionValue(ADC1);
}
```

Questions?