Analog-to-Digital Converter (ADC)

Microcontroller Application and Development

Sorayut Glomglome

Outline

1. Analog-to-Digital Conversion
2. Keywords
3. ADC Applications
4. Nyquist Sampling Theorem
5. ADC architecture
6. STM32F767 ADC Module

Learning Outcomes

1. Understanding analog-to-digital conversion
2. Using analog input with MCU
3. Reading analog-to-digital data

Analog-to-Digital Conversion : Terminology

Analog : continuously valued signal, such as temperature or speed, with infinite possible values in between

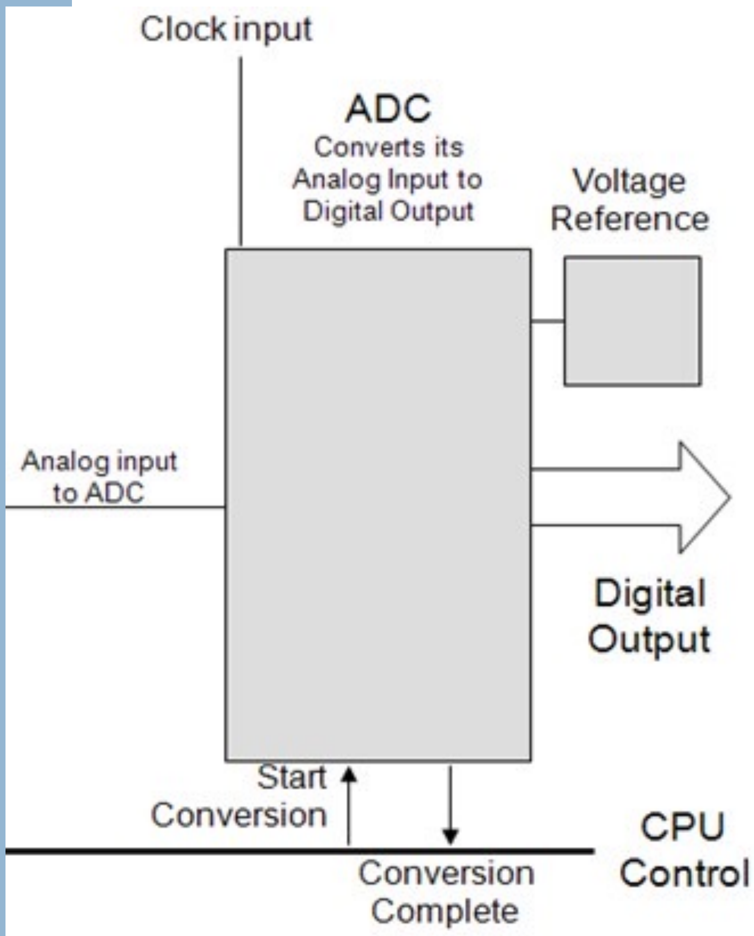
Digital : discretely valued signal, such as integers, encoded in binary

analog-to-digital converter: ADC, A/D, A2D; converts an analog signal to a digital signal

digital-to-analog converter: DAC, D/A, D2A

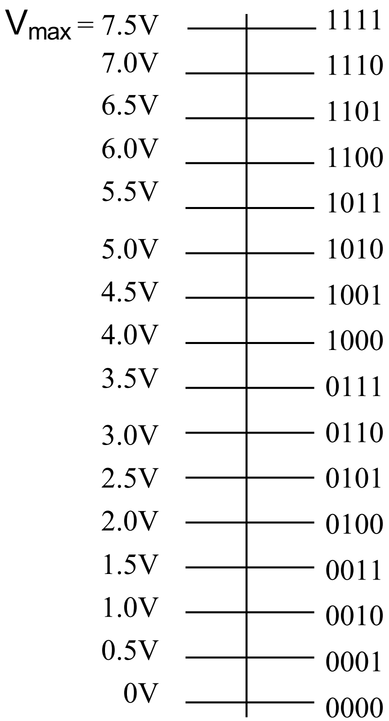
An embedded system's surroundings typically involve many analog signals.

Analog-to-Digital Converter

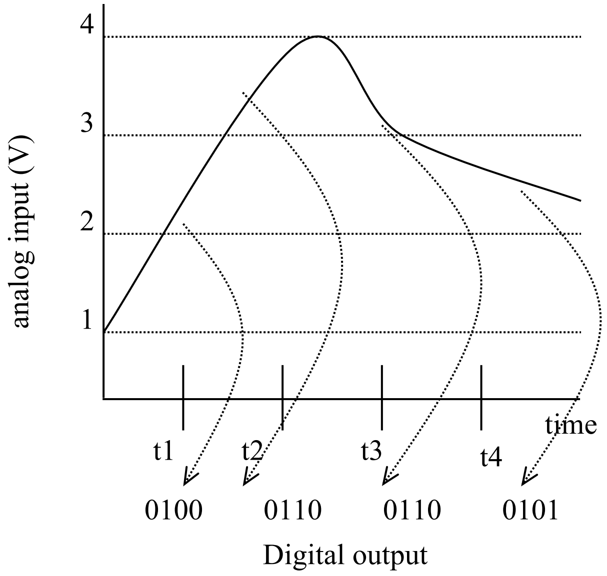


- ADC is an electronic circuit that measures the input voltage, and gives a binary output number proportional to its size.
- ADC compares an input voltage to a reference voltage.
- Conversion takes time (μs++), so the ADC needs to signal when it has finished.

Analog-to-digital converters

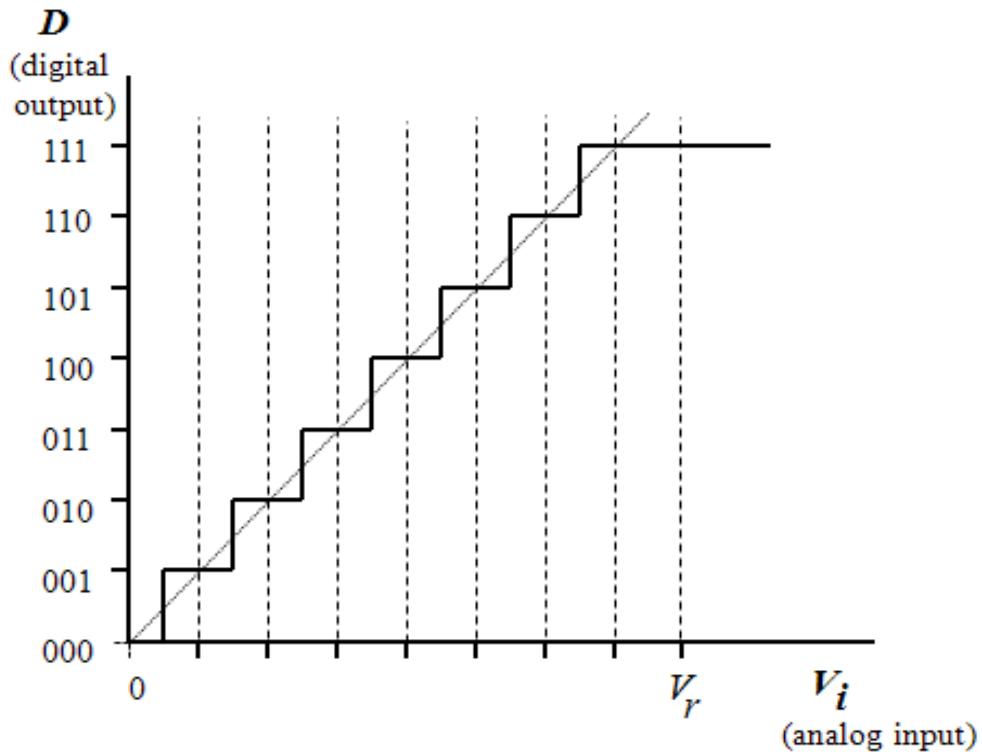


proportionality



analog to digital

Range, Resolution and Quantisation



$$D = \frac{V_i}{V_r} \, x \, 2^n$$

- The ADC action follows Equation
  - D : output binary number
  - n : bit number
  - V_i : input voltage
  - V_r : reference voltage
- The difference between the maximum and minimum input values is called the Range.
- many ADC circuits the range is equal to the reference voltage.

Proportional Signals

<u>Simple Equation</u>

Assume minimum voltage of 0 V.

Vmax = maximum voltage of the analog signal

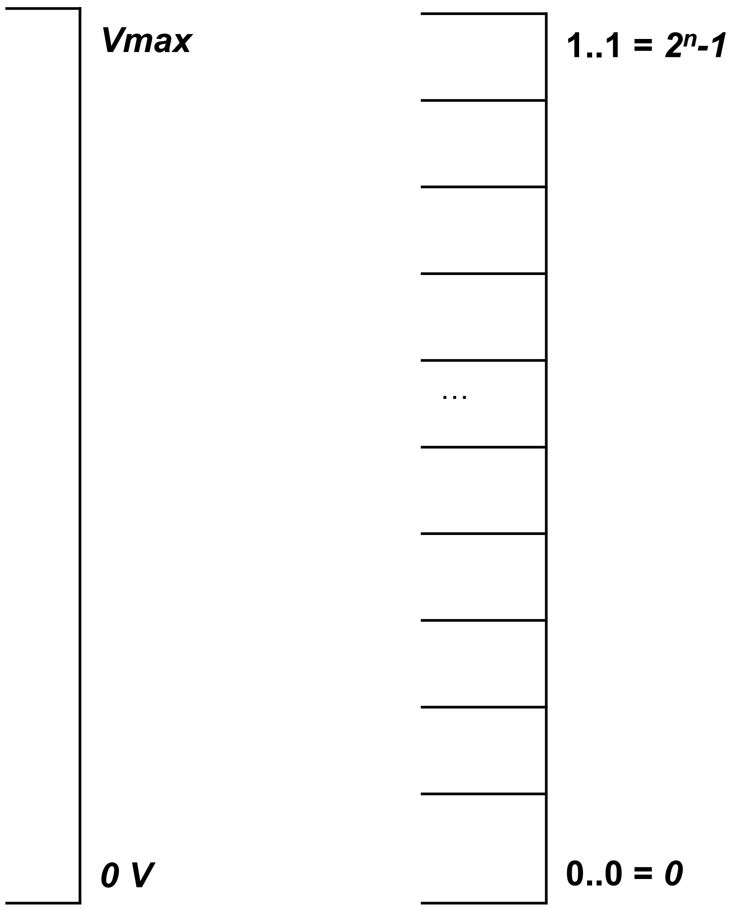
a = analog value

n = number of bits for digital encoding

- 2^n = number of digital codes
- M = number of steps, either 2^n or $2^n - 1$

d = digital encoding

$$a \,/\, Vmax = d \,/\, M$$



Resolution

Let $n = 2$

$\underline{M = 2^n - 1}$

3 steps on the digital scale

$d_0 = 0 = 0b00$

$d_{Vmax} = 3 = 0b11$

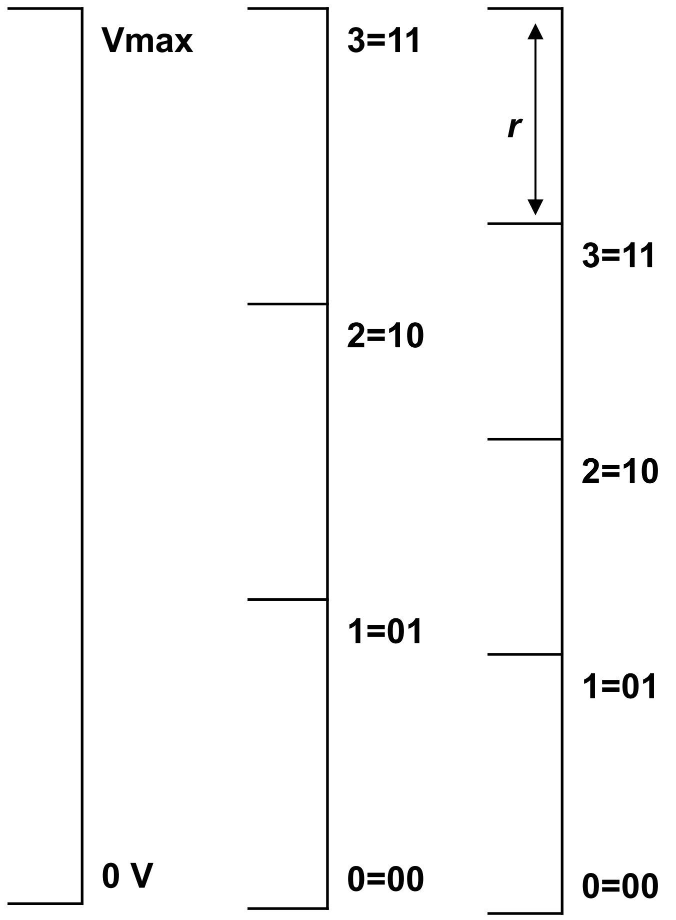
$\underline{M = 2^n}$

4 steps on the digital scale

$d_0 = 0 = 0b00$

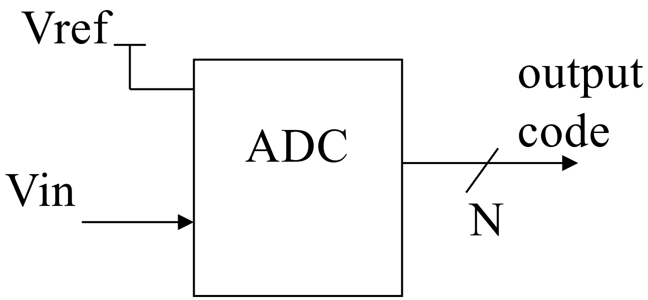
$d_{Vmax - r} = 3 = 0b11$ (no d_{Vmax})

***r*, resolution**: smallest analog change resulting from changing one bit



Resolution

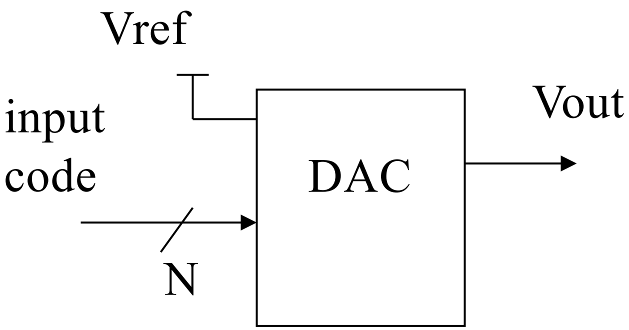
ADC: Vin = input voltage, Vref = reference voltage
N = number of bits of precision



$Vin / Vref * 2^N = output\_code$
$output\_code / 2^N * Vref = Vin$

1 LSB = $Vref/2^N$

DAC: Vout = output voltage, Vref = reference voltage,
N = number of bits of precision



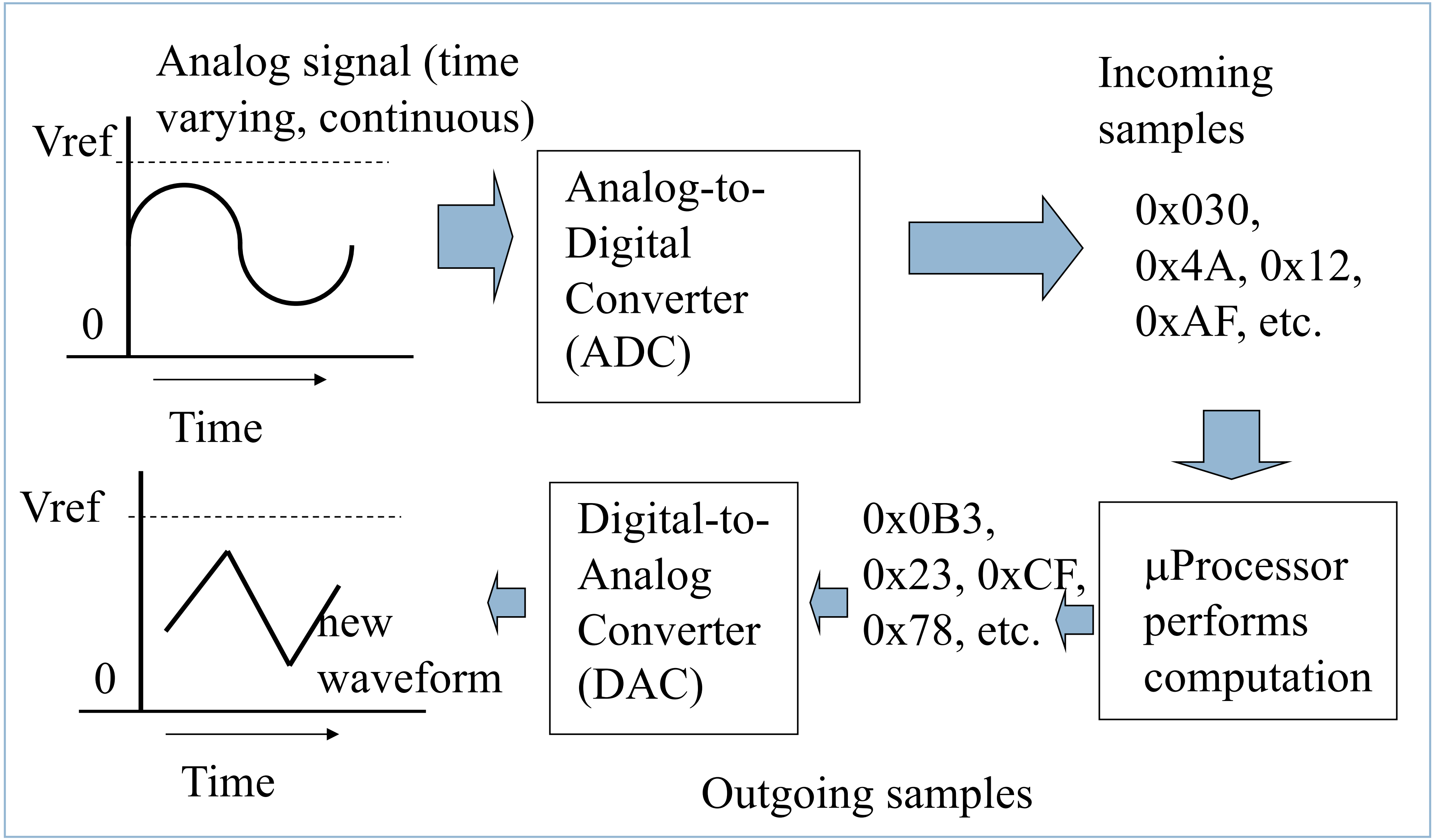
$Vout / Vref * 2^N = input\_code$
$input\_code / 2^N * Vref = Vout$

1 LSB = $Vref/2^N$

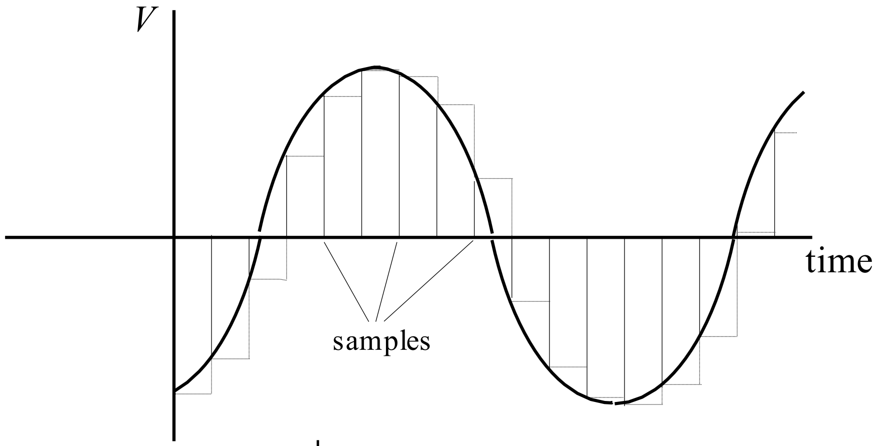
ADC Applications

- Data logging
- Audio
  - Speech recognition
  - special effects (reverb, noise cancellation, etc)
- Video
  - Filtering
  - Special effects
  - Compression

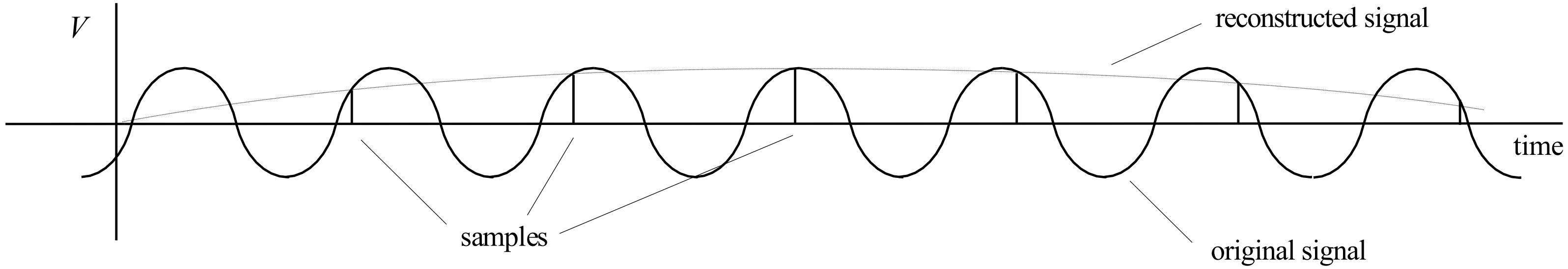
Digital Signal Processing



Sampling Frequency and Aliasing



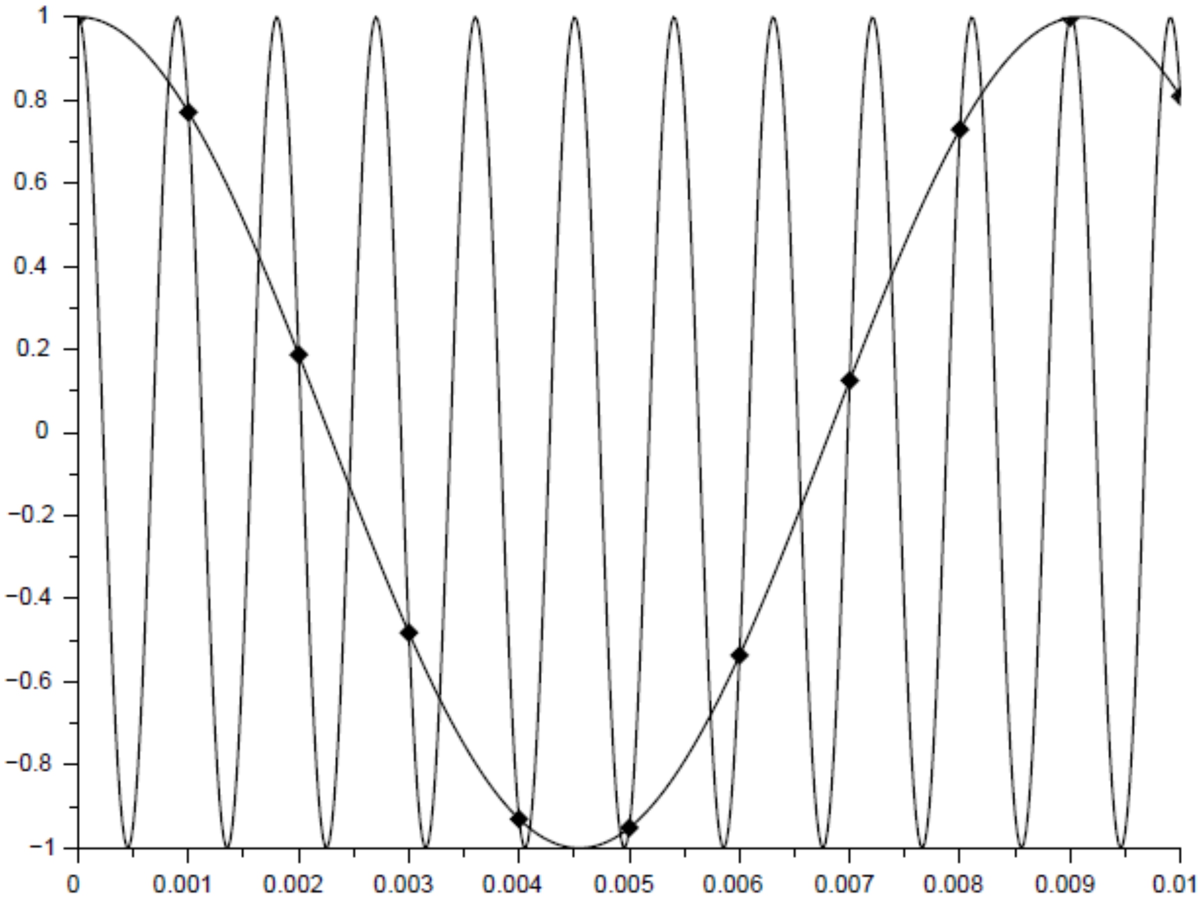
- Sample and quantise
- More samples more accurate the digital data
- sampling frequency



The Nyquist theorem (sampling theorem):

- Sampling frequency must be at least twice of the maximum signal frequency.
- If the sampling is less than twice, then *aliasing* occurs
- Aliasing : a new lower frequency signal.

Aliasing of Two Sinewaves



Sample ADC Computations

If Vref = 5V, and the 10-bit A/D output code is 0x12A, what is the ADC input voltage?

output_code/2^N * Vref = (0x12A)/2^{10} * 5 V
= 298/1024 * 5 V = 1.46 V (Vin)

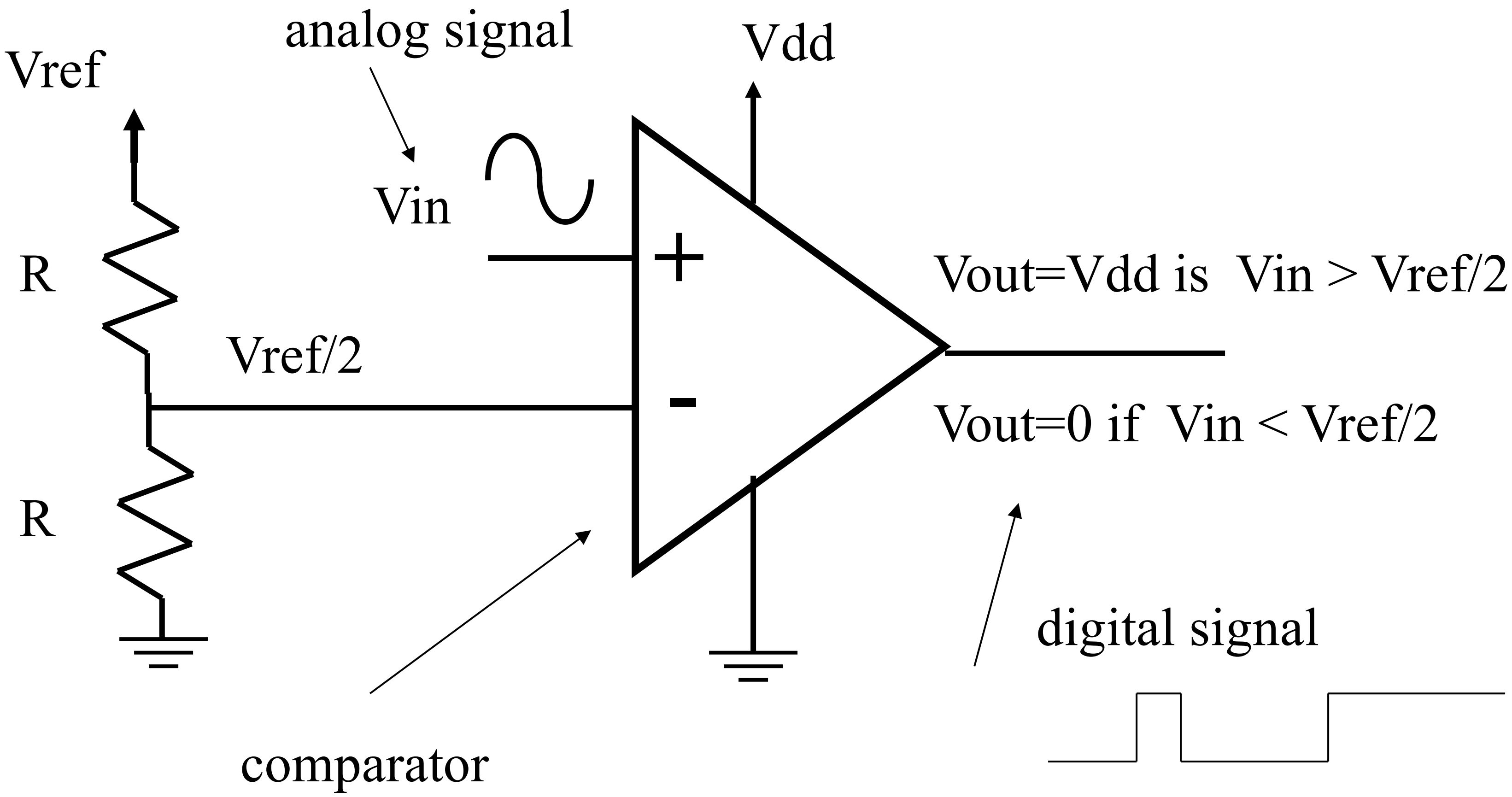
If Vref = 5V, and the upper 8 bits of the A/D output code is 0xA9, what is the ADC input voltage?

output_code/2^N * Vref = (0xA9)/2^8 * 5 V
= 169/256 * 5 V = 3.3 V (Vin)

If Vref = 4V, and the A/D input voltage is 2.35 V, what is the ADC output code, upper 8-bits?

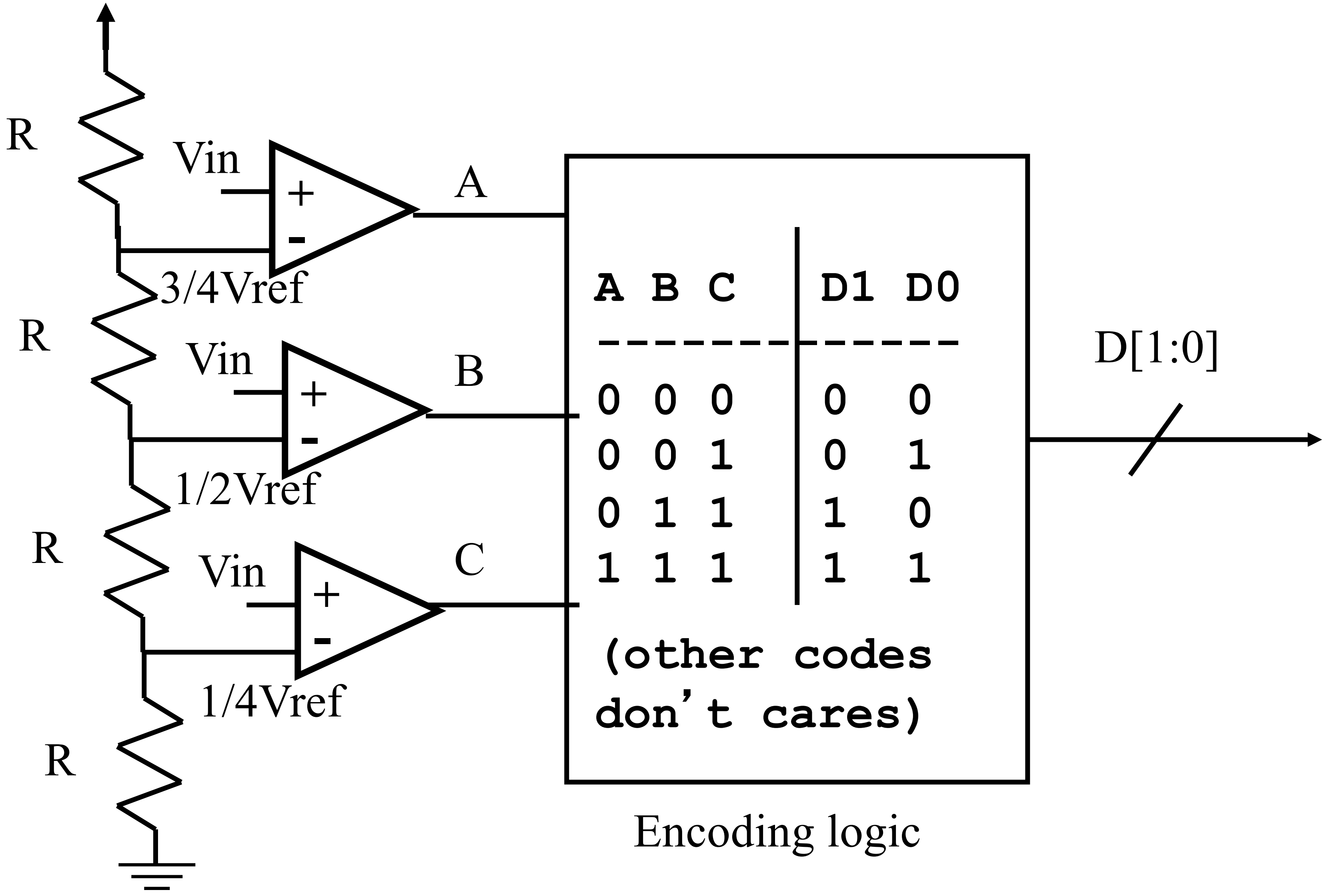
Vin/ Vref * 2^N = 2.35 V/ 4 V * 2^8
= .5875 * 256 = 150.4 = 150 = 0x96

A 1-bit ADC



analog signal
Vref
Vdd
Vin
+
R
Vout=Vdd is Vin > Vref/2
Vref/2
-
Vout=0 if Vin < Vref/2
R
digital signal
comparator

A 2-bit ADC

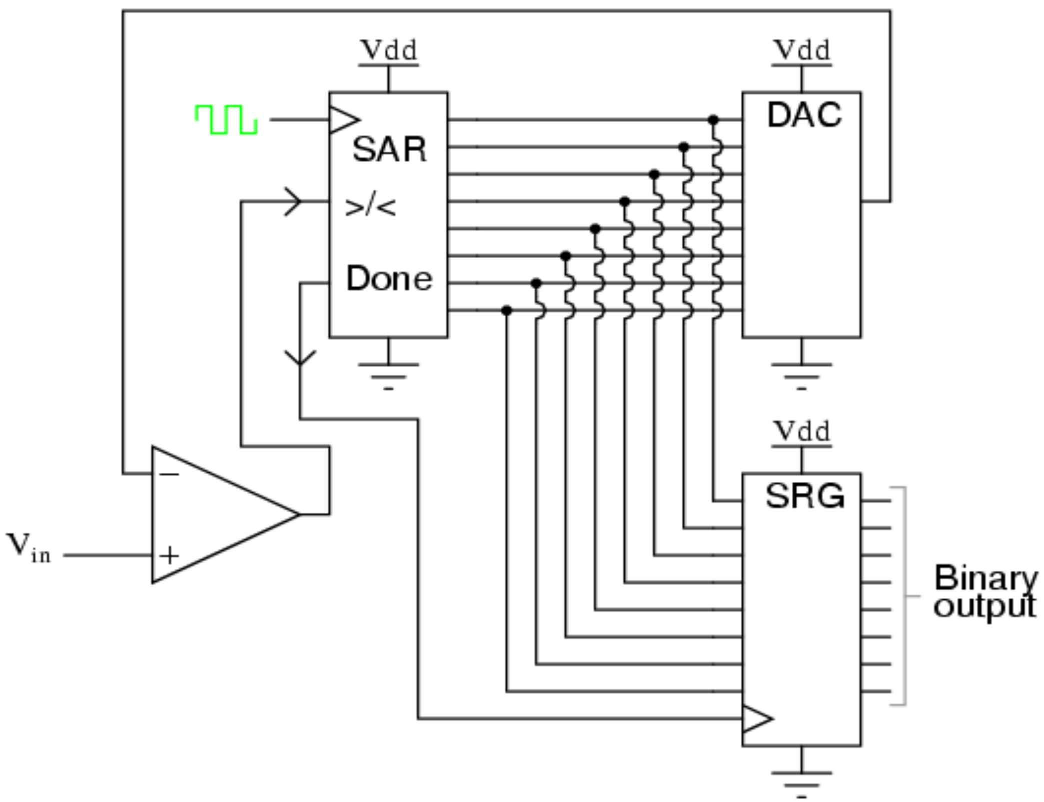

R
Vin
+
-
A
3/4Vref
R
Vin
B
1/2Vref
R
Vin
C
1/4Vref
R
A B C | D1 D0
0 0 0 | 0 0
0 0 1 | 0 1
0 1 1 | 1 0
1 1 1 | 1 1
(other codes don't cares)
D[1:0]
Encoding logic

ADC Architectures

- The previous architectures are called *Flash* ADCs
  - Fastest possible conversion time
  - Requires the most transistors of any architecture
  - N-bit converter requires 2^N-1 comparators.
  - Commercially available flash converters up to 12 bits.
  - Conversion done in one clock cycle

- *Successive approximation* ADCs
  - Use only one comparator
  - Take one clock cycle per bit
  - High precision (16-bit converters are available)

Successive Approximation ADC



Output is Q[N]

First, set DAC to produce Vref/2.

Output of Comparator is Q[N-1] (MSB)

If MSB =1 , then Vin between Vref and Vref/2, so set DAC to produce ¾ Vref.

If MSB=0, then Vin between Vref/2 and 0, so set DAC to ½ Vref.

Output of comparator is now Q[N-2].

Do this for each bit.

Takes N cycles.

ADC using successive approximation

- Given an analog input signal whose voltage should range from 0 to 15 volts, and an 8-bit digital encoding, calculate the correct encoding for 5 volts. Then trace the successive-approximation approach to find the correct encoding.
- Assume $M = 2^n - 1$

$$a / Vmax = d / M$$

$$5 / 15 = d / (256 - 1)$$

$d = 85$ or binary 01010101

ADC using successive approximation

Step 1-4: determine bits 0-3

$\frac{1}{2}(V_{max} - V_{min}) = 7.5$ volts
$V_{max} = 7.5$ volts.

0	0	0	0	0	0	0	0

$\frac{1}{2}(7.5 + 0) = 3.75$ volts
$V_{min} = 3.75$ volts.

0	1	0	0	0	0	0	0

$\frac{1}{2}(7.5 + 3.75) = 5.63$ volts
$V_{max} = 5.63$ volts

0	1	0	0	0	0	0	0

$\frac{1}{2}(5.63 + 3.75) = 4.69$ volts
$V_{min} = 4.69$ volts.

0	1	0	1	0	0	0	0

ADC using successive approximation

Step 5-8: Determine bits 4-7

$\frac{1}{2}(5.63 + 4.69) = 5.16$ volts
$V_{max} = 5.16$ volts.

0	1	0	1	0	0	0	0

$\frac{1}{2}(5.16 + 4.69) = 4.93$ volts
$V_{min} = 4.93$ volts.

0	1	0	1	0	1	0	0

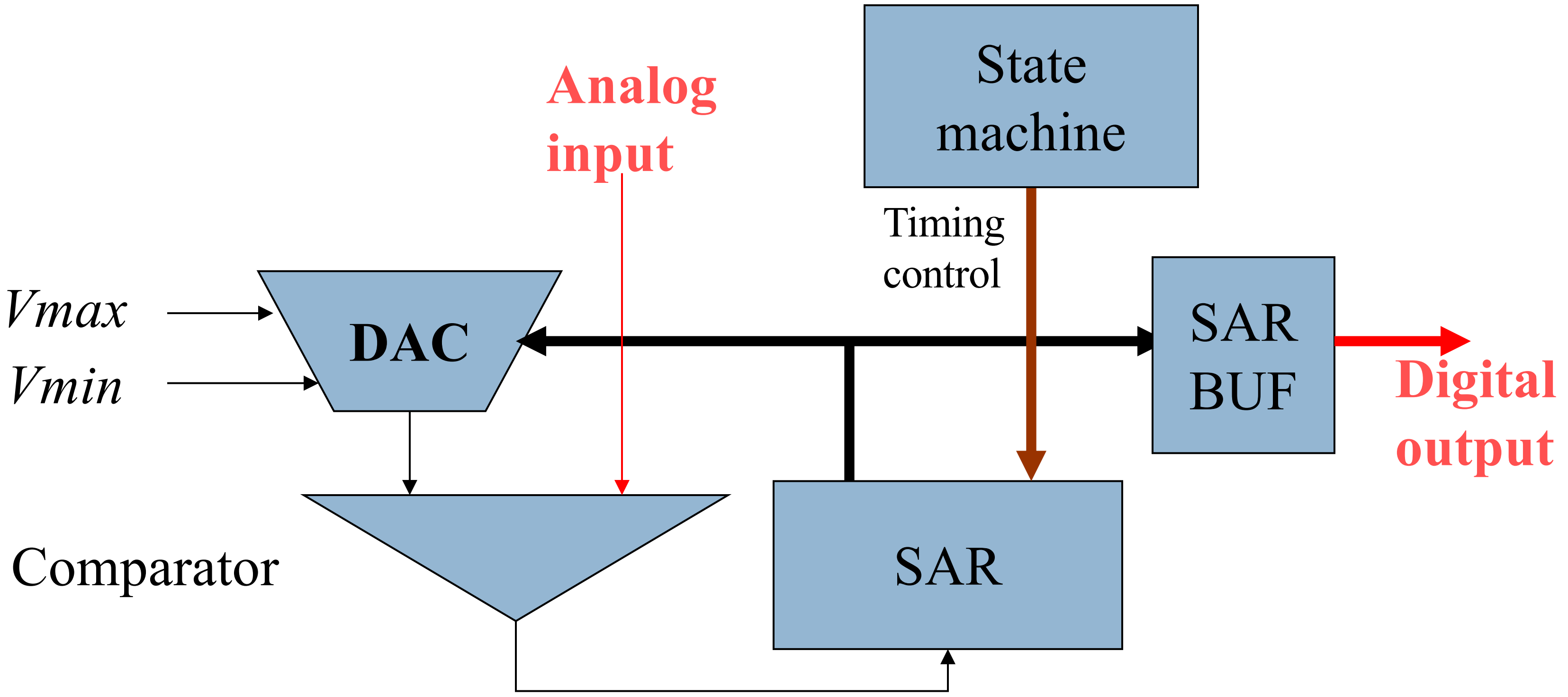
$\frac{1}{2}(5.16 + 4.93) = 5.05$ volts
$V_{max} = 5.05$ volts.

0	1	0	1	0	1	0	0

$\frac{1}{2}(5.05 + 4.93) =$
4.99 volts

0	1	0	1	0	1	0	1

Constructing ADC



SAR: Successive approximation register

STM32F767 ADC Module

- 12-bit ADC is a successive approximation analog-to-digital converter
- 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 left or right-aligned 16-bit data register.

STM32F767 ADC Features

- 3 ADCs
- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- Interrupt generation at the end of conversion
- Single and continuous conversion modes
- Regular and Injection mode
- Data alignment with in-built data coherency
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed
- ADC input range: VREF– ≤ VIN ≤ VREF+

Examples of Conversion Mode

Figure 4. Single-channel, continuous conversion mode

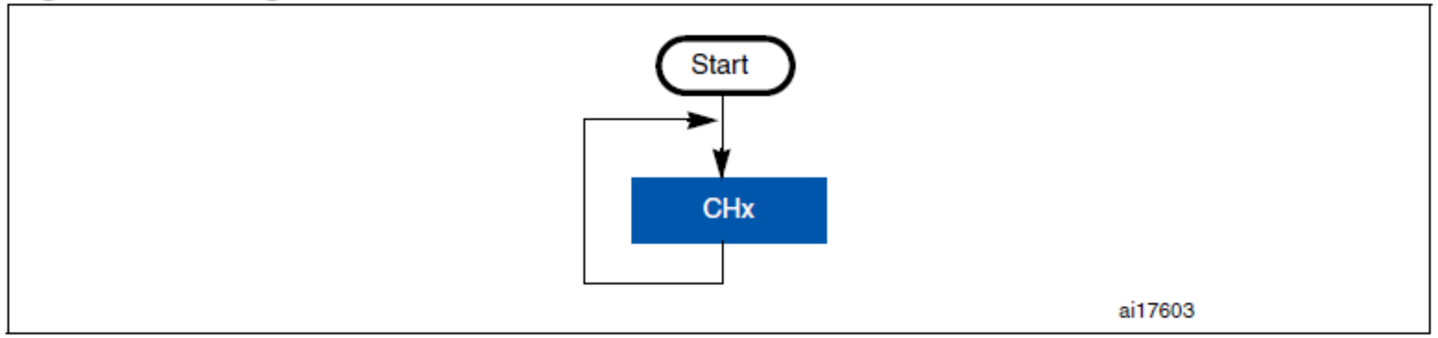
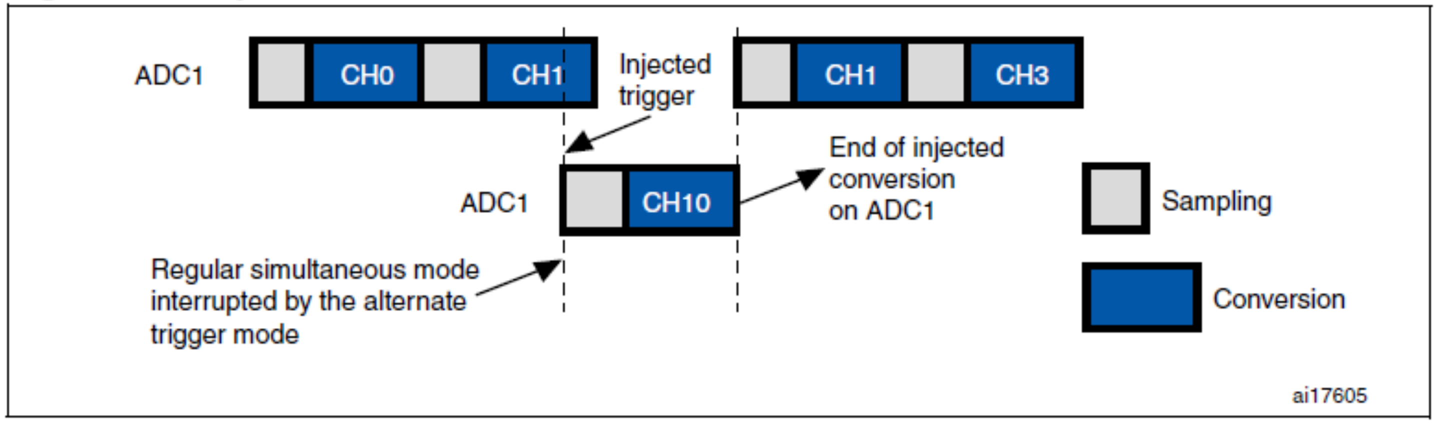
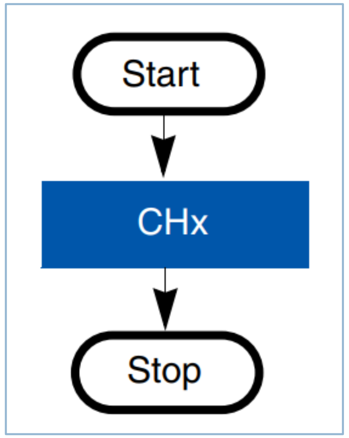


Figure 6. Injected conversion mode

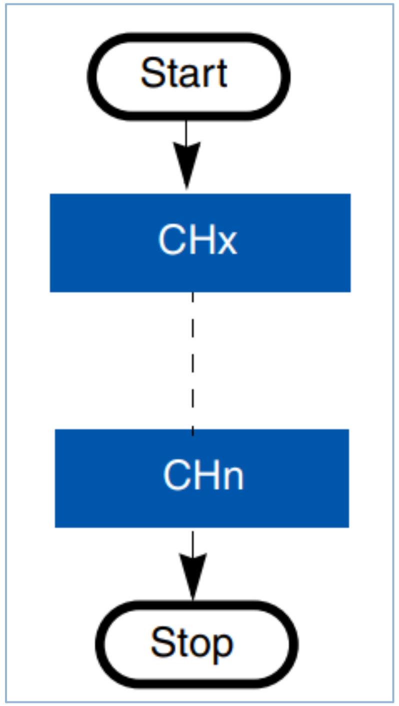


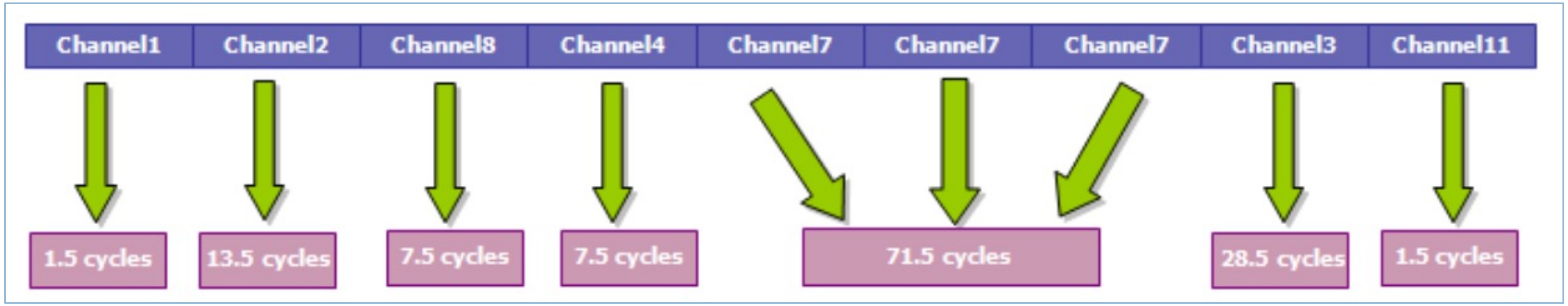
Single Conversion

Single Channel

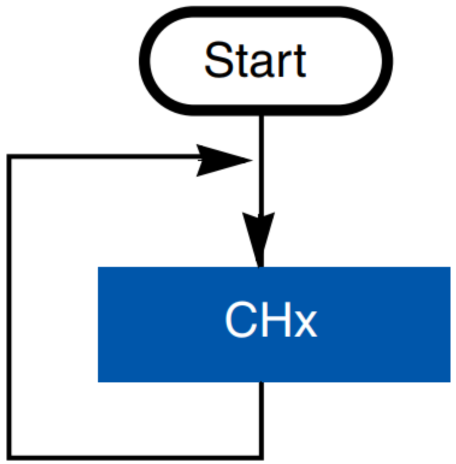


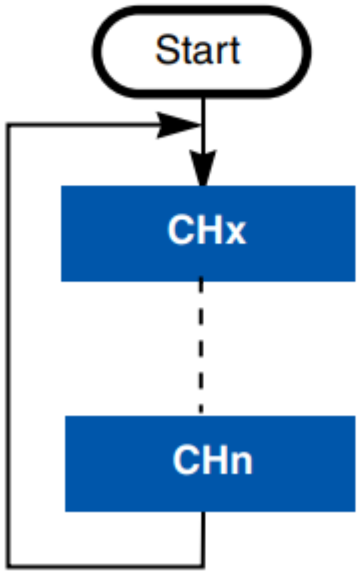
Multichannel





Continuous Mode





Single Channel

Multichannel

Other modes

Figure 6. Injected conversion mode

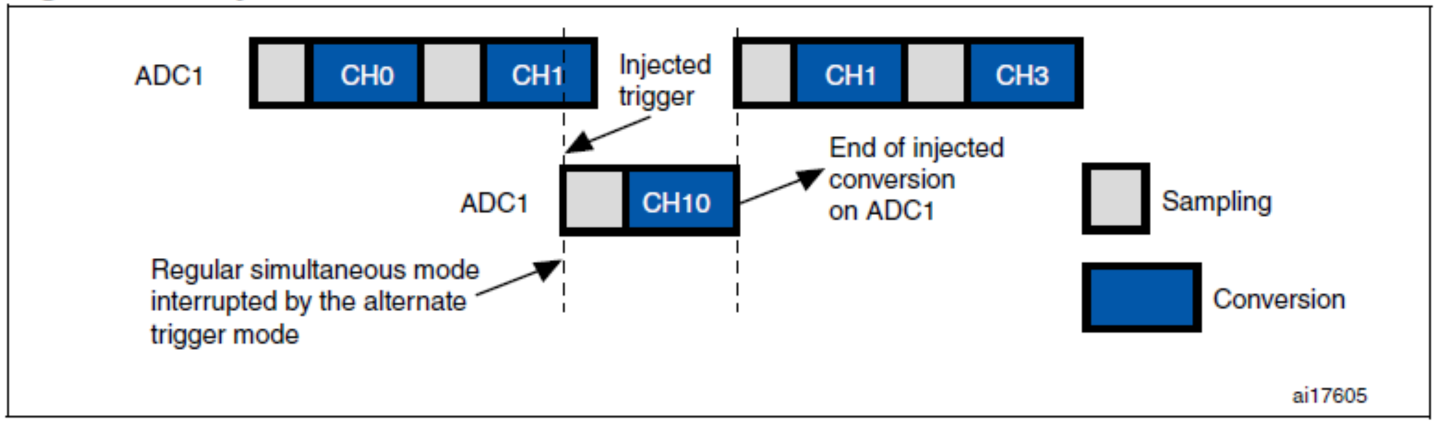
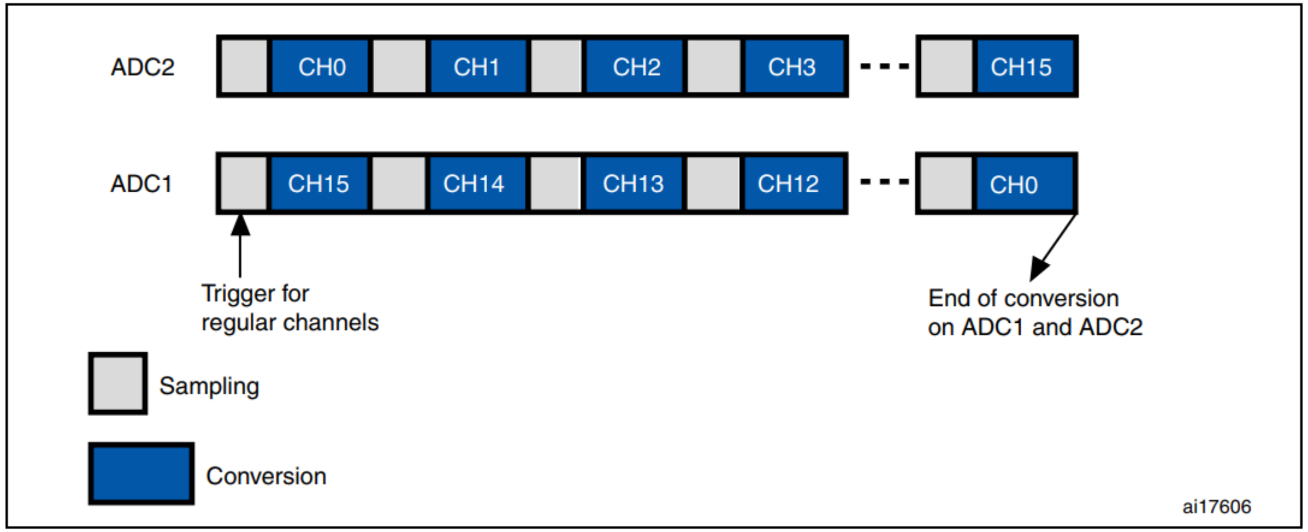
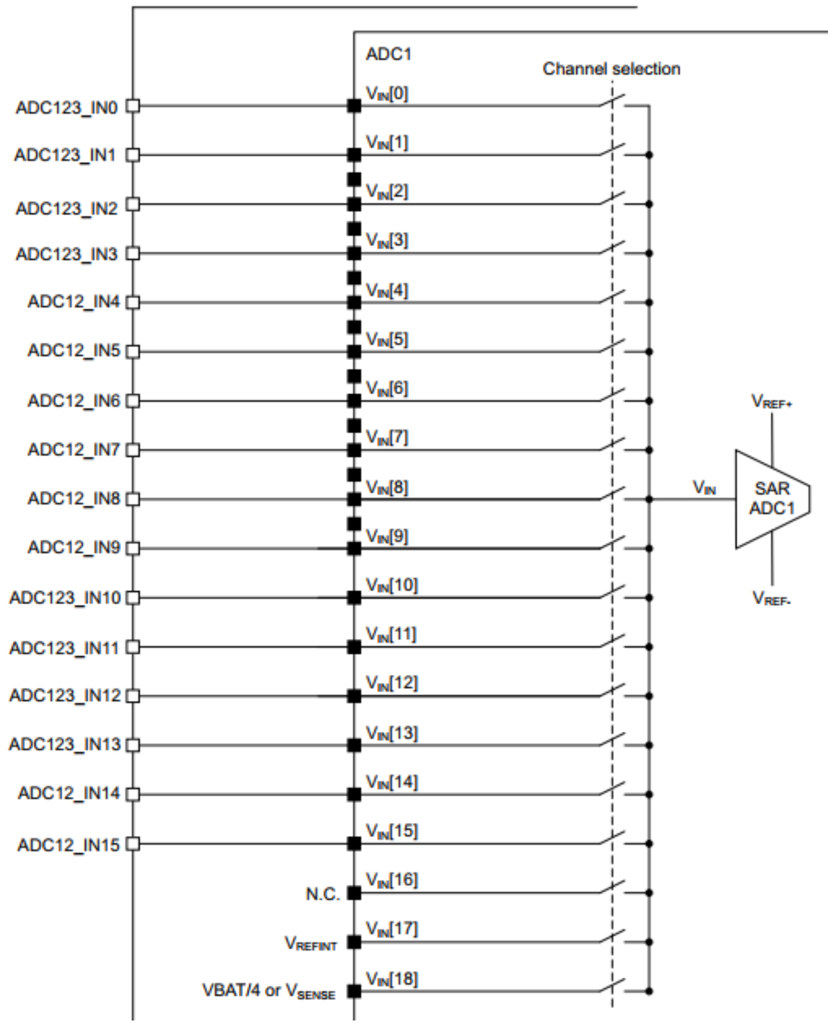
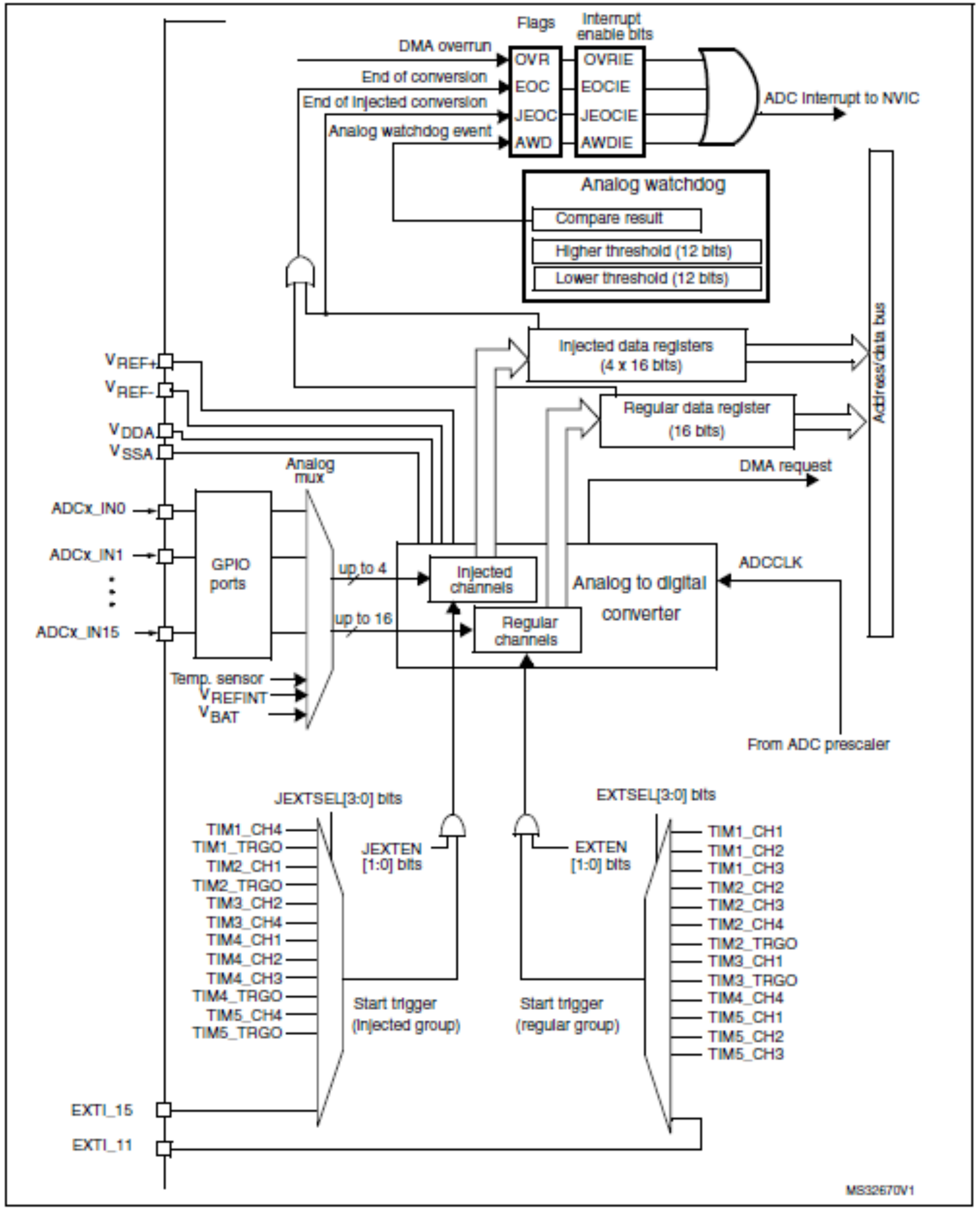


Figure 7. Dual regular simultaneous mode

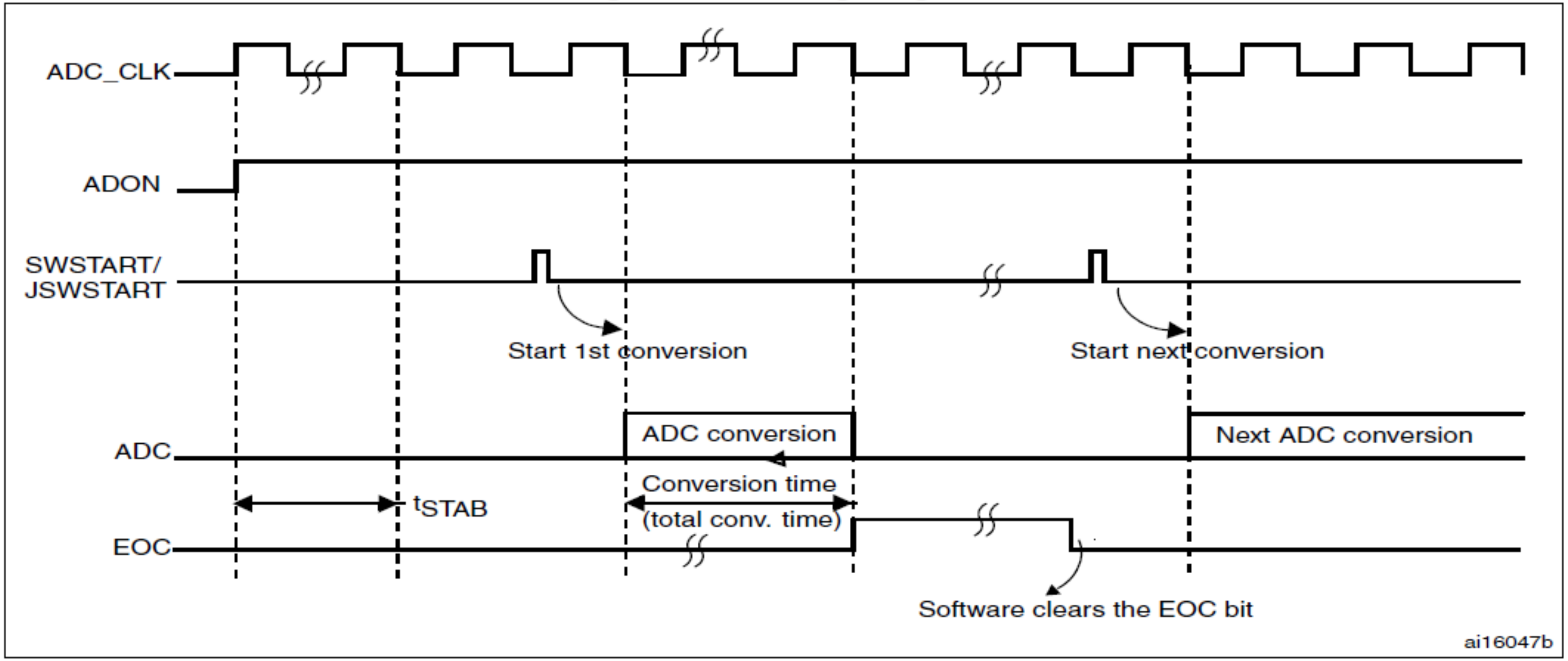


ADC Block Diagram





ADC Timing Diagram



Data Alignment

Figure 35. Right alignment of 12-bit data

Injected group

SEXT	SEXT	SEXT	SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Regular group

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

ai16050

Figure 36. Left alignment of 12-bit data

Injected group

SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0

Regular group

D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0

ai16051

Figure 37. Left alignment of 6-bit data

Injected group

SEXT	SEXT	SEXT	SEXT	SEXT	SEXT	SEXT	SEXT	SEXT	D5	D4	D3	D2	D1	D0	0

Regular group

0	0	0	0	0	0	0	0	D5	D4	D3	D2	D1	D0	0	0

ai16052

11.12.14 ADC regular data register (ADC_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **DATA[15:0]:** Regular data

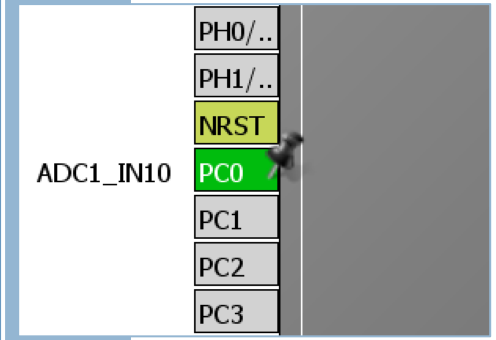
These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in *Figure 35* and *Figure 36*.

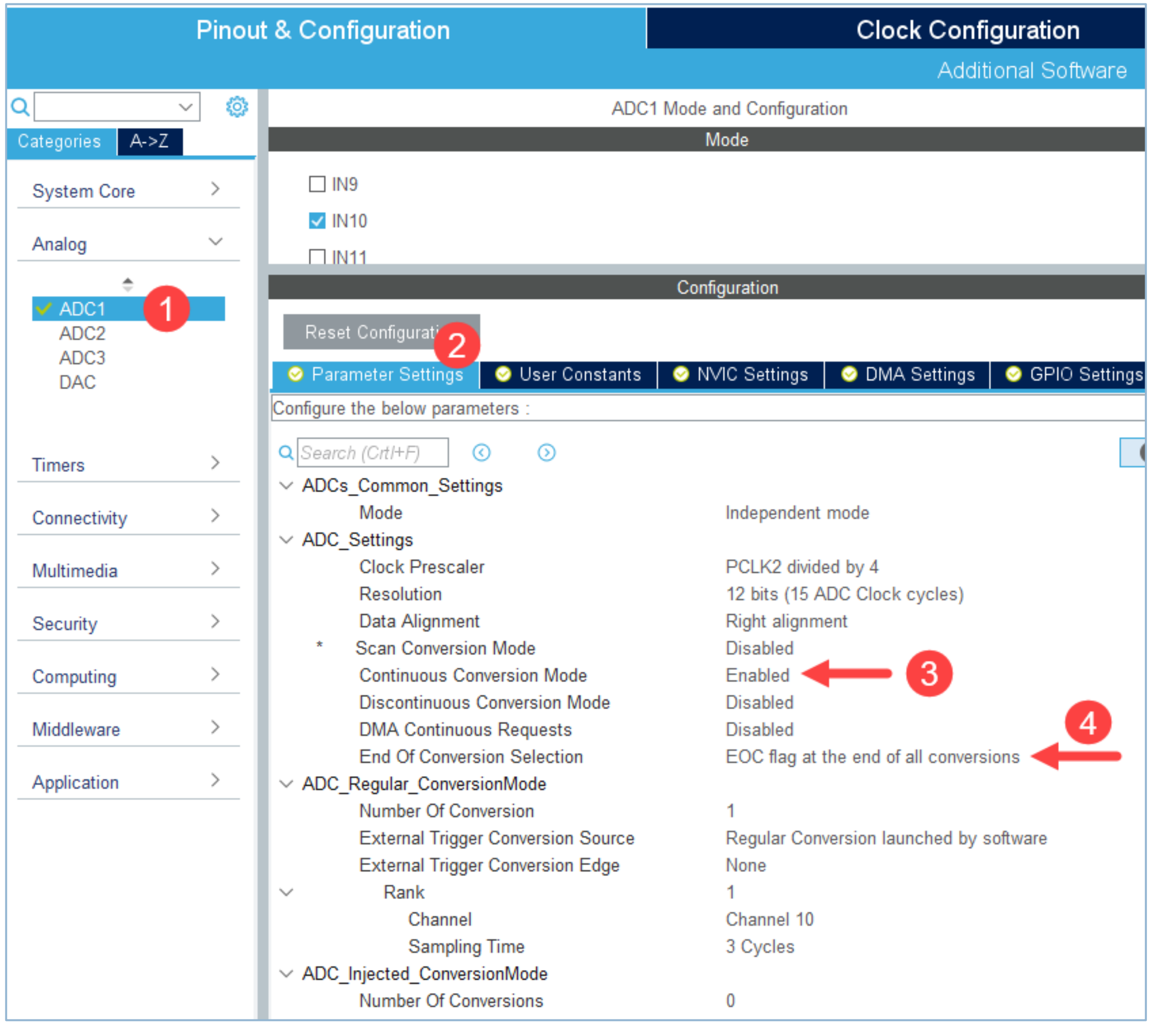
```
volatile uint32_t adc_val = 0;

HAL_ADC_Start(&hadc1);
```

```
while (1){
  while ( HAL_ADC_PollForConversion(&hadc1, 100) != HAL_OK ){}
  adc_val = HAL_ADC_GetValue(&hadc1);
}
```

Single Channel Continuous Conversion





```c
ADC_HandleTypeDef hadc1;

/* ADC1 init function */
void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  /** Configure the global features of the ADC
  (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel
  its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_10;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

```c
void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* ADC1 clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOC_CLK_ENABLE();
    /**ADC1 GPIO Configuration
    PC0     ------> ADC1_IN10
    */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /* USER CODE BEGIN ADC1_MspInit 1 */

  /* USER CODE END ADC1_MspInit 1 */
  }
}
```

ADC Blocking Call (Polling)

```
volatile uint32_t adc_val = 0;

HAL_ADC_Start(&hadc1);

while (1){
  while ( HAL_ADC_PollForConversion(&hadc1, 100) != HAL_OK ){}
  adc_val = HAL_ADC_GetValue(&hadc1);
}
```

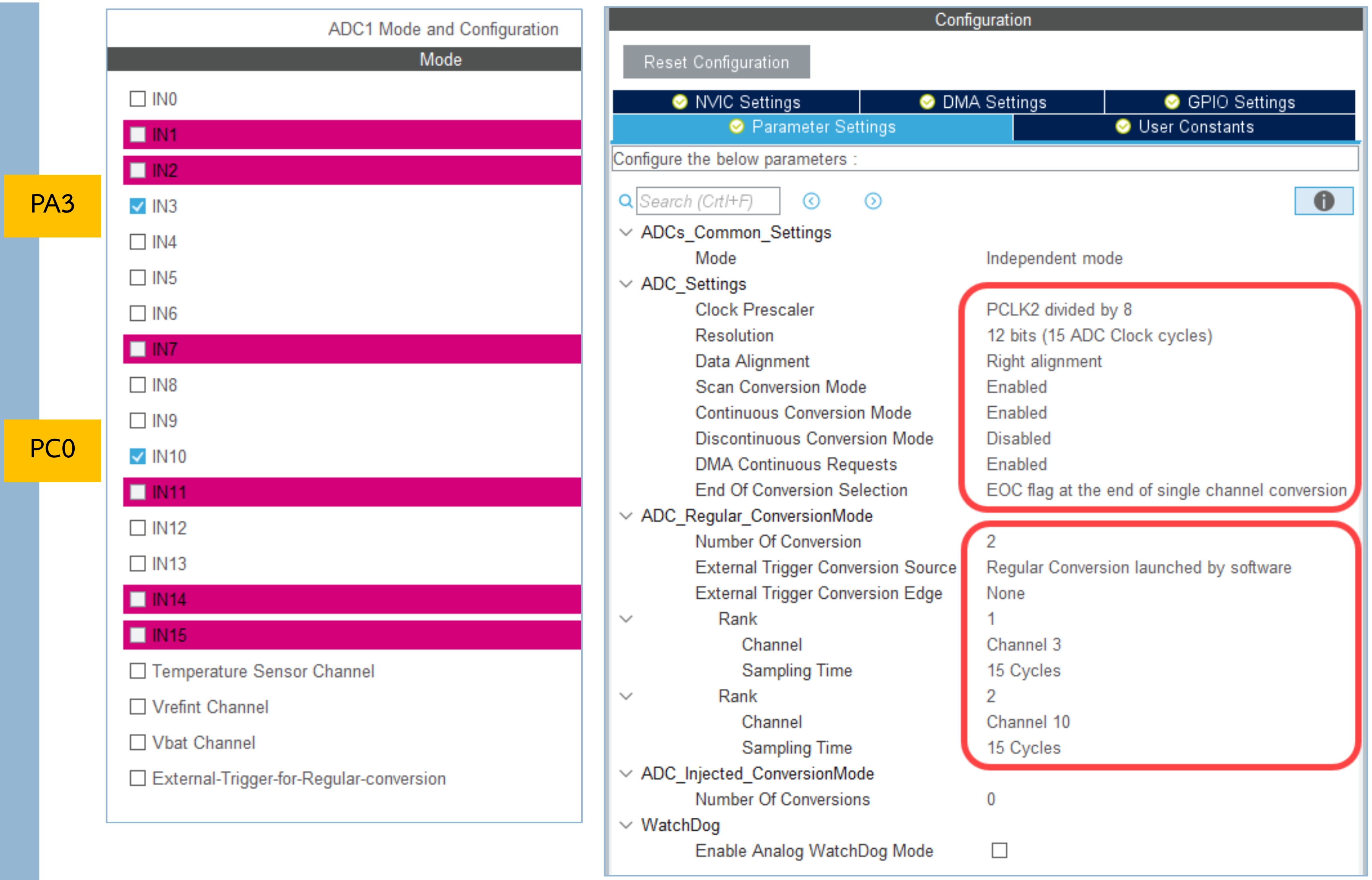
HAL_ADC_GetValue

Function Name	**uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)**
Function Description	Get ADC regular group conversion result.
Parameters	• **hadc:** ADC handle
Return values	• Converted value
Notes	• Reading DR register automatically clears EOC (end of conversion of regular group) flag.

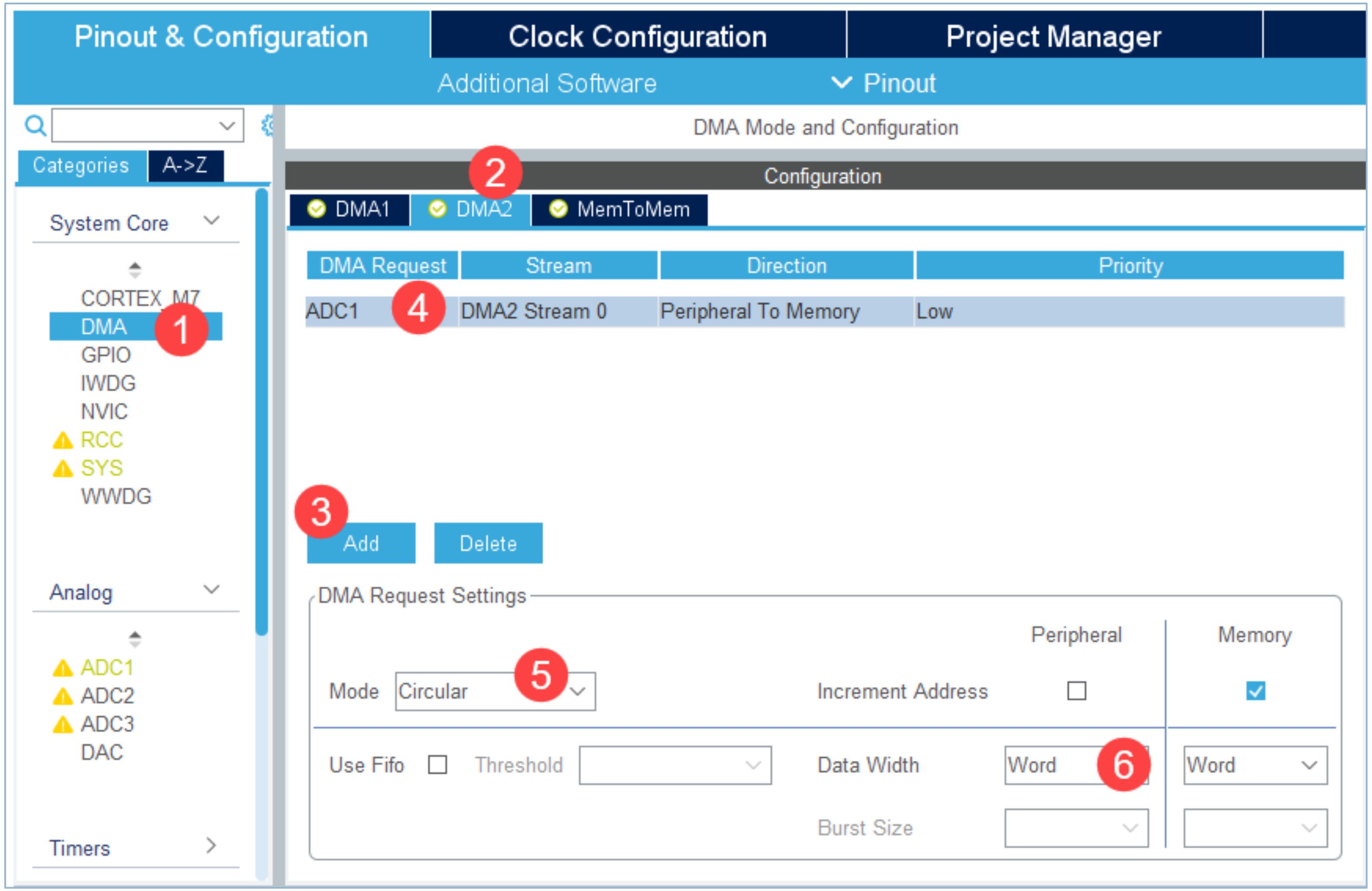
HAL_ADC_PollForConversion

Function Name	**HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)**
Function Description	Poll for regular conversion complete.
Parameters	• **hadc:** pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • **Timeout:** Timeout value in millisecond.
Return values	• **HAL:** status
Notes	• ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function. • This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each

Multichannel Continuous with DMA



Direct Memory Access Controller Config



DMA Global Interrupt

NVIC Mode and Configuration

Configuration

NVIC | Code generation

Priority Group: 2 bits for pre-emption priority 2 bits f... (1) — ☐ Sort by Premption Priority and S

Search: Search (Crtl+F) — ☐ Show only enabled interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Prio
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
ADC1, ADC2 and ADC3 global interrupts	☐	0	0
USART3 global interrupt	☐	0	0
EXTI line[15:10] interrupts	☐	0	0
DMA2 stream0 global interrupt	☑	1 (2)	0
USB On The Go FS global interrupt	☐	0	0
FPU global interrupt	☐	0	0

stm32f7xx_it.c

```
void DMA2_Stream0_IRQHandler(void)
{
  /* USER CODE BEGIN DMA2_Stream0_IRQn 0 */

  /* USER CODE END DMA2_Stream0_IRQn 0 */
  HAL_DMA_IRQHandler(&hdma_adc1);
  /* USER CODE BEGIN DMA2_Stream0_IRQn 1 */

  /* USER CODE END DMA2_Stream0_IRQn 1 */
}
```

ADC Non Blocking Call

Function	Interrupt	DMA
Trigger	HAL_ADC_Start_IT()	HAL_ADC_Start_DMA()
ISR	ADC_IRQHandler()	DMA2_Stream0_IRQHandler()
Callback	HAL_ADC_ConvCpltCallback()	HAL_ADC_ConvCpltCallback() HAL_ADC_ConvHalfCpltCallback()
Error Callback	HAL_ADC_ErrorCallback()	HAL_ADC_ErrorCallback()
Stop	HAL_ADC_Stop_IT()	HAL_ADC_Stop_DMA()

HAL_ADC_Start_DMA

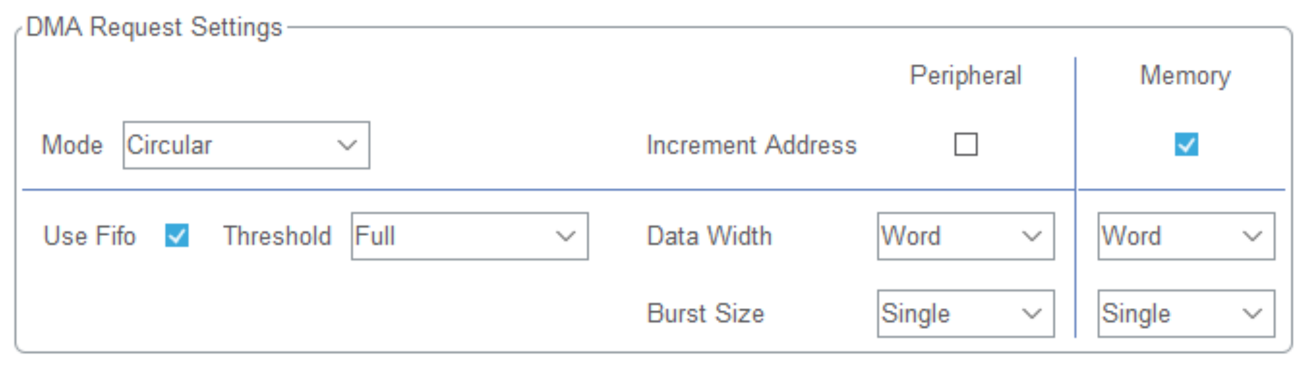
Function Name	**HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)**
Function Description	Enables ADC DMA request after last transfer (Single-ADC mode) and enables ADC peripheral.
Parameters	• **hadc:** pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • **pData:** The destination Buffer address. • **Length:** The length of data to be transferred from ADC peripheral to memory.

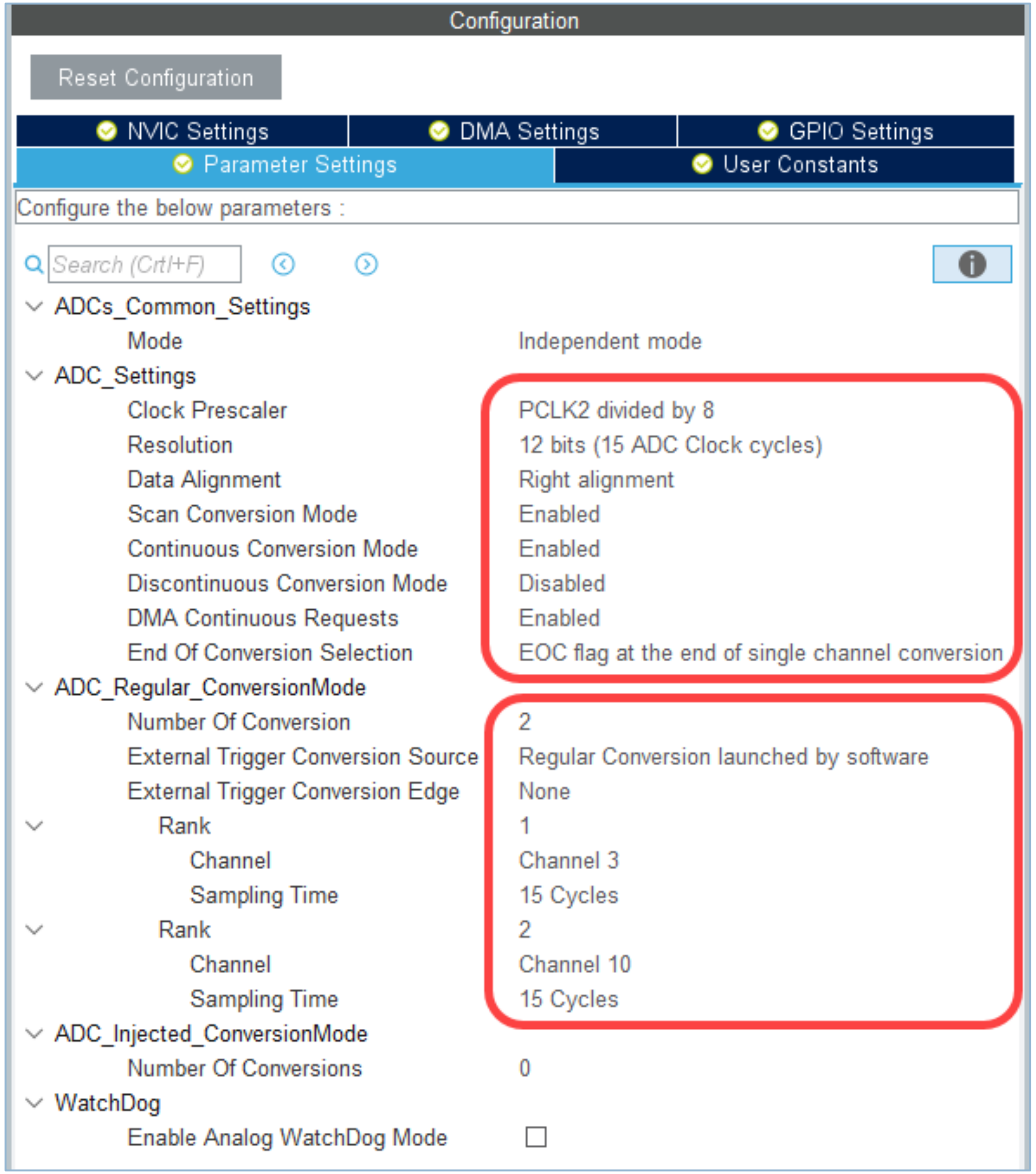
```
/* USER CODE BEGIN PV */
 uint32_t adc_val[2];
/* USER CODE END PV */
```

```
/* USER CODE BEGIN 2 */
HAL_ADC_Start_DMA(&hadc1, adc_val, 2);
/* USER CODE END 2 */
```

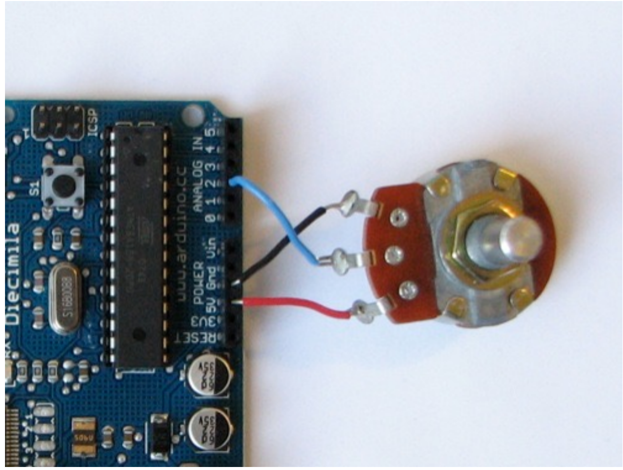
Beware of Overrun

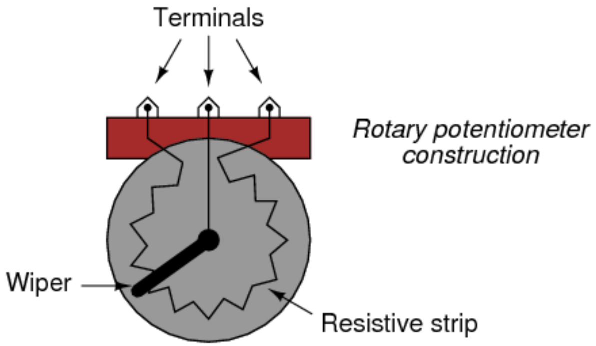
- Slow ADC Down
- Use FIFO in DMA
- Check OVR Flag
- Restart ADC

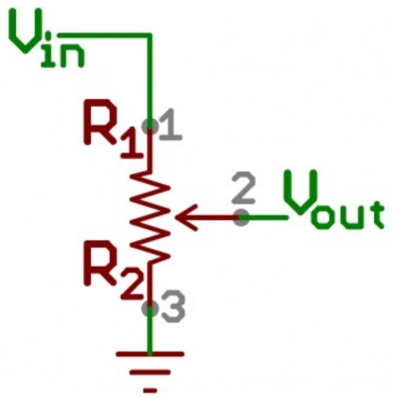




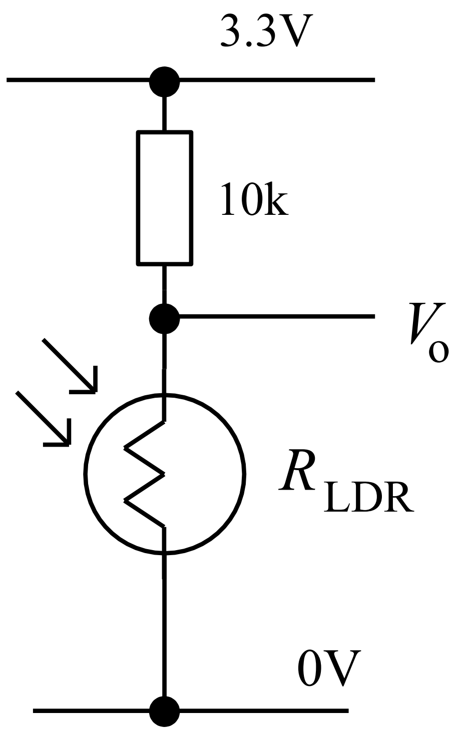
Potentiometer as Voltage Devider







Simple Analog Sensors : the Light Dependent Resistor

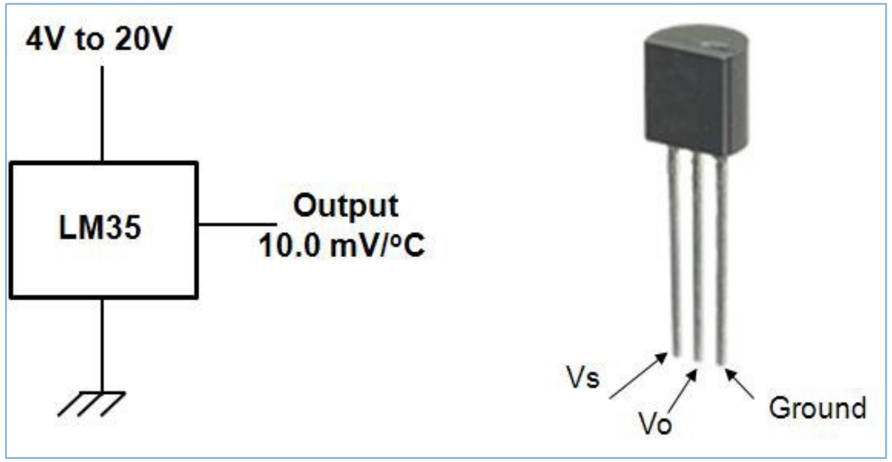


- LDR is made from semiconductor material.
- the brighter the light, the more electrons are released
- Released electrons are available to conduct electricity; the resistance falls
- Remove the light, electrons are back into their place; the resistance goes up again.

Illumination (lux)	R_{LDR} (Ω)	V_o
Dark	$\geq$ 1.0 M	$\geq$ 3.27 V
10	9k	1.56 V
1,000	400	0.13 V

Integrated Circuit Temperature Sensor

- Semiconductor action is highly dependent on temperature
- LM35 has an output of 10 mV/°C
- Up to 110 °C



Smoothing ADC Signal

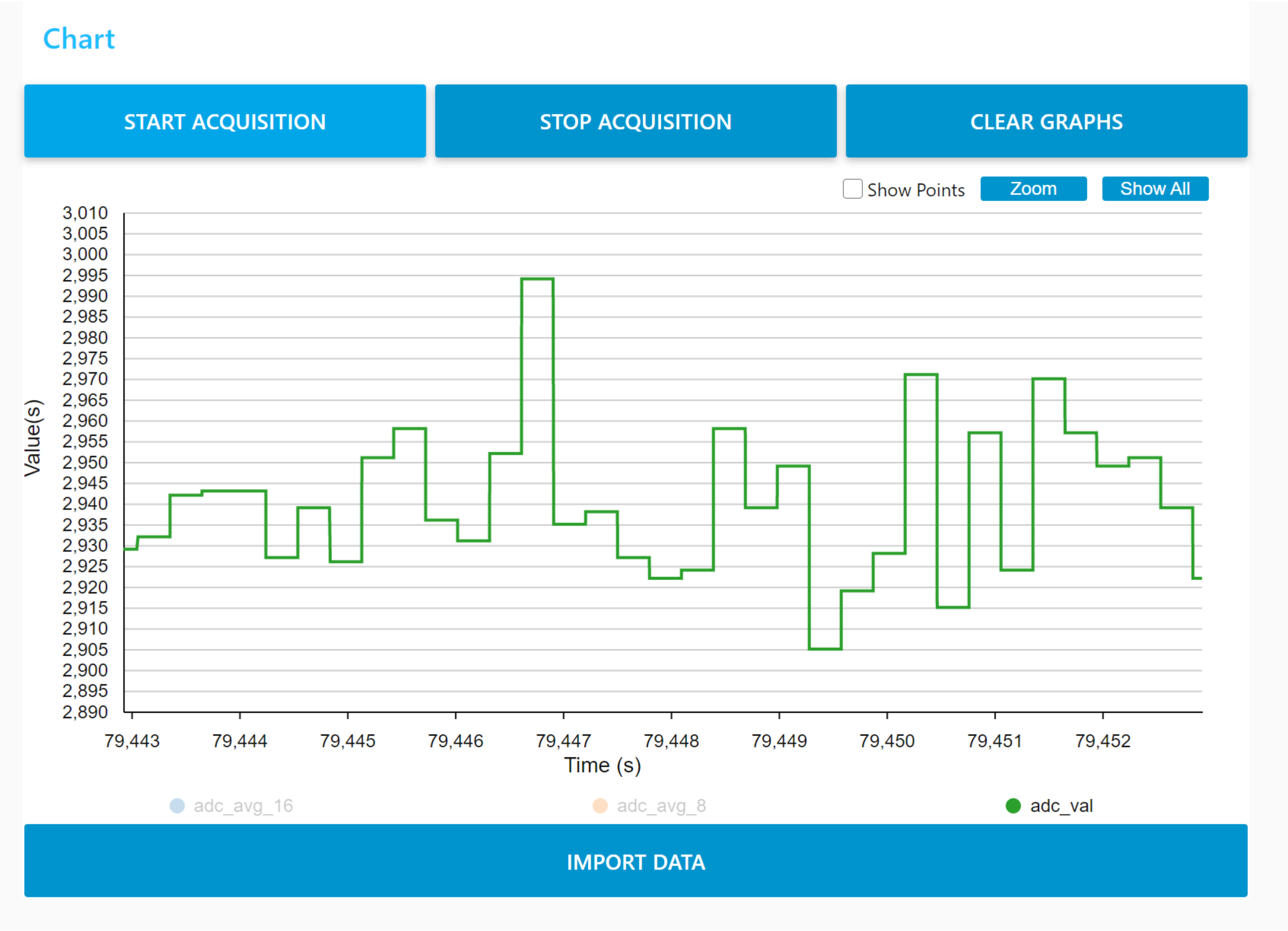


Chart
START ACQUISITION
STOP ACQUISITION
CLEAR GRAPHS
Show Points
Zoom
Show All
3,010
3,005
3,000
2,995
2,990
2,985
2,980
2,975
2,970
2,965
2,960
2,955
2,950
2,945
2,940
2,935
2,930
2,925
2,920
2,915
2,910
2,905
2,900
2,895
2,890
Value(s)
79,443
79,444
79,445
79,446
79,447
79,448
79,449
79,450
79,451
79,452
Time (s)
adc_avg_16
adc_avg_8
adc_val
IMPORT DATA

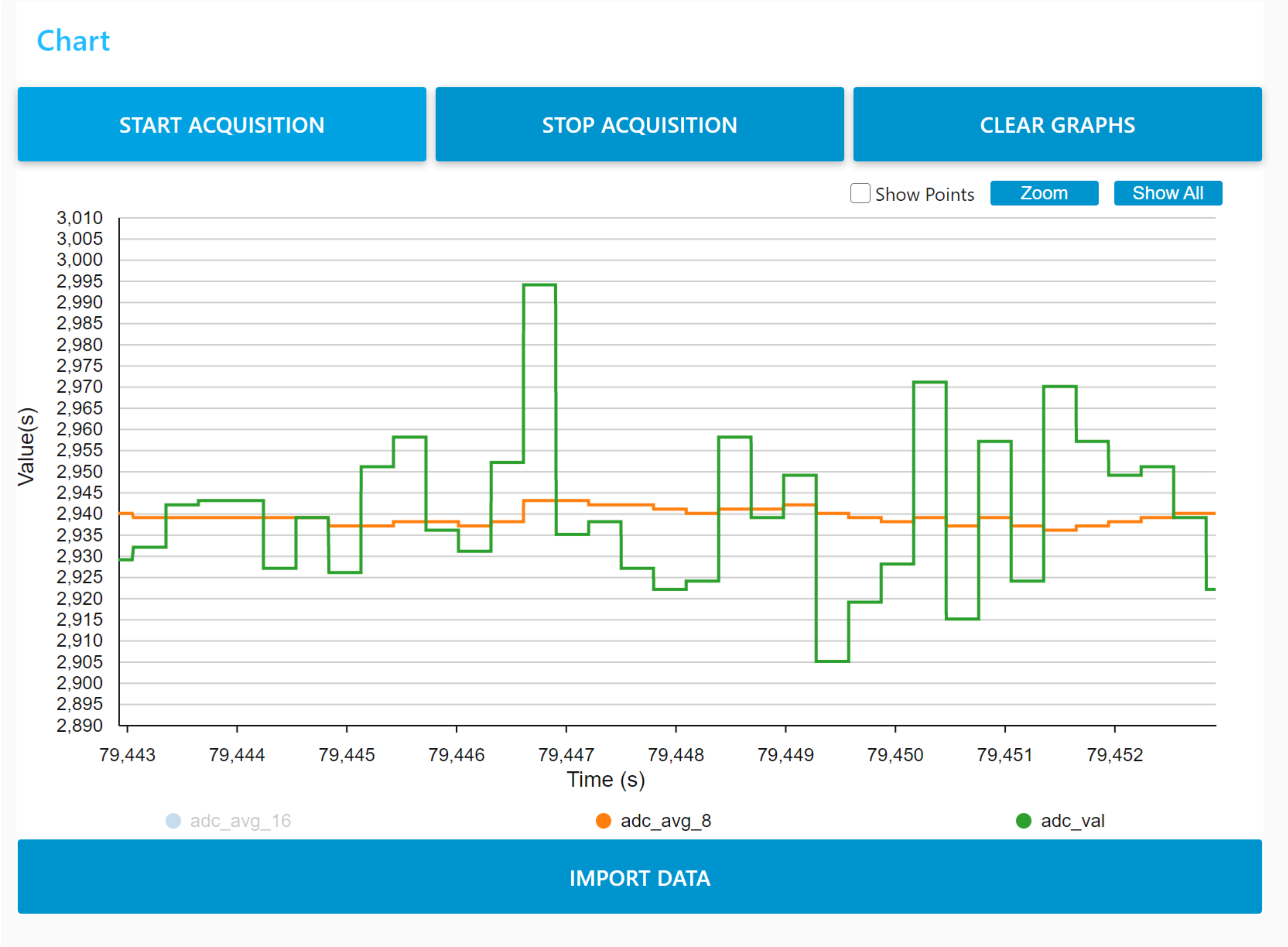


Chart
START ACQUISITION
STOP ACQUISITION
CLEAR GRAPHS
Show Points
Zoom
Show All
3,010
3,005
3,000
2,995
2,990
2,985
2,980
2,975
2,970
2,965
2,960
2,955
2,950
2,945
2,940
2,935
2,930
2,925
2,920
2,915
2,910
2,905
2,900
2,895
2,890
Value(s)
79,443
79,444
79,445
79,446
79,447
79,448
79,449
79,450
79,451
79,452
Time (s)
adc_avg_16
adc_avg_8
adc_val
IMPORT DATA

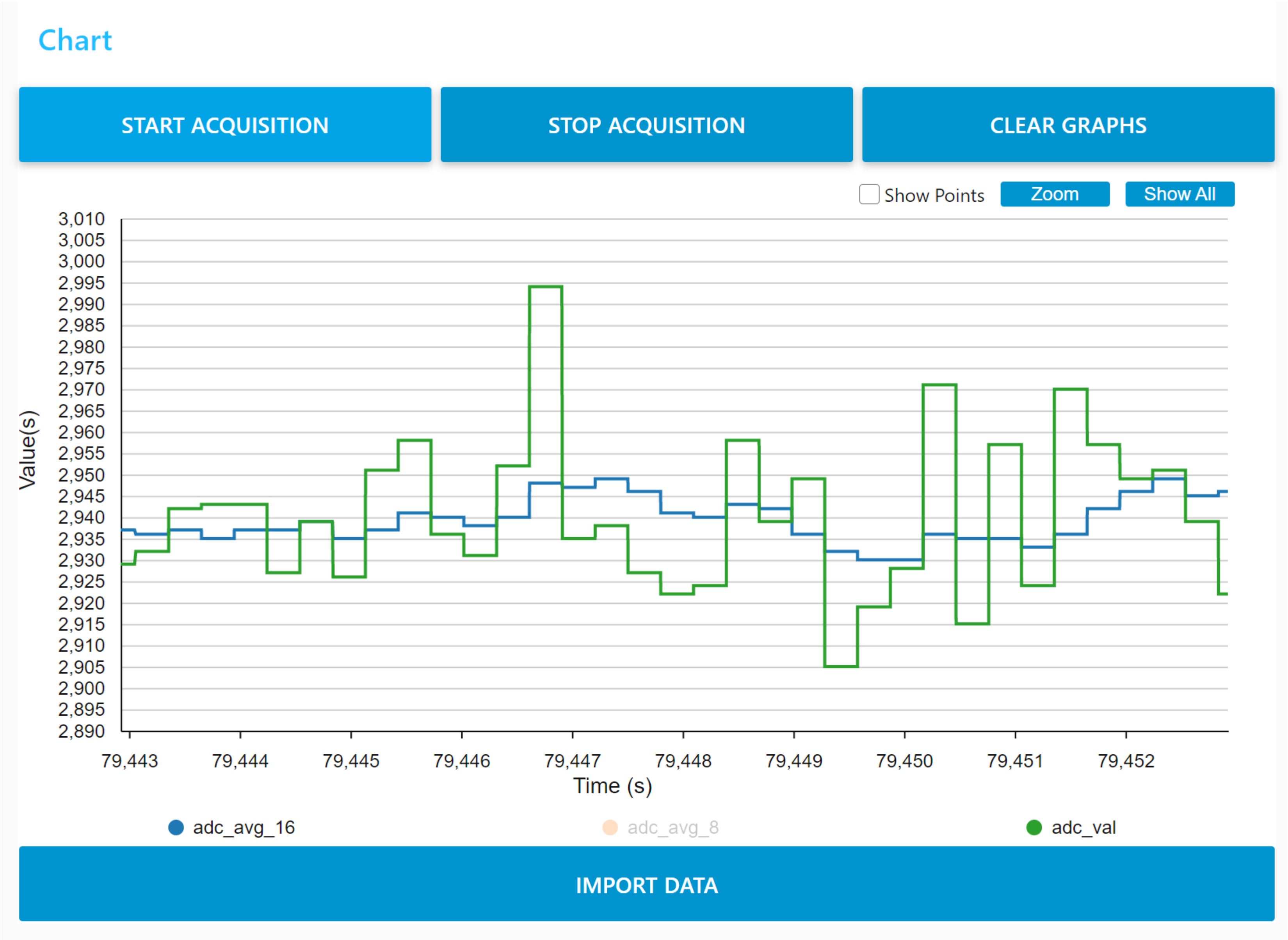


Chart
START ACQUISITION
STOP ACQUISITION
CLEAR GRAPHS
Show Points
Zoom
Show All
3,010
3,005
3,000
2,995
2,990
2,985
2,980
2,975
2,970
2,965
2,960
2,955
2,950
2,945
2,940
2,935
2,930
2,925
2,920
2,915
2,910
2,905
2,900
2,895
2,890
Value(s)
79,443
79,444
79,445
79,446
79,447
79,448
79,449
79,450
79,451
79,452
Time (s)
adc_avg_16
adc_avg_8
adc_val
IMPORT DATA

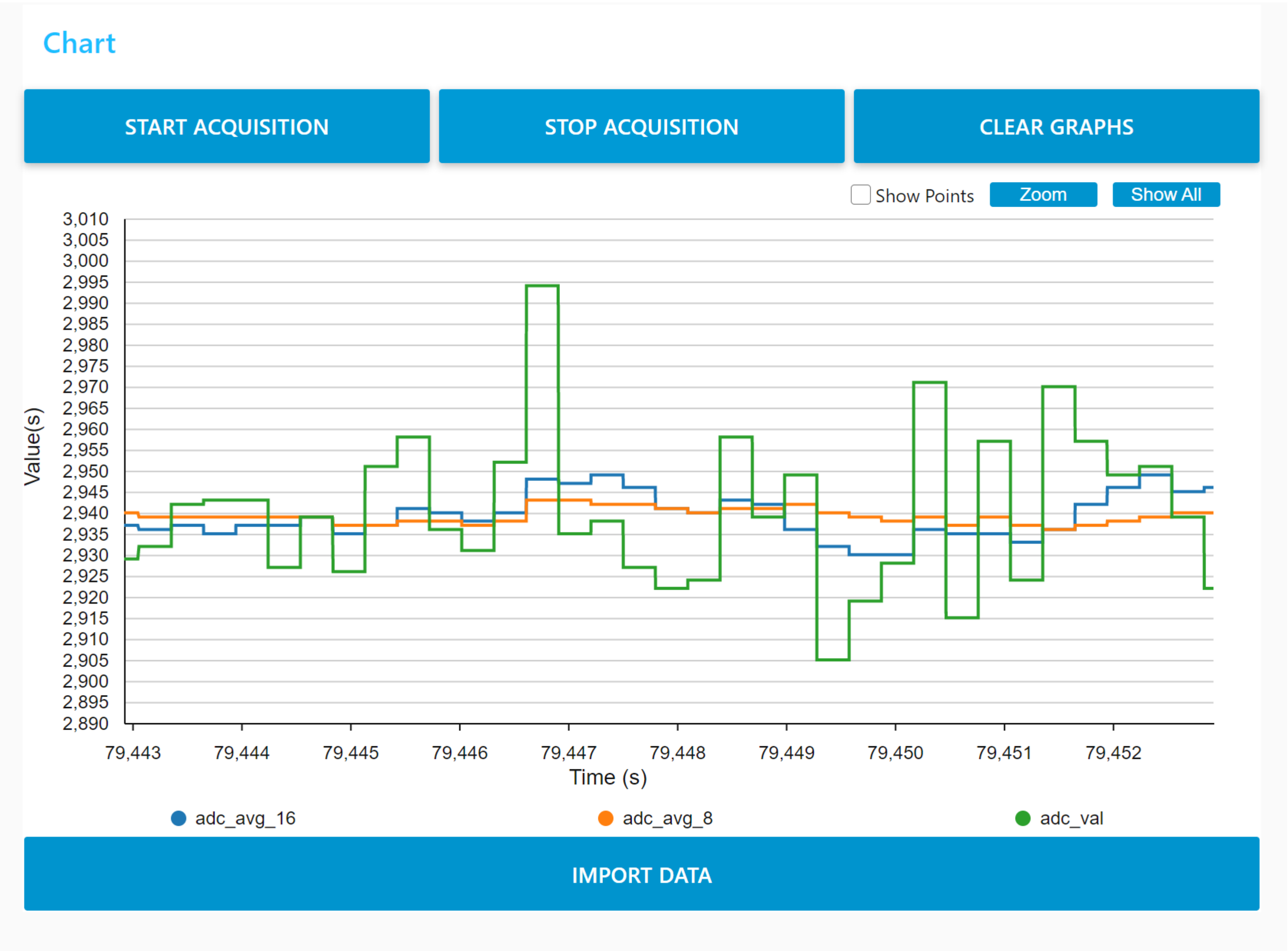


Chart
START ACQUISITION
STOP ACQUISITION
CLEAR GRAPHS
Show Points
Zoom
Show All
3,010
3,005
3,000
2,995
2,990
2,985
2,980
2,975
2,970
2,965
2,960
2,955
2,950
2,945
2,940
2,935
2,930
2,925
2,920
2,915
2,910
2,905
2,900
2,895
2,890
Value(s)
79,443
79,444
79,445
79,446
79,447
79,448
79,449
79,450
79,451
79,452
Time (s)
adc_avg_16
adc_avg_8
adc_val
IMPORT DATA

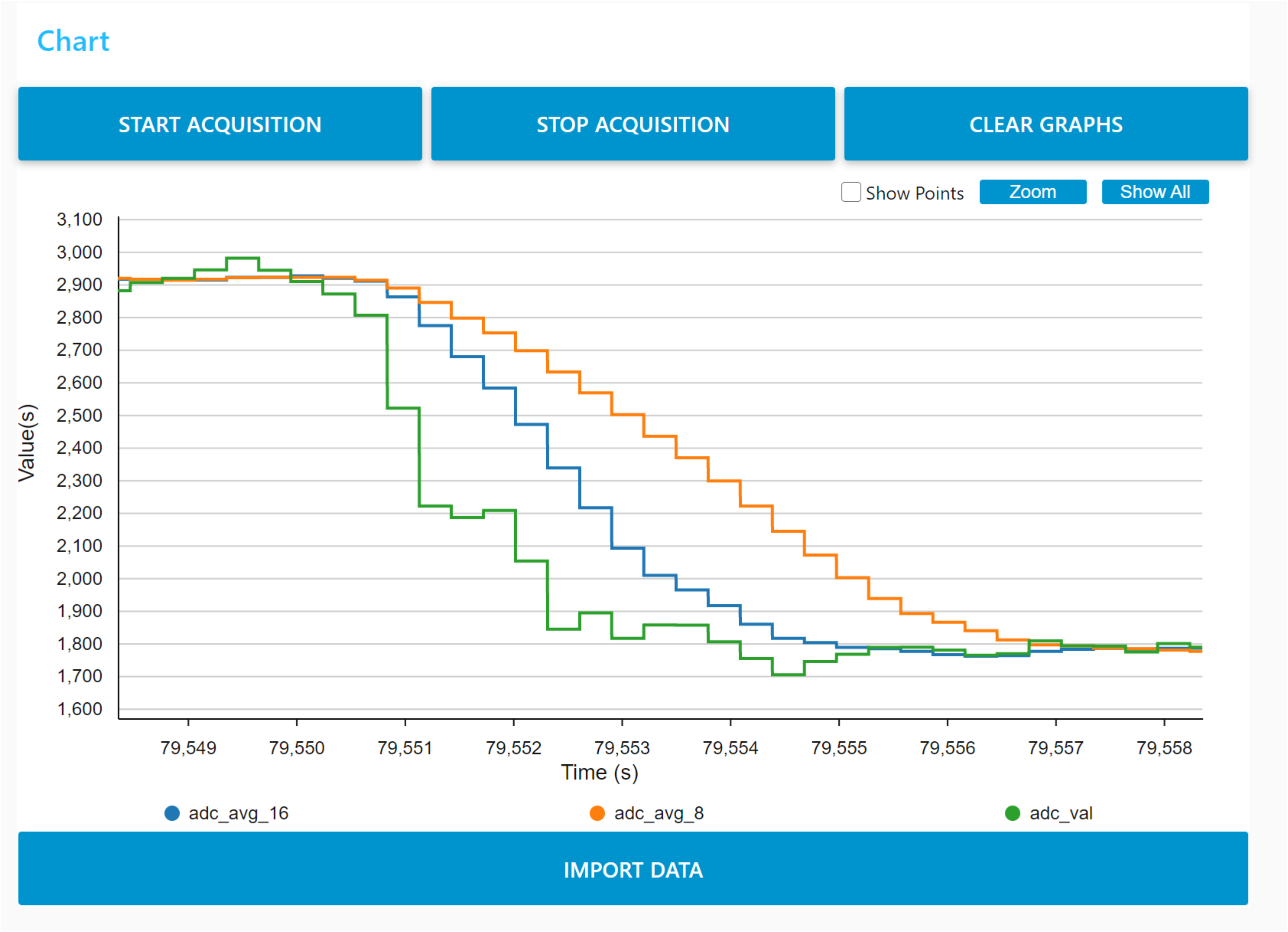


Chart
START ACQUISITION
STOP ACQUISITION
CLEAR GRAPHS
Show Points
Zoom
Show All
3,100
3,000
2,900
2,800
2,700
2,600
2,500
2,400
2,300
2,200
2,100
2,000
1,900
1,800
1,700
1,600
Value(s)
79,549
79,550
79,551
79,552
79,553
79,554
79,555
79,556
79,557
79,558
Time (s)
adc_avg_16
adc_avg_8
adc_val
IMPORT DATA

Moving Average

```c
int average_8(int x) {
    static int samples[16];
    static int i = 0;
    static int total = 0;

    /* Update the moving average */
    total += x - samples[i];
    samples[i] = x;

    /* Update the index */
    i = (i==15 ? 0 : i+1);

    return total>>4;
}
```

```c
int average_16(int x) {
    static int samples[8];
    static int i = 0;
    static int total = 0;

    /* Update the moving average */
    total += x - samples[i];
    samples[i] = x;

    /* Update the index */
    i = (i==7 ? 0 : i+1);

    return total>>3;
}
```

[Averaging or smoothing ADC values - Page 1 (eevblog.com)](#)

Summary

- Many applications using Analog voltage as input to MCU
- ADC converts analog signal to n bit digital code
- Keywords: input range, resolution, sampling rate and conversion time
- Nyquist's sampling theorem : The sampling frequency must be at least twice of the highest frequency analog input.
- ADC Data can be further processed, and displayed or stored.
- Numerous sensors have an analog output; can be directly connected to MCU ADC input.