

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

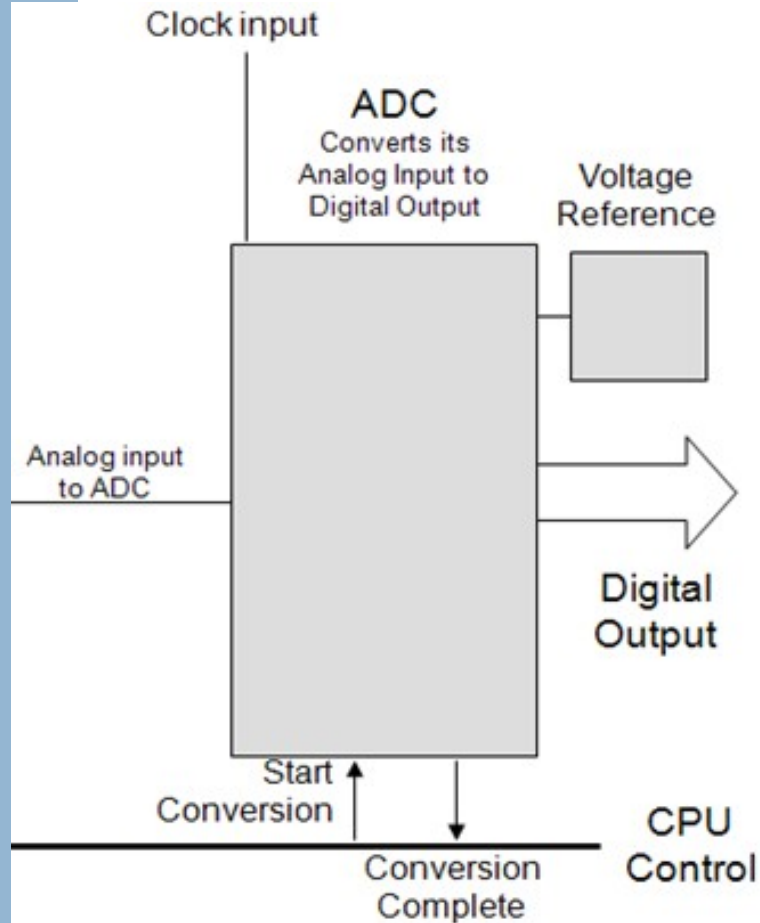
Digital : discretely valued signal, such as integers, encoded in binary

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

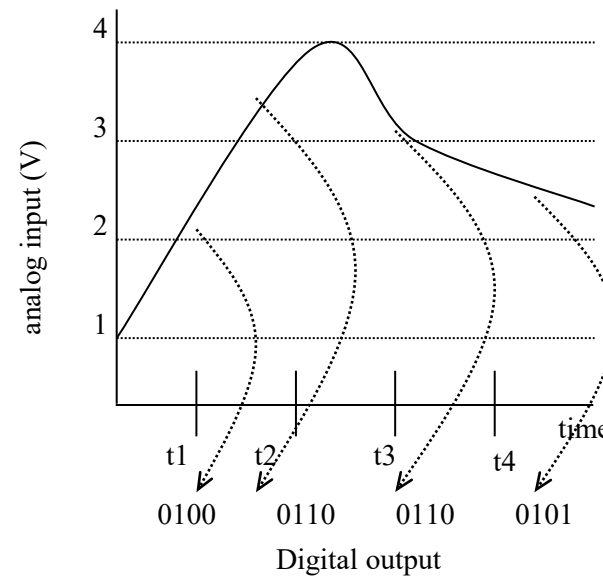


- ADC is an electronic circuit that **measures the input voltage**, and gives a binary output number **proportional** to its size.
- ADC compares an input voltage to a **reference voltage**.
- Conversion takes **time ($\mu s++$)**, so the ADC needs to **signal** when it has finished.

Analog-to-digital converters

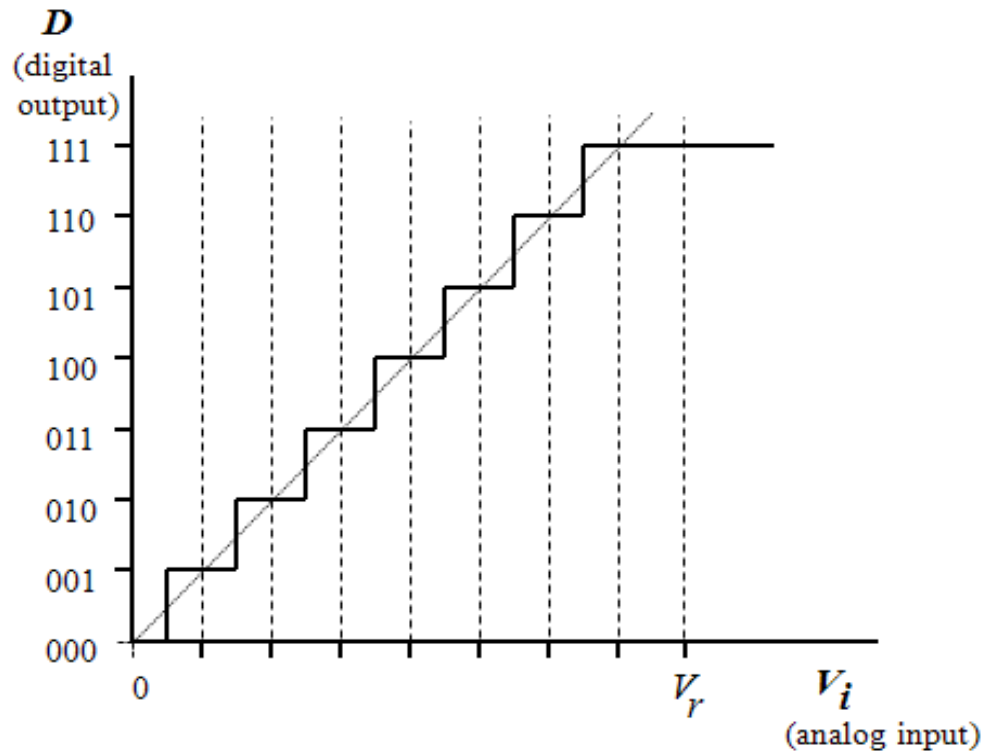
$V_{\max} = 7.5\text{V}$	1111
7.0V	1110
6.5V	1101
6.0V	1100
5.5V	1011
5.0V	1010
4.5V	1001
4.0V	1000
3.5V	0111
3.0V	0110
2.5V	0101
2.0V	0100
1.5V	0011
1.0V	0010
0.5V	0001
0V	0000

proportionality



analog to digital

Range, Resolution and Quantisation



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

- 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

Simple Equation

Assume minimum voltage of 0 V.

V_{max} = maximum voltage of the analog signal

a = analog value

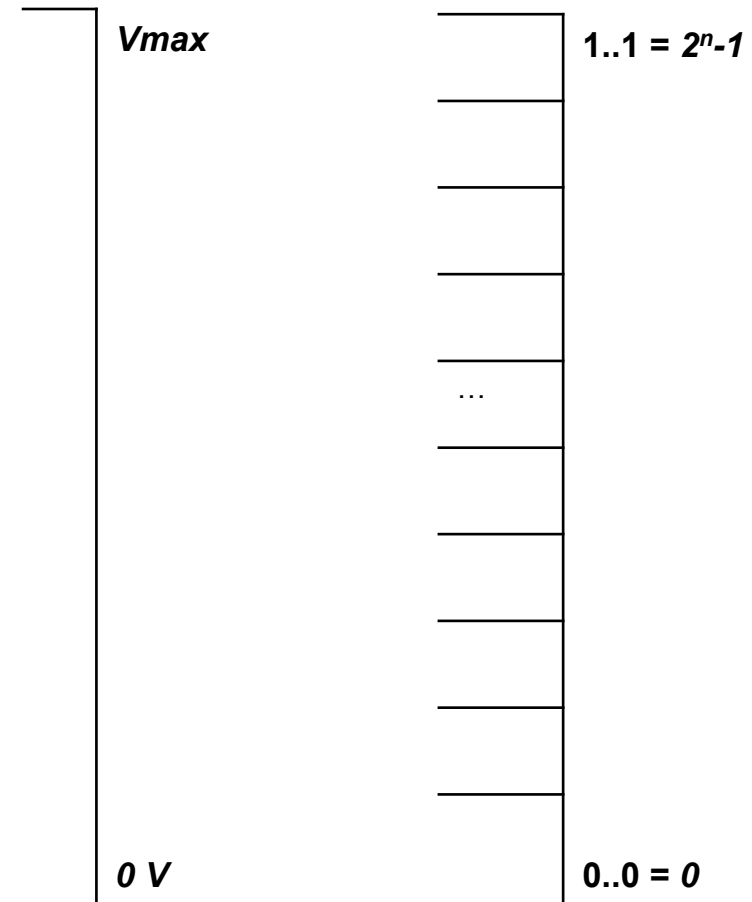
n = number of bits for digital encoding

2^n = number of digital codes

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

d = digital encoding

$$a / V_{max} = d / M$$



Resolution

Let $n = 2$

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

3 steps on the digital scale

$$d_0 = 0 = 0b00$$

$$d_{V_{max}} = 3 = 0b11$$

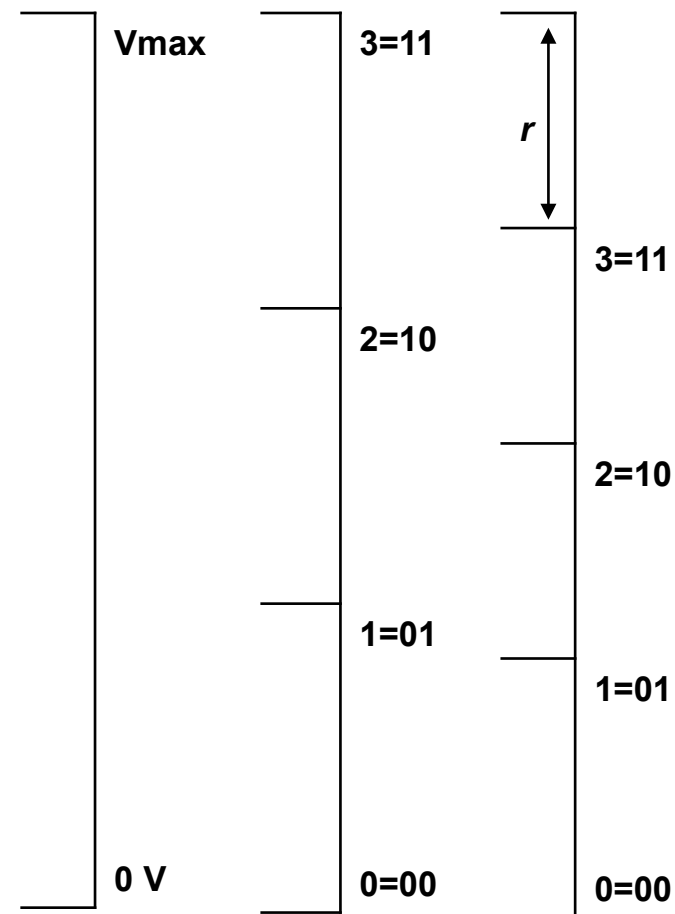
$$\underline{M = 2^n}$$

4 steps on the digital scale

$$d_0 = 0 = 0b00$$

$$d_{V_{max} - r} = 3 = 0b11 \text{ (no } d_{V_{max}} \text{)}$$

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



Resolution

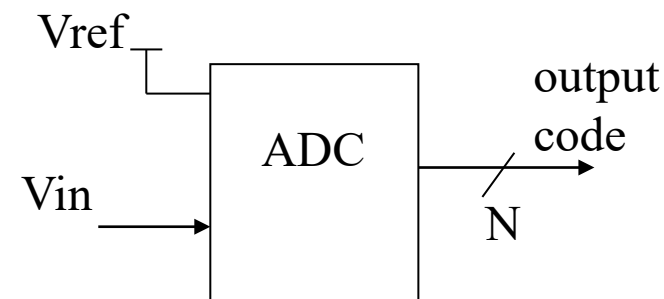
ADC: V_{in} = input voltage, V_{ref} = reference voltage

N = number of bits of precision

$$V_{in} / V_{ref} * 2^N = \text{output_code}$$

$$\text{output_code} / 2^N * V_{ref} = V_{in}$$

$$1 \text{ LSB} = V_{ref} / 2^N$$



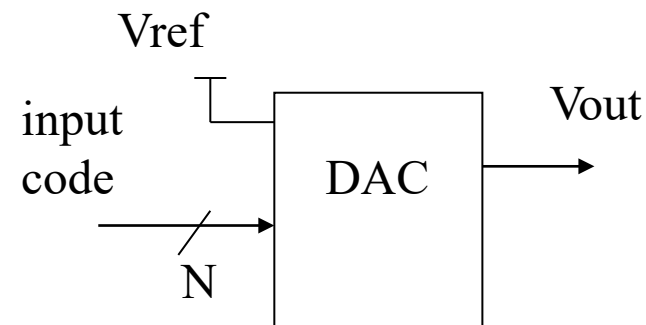
DAC: V_{out} = output voltage, V_{ref} = reference voltage,

N = number of bits of precision

$$V_{out} / V_{ref} * 2^N = \text{input_code}$$

$$\text{input_code} / 2^N * V_{ref} = V_{out}$$

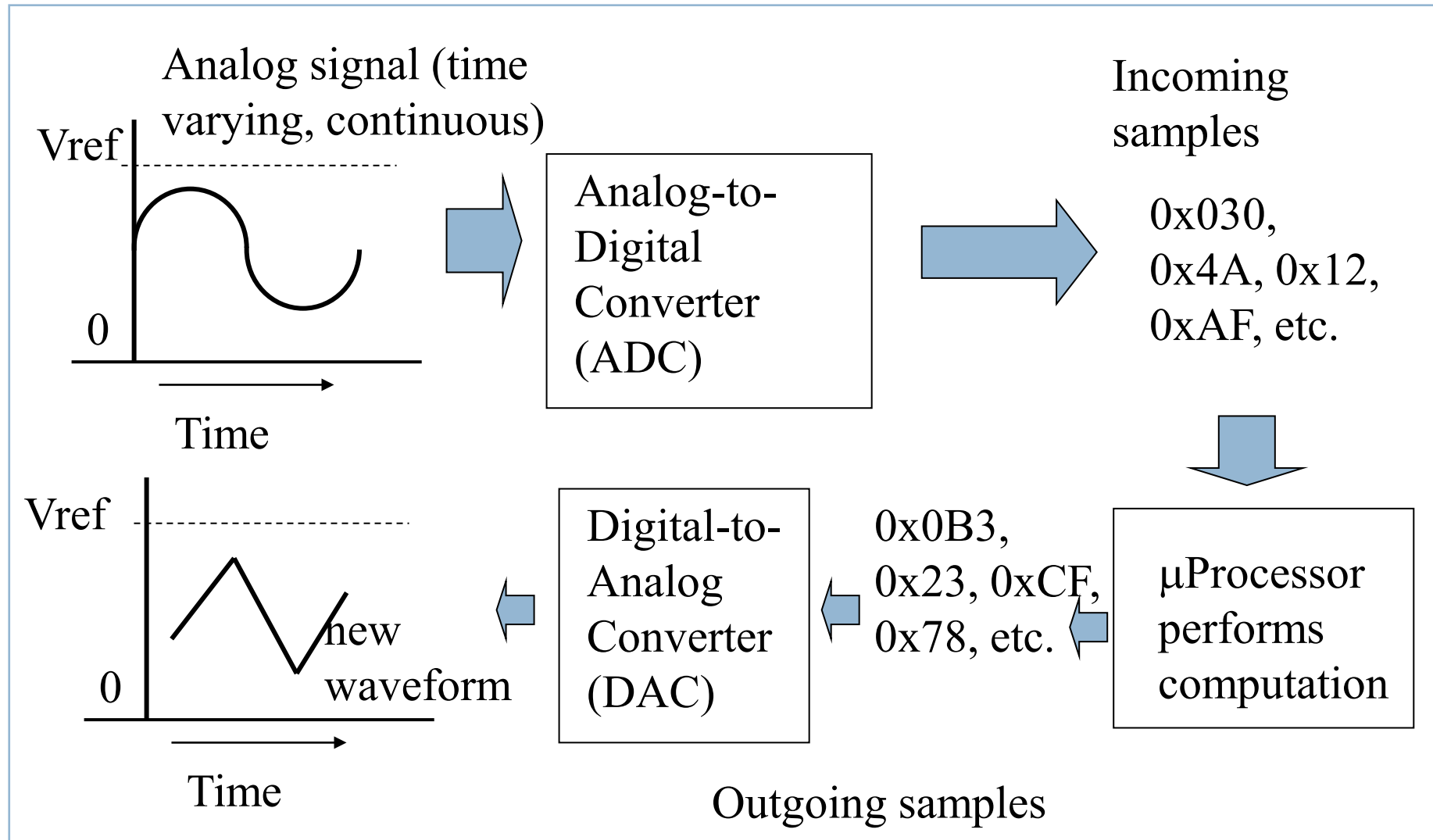
$$1 \text{ LSB} = V_{ref} / 2^N$$



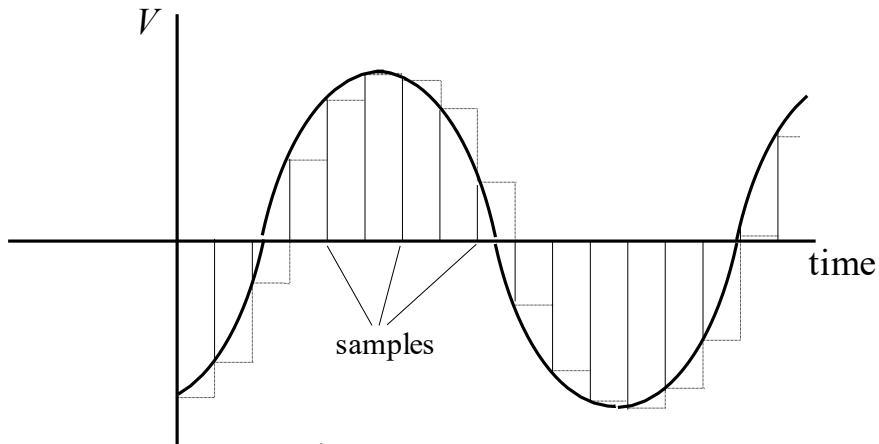
ADC Applications

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

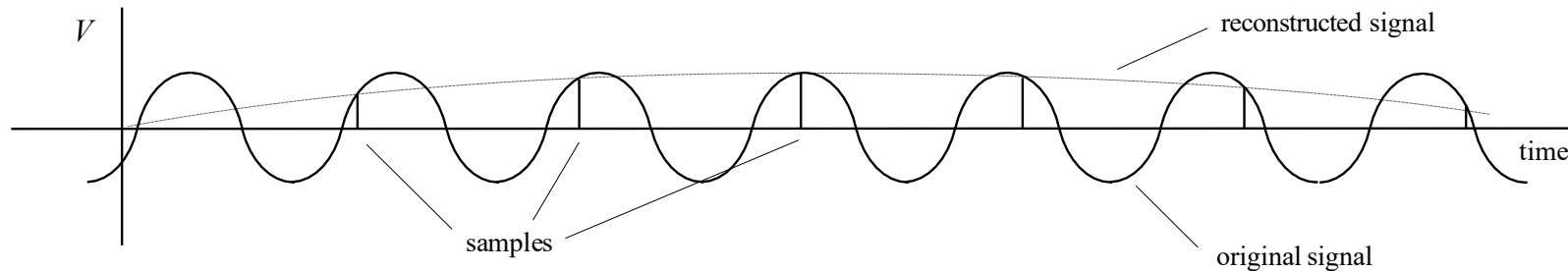
Digital Signal Processing



Sampling Frequency and Aliasing



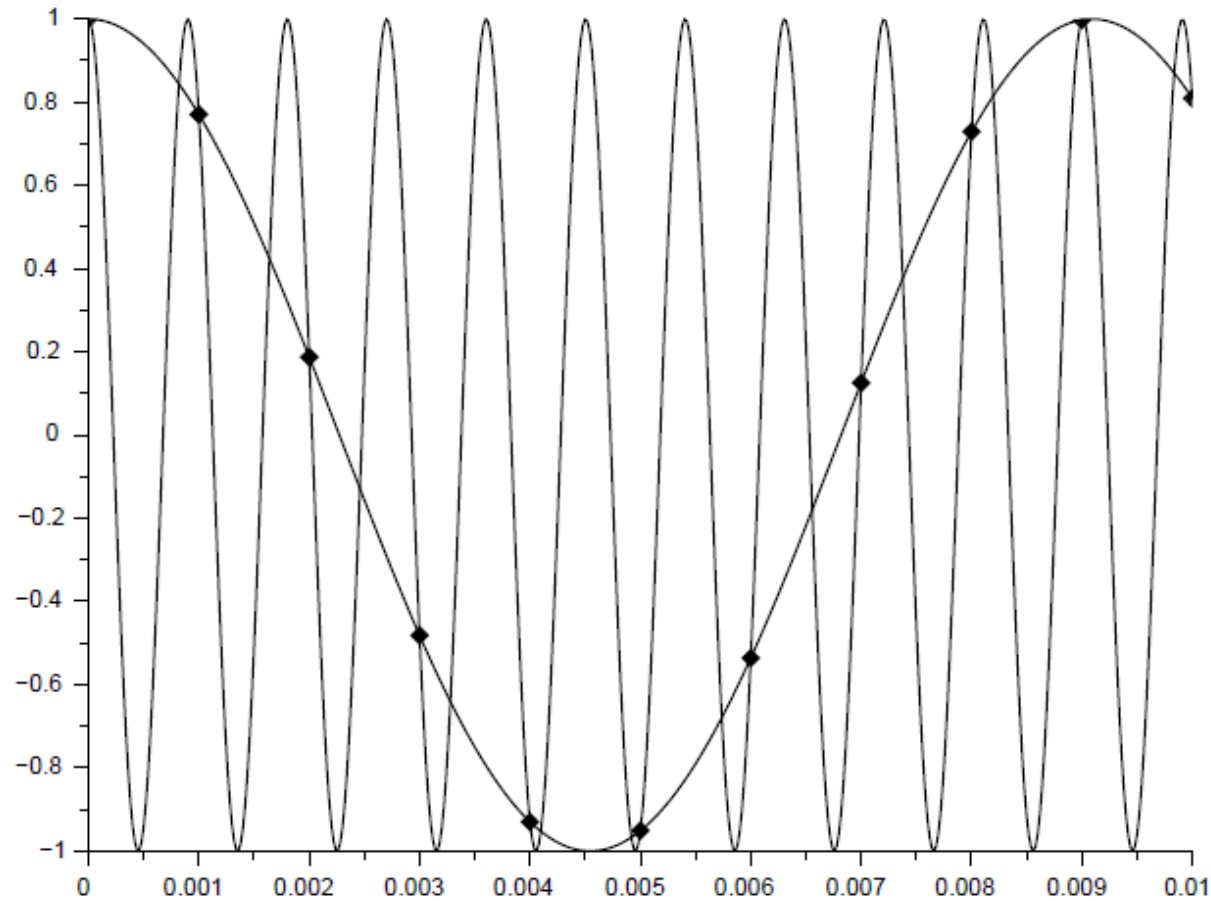
- Sample and quantise
- More samples more accurate the digital data
- sampling frequency



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 $V_{ref} = 5V$, and the 10-bit A/D output code is 0x12A, what is the ADC input voltage?

$$\begin{aligned} \text{output_code}/2^N * V_{ref} &= (0x12A)/2^{10} * 5 \text{ V} \\ &= 298/1024 * 5 \text{ V} = 1.46 \text{ V (Vin)} \end{aligned}$$

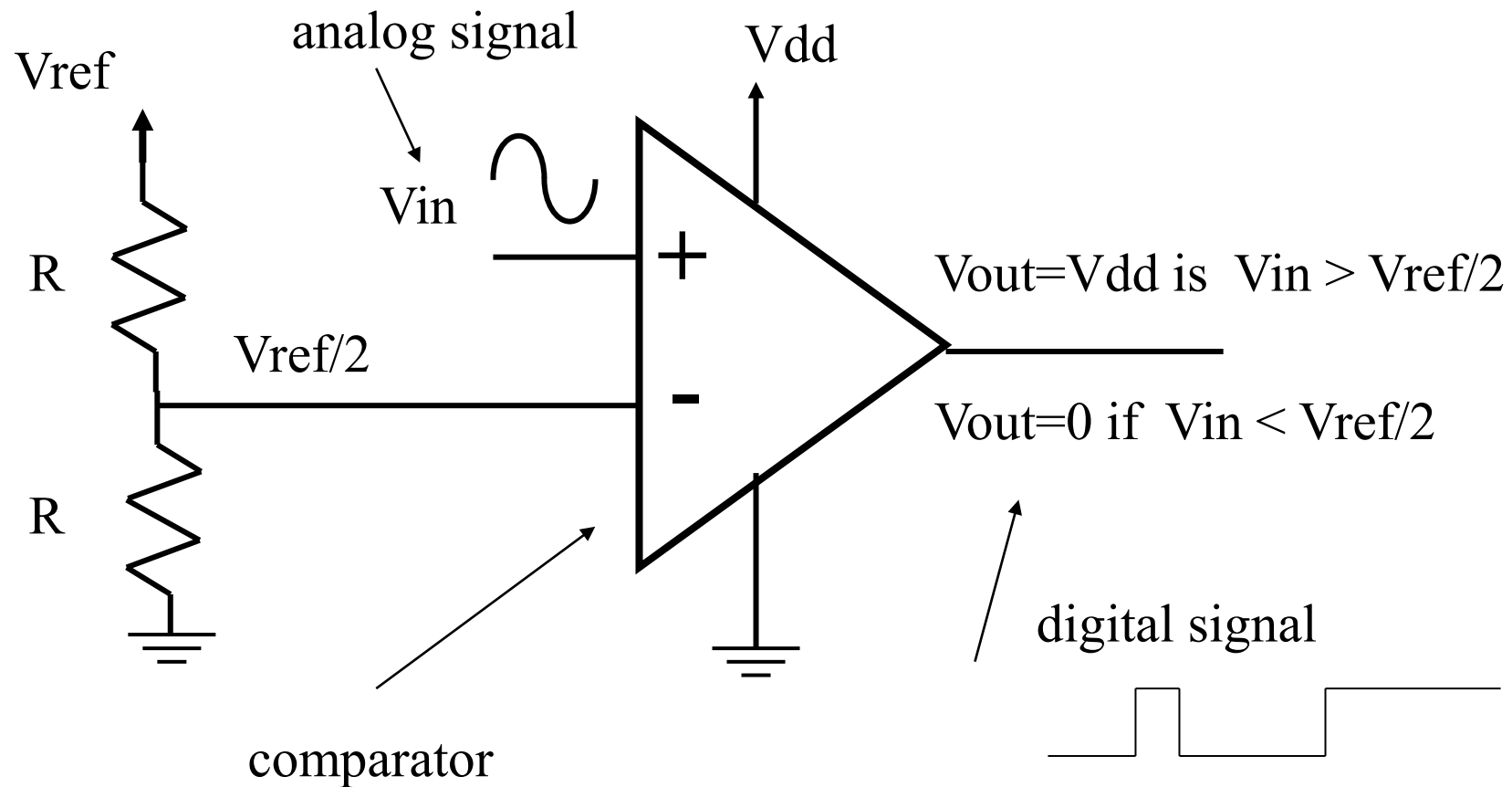
If $V_{ref} = 5V$, and the upper 8 bits of the A/D output code is 0xA9, what is the ADC input voltage?

$$\begin{aligned} \text{output_code}/2^N * V_{ref} &= (0xA9)/2^8 * 5 \text{ V} \\ &= 169/256 * 5 \text{ V} = 3.3 \text{ V (Vin)} \end{aligned}$$

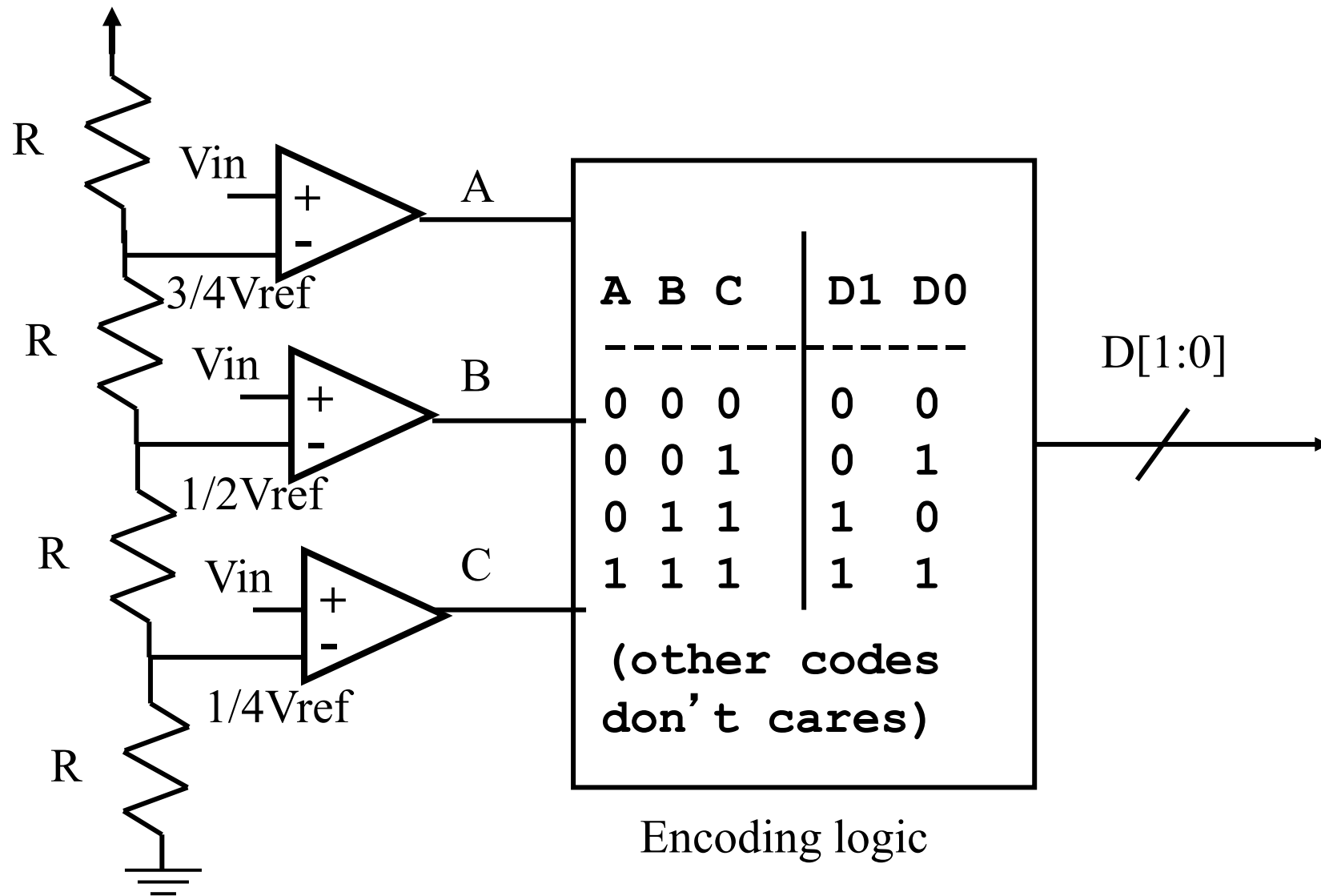
If $V_{ref} = 4V$, and the A/D input voltage is 2.35 V, what is the ADC output code, upper 8-bits?

$$\begin{aligned} V_{in}/V_{ref} * 2^N &= 2.35 \text{ V} / 4 \text{ V} * 2^8 \\ &= .5875 * 256 = 150.4 = 150 = 0x96 \end{aligned}$$

A 1-bit ADC



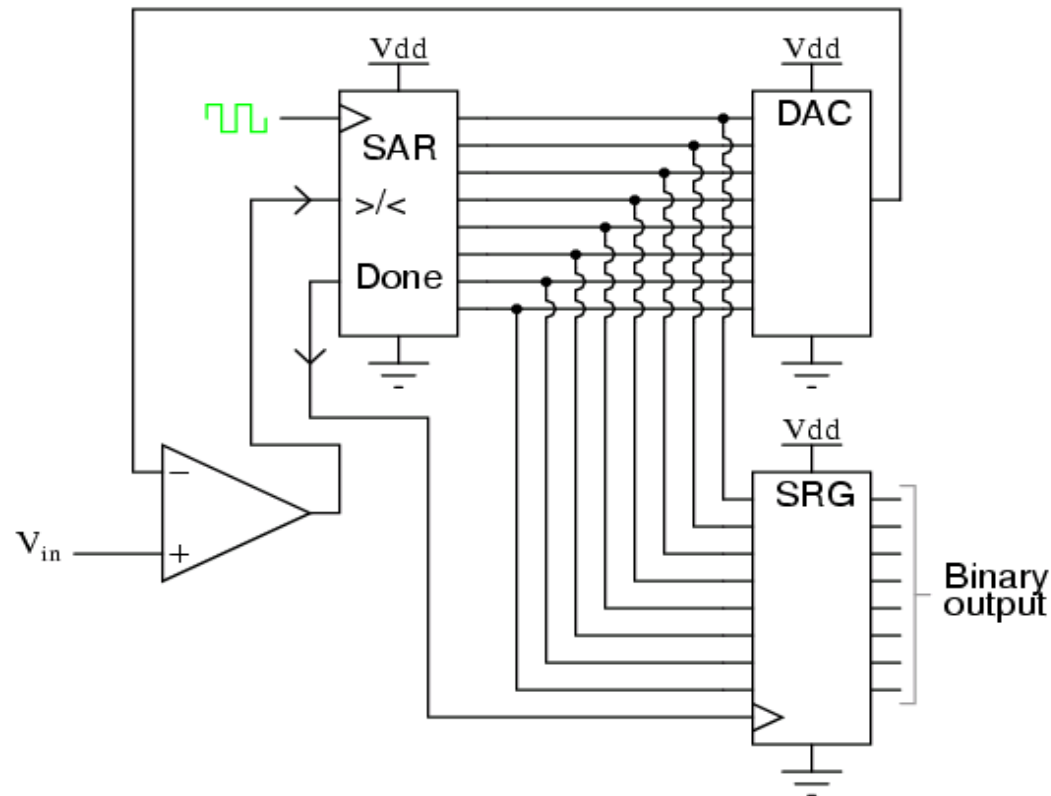
A 2-bit ADC



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 $V_{ref}/2$.

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

If MSB = 1, then V_{in} between V_{ref} and $V_{ref}/2$, so set DAC to produce $3/4 V_{ref}$.

If MSB=0, then V_{in} between $V_{ref}/2$ and 0, so set DAC to $1/2 V_{ref}$.

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 $M = 2^n - 1$

$$a / V_{max} = d / M$$

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

$$d = 85 \text{ or binary } 01010101$$

ADC using successive approximation

Step 1-4: determine bits 0-3

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

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

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

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

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

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

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

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

ADC using successive approximation

Step 5-8: Determine bits 4-7

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

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

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

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

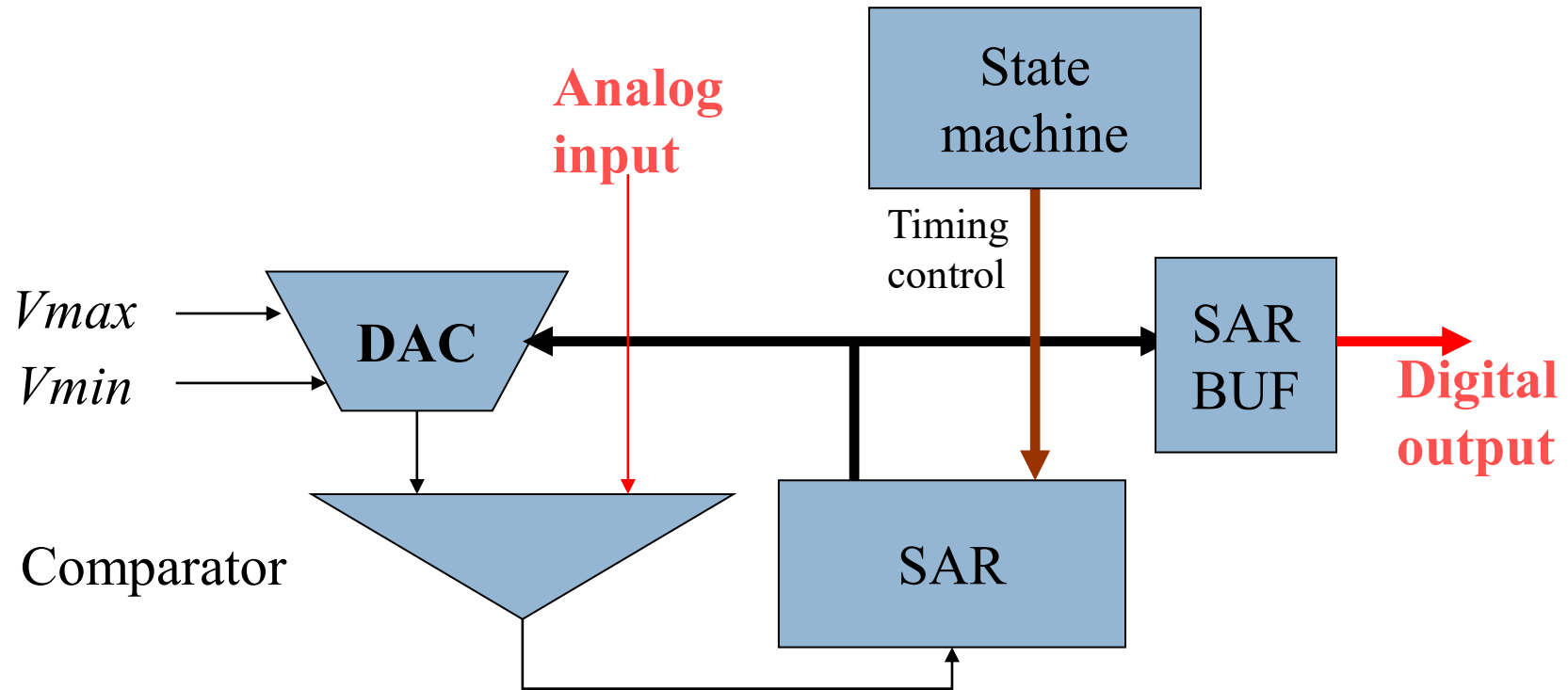
0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

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: $V_{REF-} \leq V_{IN} \leq V_{REF+}$

Examples of Conversion Mode

Figure 4. Single-channel, continuous conversion mode

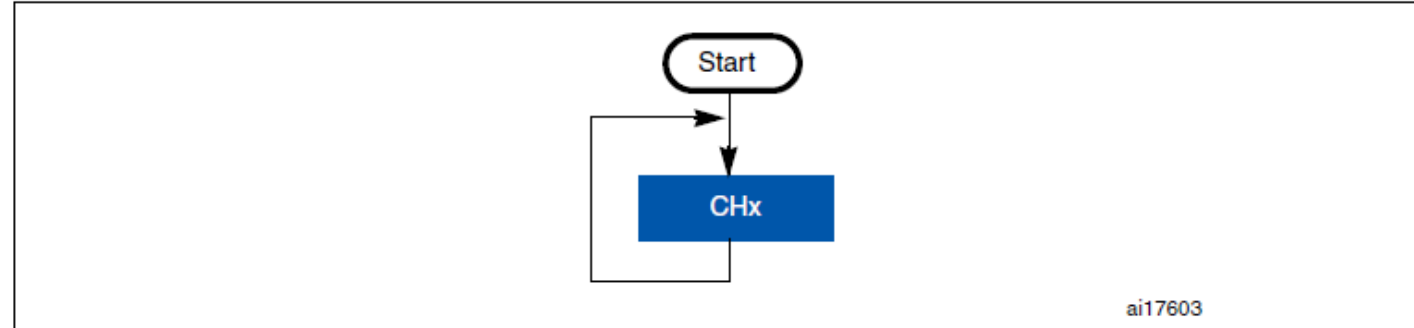
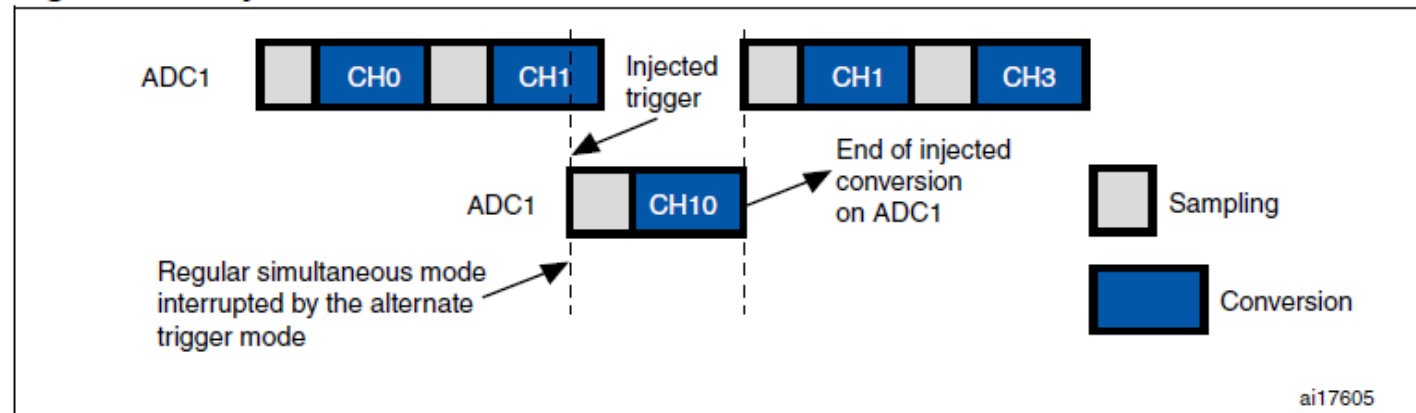
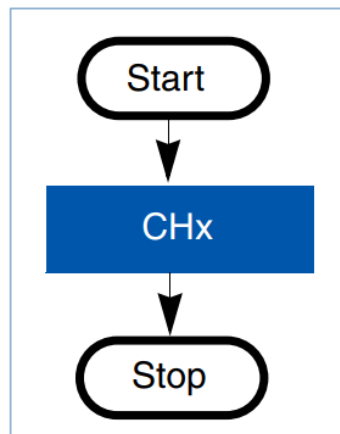


Figure 6. Injected conversion mode

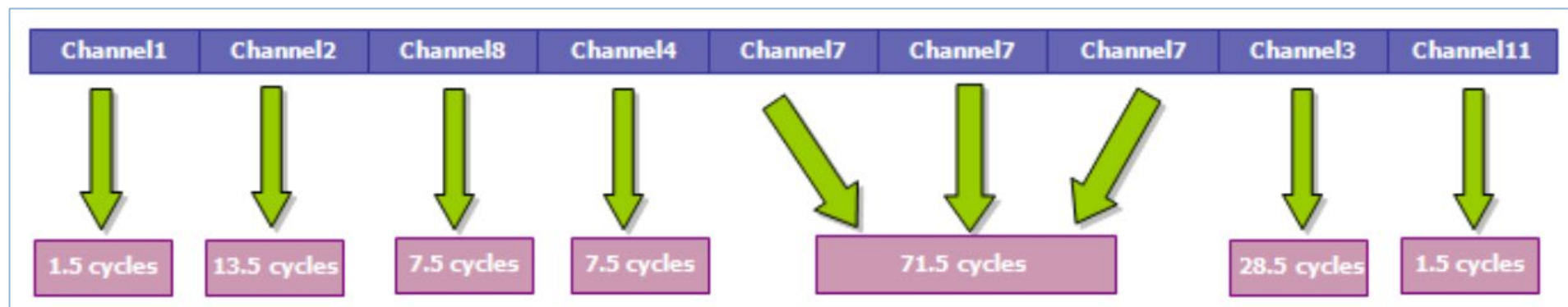
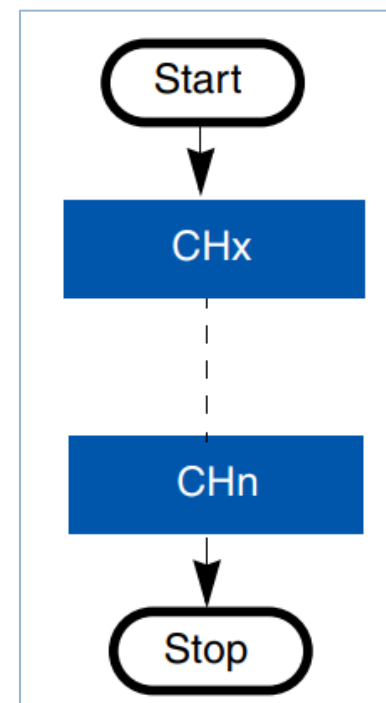


Single Conversion

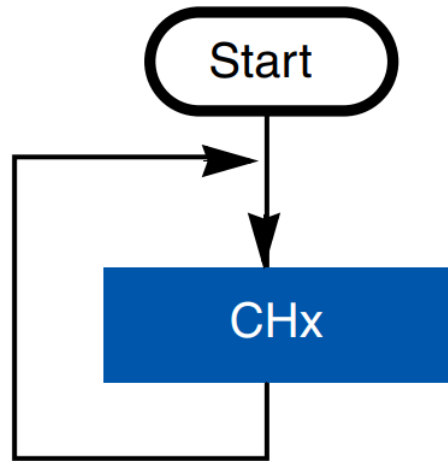
Single Channel



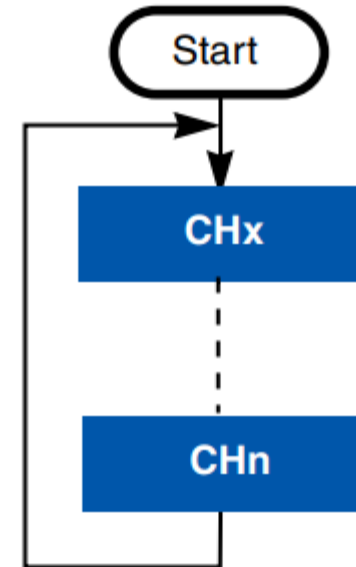
Multichannel



Continuous Mode



Single Channel



Multichannel

Other modes

Figure 6. Injected conversion mode

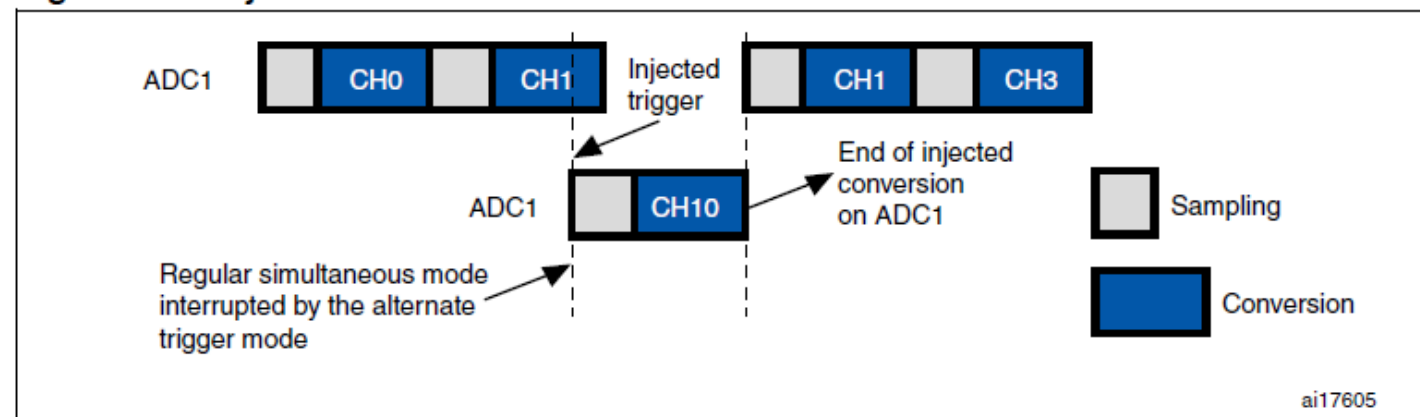
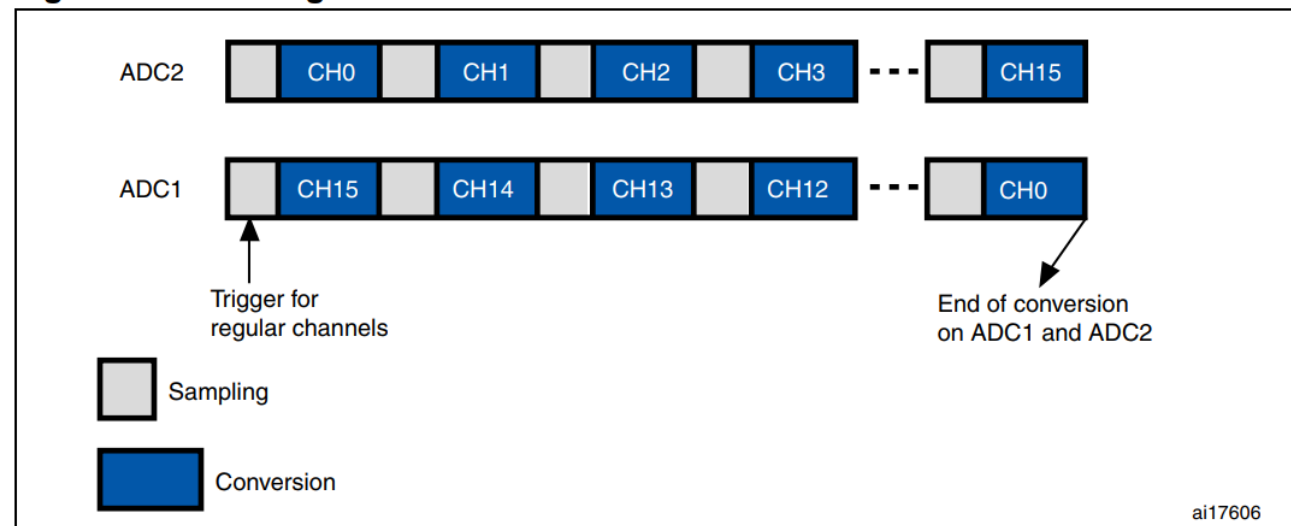
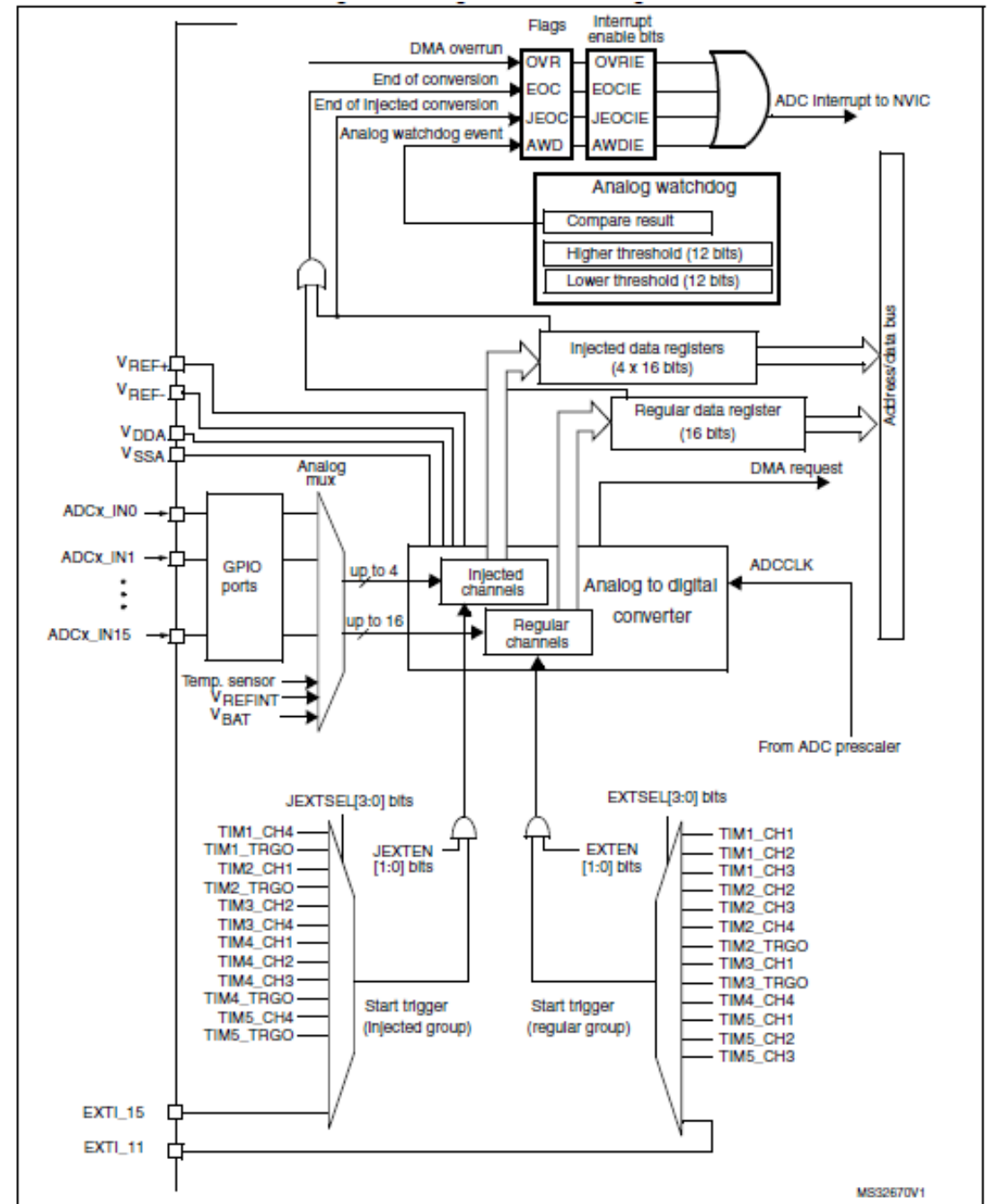
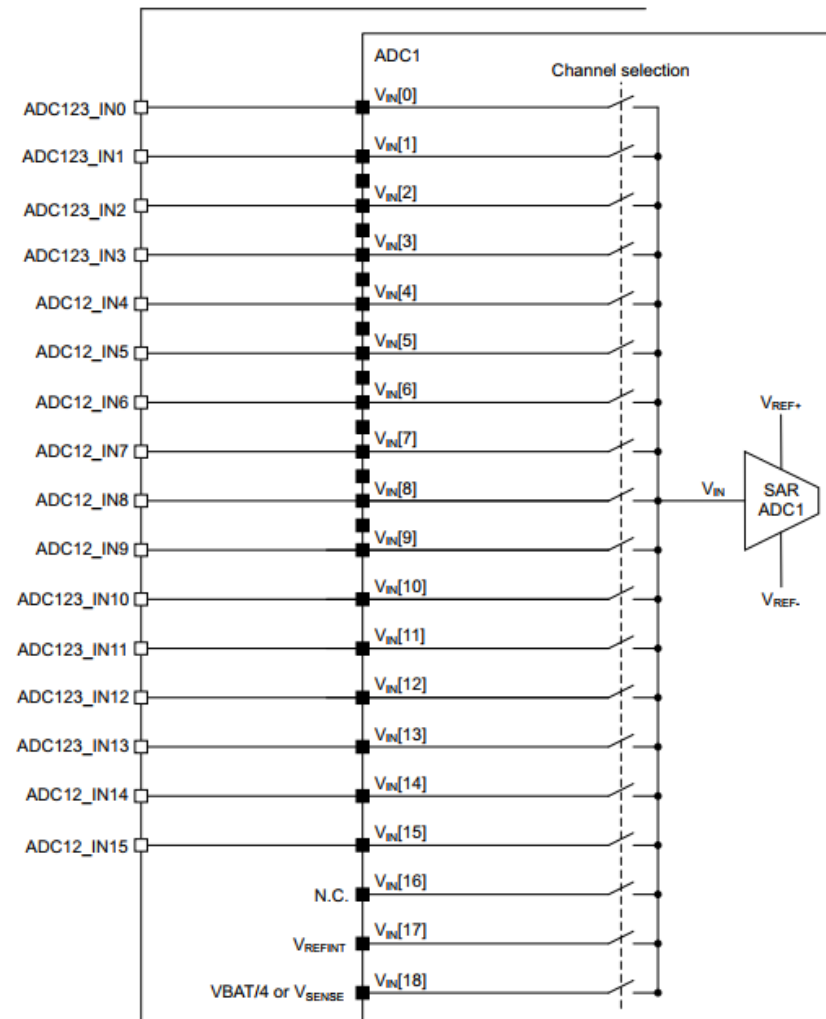


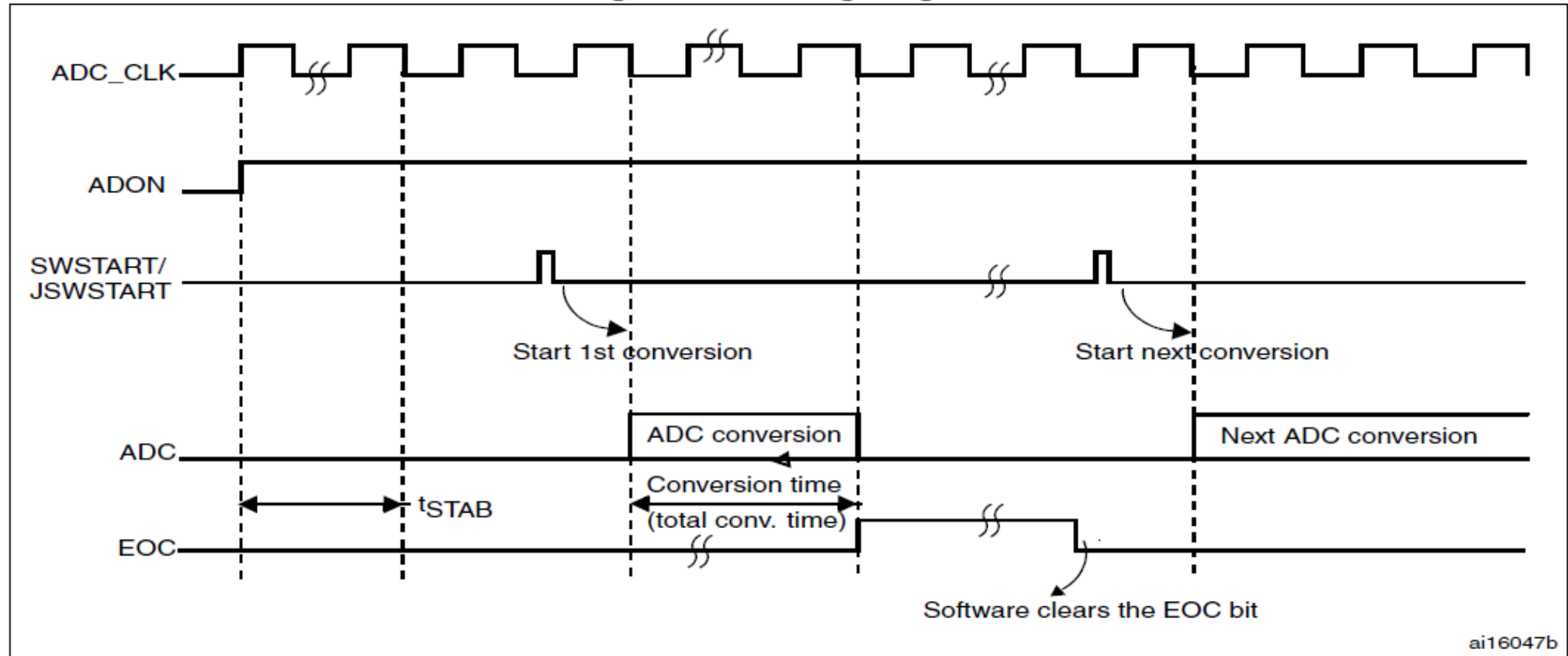
Figure 7. Dual regular simultaneous mode



ADC Block Diagram



ADC Timing Diagram



Data Alignment

Figure 35. Right alignment of 12-bit data

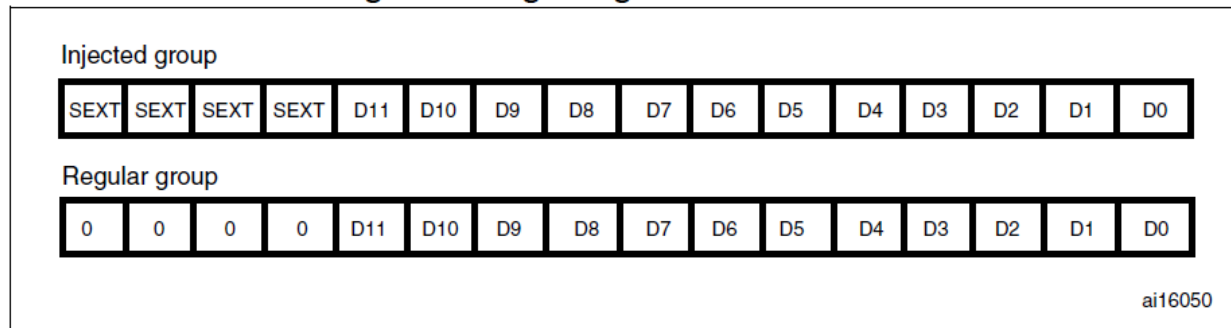


Figure 36. Left alignment of 12-bit data

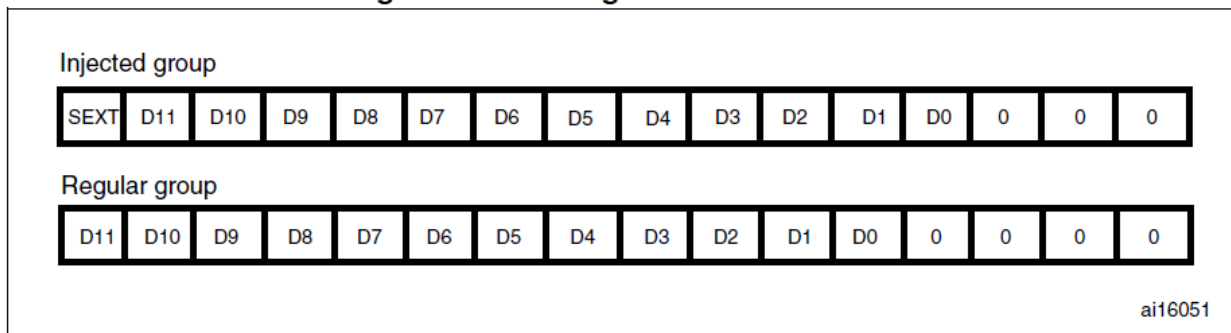
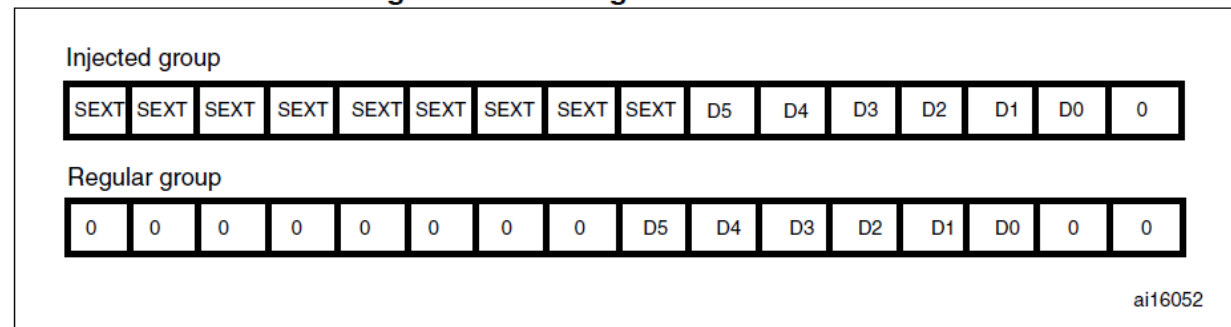


Figure 37. Left alignment of 6-bit data



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

The image shows the STM32CubeMX Pinout & Configuration window. On the left, the pinout table shows the connection for ADC1_IN10 to PC0. The main window is divided into two tabs: Pinout & Configuration and Clock Configuration. The Pinout & Configuration tab is active, showing the ADC1 Mode and Configuration settings. The Mode section shows IN10 selected. The Configuration section shows the following settings:

- Reset Configuration: (2)
- Parameter Settings: (1)
- User Constants: (3)
- NVIC Settings: (4)
- DMA Settings: (4)
- GPIO Settings: (4)

The configuration parameters are as follows:

Parameter	Value
Mode	Independent mode
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled (3)
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of all conversions (4)
Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 10
Sampling Time	3 Cycles
Number Of Conversions	0

```
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();
    }
}
```

```
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	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• Converted value
Notes	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.• Timeout: Timeout value in millisecond.
Return values	<ul style="list-style-type: none">• HAL: status
Notes	<ul style="list-style-type: none">• 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

PA3

PC0

ADC1 Mode and Configuration

Mode

- ☐ IN0
- ☒ IN1
- ☒ IN2
- ☒ IN3
- ☐ IN4
- ☐ IN5
- ☐ IN6
- ☒ IN7
- ☐ IN8
- ☐ IN9
- ☒ IN10
- ☒ IN11
- ☐ IN12
- ☐ IN13
- ☒ IN14
- ☒ IN15
- ☐ Temperature Sensor Channel
- ☐ Vrefint Channel
- ☐ Vbat Channel
- ☐ External-Trigger-for-Regular-conversion

Configuration

Reset Configuration

✓ NVIC Settings ✓ DMA Settings ✓ GPIO Settings

✓ Parameter Settings ✓ User Constants

Configure the below parameters :

Search (Ctrl+F) ⓘ

✓ ADCs_Common_Settings

Mode Independent mode

✓ ADC_Settings

Clock Prescaler PCLK2 divided by 8

Resolution 12 bits (15 ADC Clock cycles)

Data Alignment Right alignment

Scan Conversion Mode Enabled

Continuous Conversion Mode Enabled

Discontinuous Conversion Mode Disabled

DMA Continuous Requests Enabled

End Of Conversion Selection EOC flag at the end of single channel conversion

✓ ADC_Regular_ConversionMode

Number Of Conversion 2

External Trigger Conversion Source Regular Conversion launched by software

External Trigger Conversion Edge None

✓ Rank 1

Channel Channel 3

Sampling Time 15 Cycles

✓ Rank 2

Channel Channel 10

Sampling Time 15 Cycles

✓ ADC_Injected_ConversionMode

Number Of Conversions 0

✓ WatchDog

Enable Analog WatchDog Mode ☐

Direct Memory Access Controller Config

The screenshot shows the STM32CubeMX software interface for configuring the DMA controller. The interface is divided into several sections:

- Pinout & Configuration**: The left sidebar contains a search bar, a "Categories" dropdown set to "A->Z", and a "System Core" section. Under "System Core", the "DMA" option is selected and highlighted with a red circle labeled **1**. Other options include CORTEX_M7, GPIO, IWDG, NVIC, RCC, SYS, and WWDG. Below this is an "Analog" section with ADC1, ADC2, ADC3, and DAC, and a "Timers" section.
- Clock Configuration**: The top middle tab.
- Project Manager**: The top right tab.
- DMA Mode and Configuration**: The main configuration area. It has a sub-section "Configuration" with tabs for DMA1, DMA2, and MemToMem. DMA2 is selected and highlighted with a red circle labeled **2**. Below this is a table with columns: DMA Request, Stream, Direction, and Priority. A row is configured with DMA Request "ADC1" (circled with a red circle labeled **4**), Stream "DMA2 Stream 0", Direction "Peripheral To Memory", and Priority "Low".
- DMA Request Settings**: A section below the table with "Add" and "Delete" buttons (circled with a red circle labeled **3**). It contains settings for the selected DMA request:
 - Mode**: Set to "Circular" (circled with a red circle labeled **5**).
 - Increment Address**: A checkbox that is unchecked.
 - Data Width**: Set to "Word" (circled with a red circle labeled **6**).
 - Burst Size**: A dropdown menu.
 - Peripheral** and **Memory** sections: The "Memory" section has a checked checkbox, while the "Peripheral" section is empty.

DMA Global Interrupt

NVIC Mode and Configuration			
Configuration			
<input checked="" type="checkbox"/> NVIC	<input checked="" type="checkbox"/> Code generation		
Priority Group	2 bits for pre-emption priority 2 bits f. 1	<input type="checkbox"/> Sort by Preemption Priority and S	
Search	Search (Ctrl+F)	<input type="checkbox"/> Show only enabled interrupts	
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Prio
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
ADC1, ADC2 and ADC3 global interrupts	<input type="checkbox"/>	0	0
USART3 global interrupt	<input type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input type="checkbox"/>	0	0
<u>DMA2 stream0 global interrupt</u>	<input checked="" type="checkbox"/>	1 2	0
USB On The Go FS global interrupt	<input type="checkbox"/>	0	0
FPU global interrupt	<input type="checkbox"/>	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	<ul style="list-style-type: none">• 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

DMA Request Settings

	Peripheral	Memory
Mode <input type="text" value="Circular"/>	Increment Address <input type="checkbox"/>	<input checked="" type="checkbox"/>
Use Fifo <input checked="" type="checkbox"/> Threshold <input type="text" value="Full"/>	Data Width <input type="text" value="Word"/>	<input type="text" value="Word"/>
	Burst Size <input type="text" value="Single"/>	<input type="text" value="Single"/>

Configuration

Reset Configuration

✓ NVIC Settings ✓ DMA Settings ✓ GPIO Settings

✓ Parameter Settings ✓ User Constants

Configure the below parameters :

Search (Ctrl+F) ⏪ ⏩ ⓘ

▼ ADCs_Common_Settings

Mode Independent mode

▼ ADC_Settings

Clock Prescaler PCLK2 divided by 8

Resolution 12 bits (15 ADC Clock cycles)

Data Alignment Right alignment

Scan Conversion Mode Enabled

Continuous Conversion Mode Enabled

Discontinuous Conversion Mode Disabled

DMA Continuous Requests Enabled

End Of Conversion Selection EOC flag at the end of single channel conversion

▼ ADC_Regular_ConversionMode

Number Of Conversion 2

External Trigger Conversion Source Regular Conversion launched by software

External Trigger Conversion Edge None

▼ Rank 1

Channel Channel 3

Sampling Time 15 Cycles

▼ Rank 2

Channel Channel 10

Sampling Time 15 Cycles

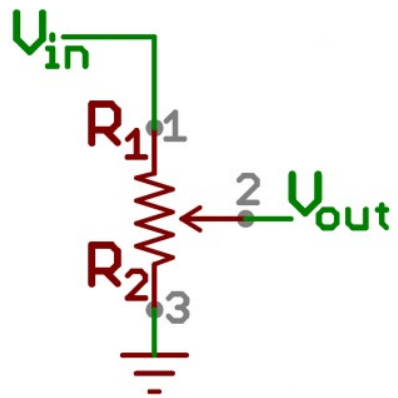
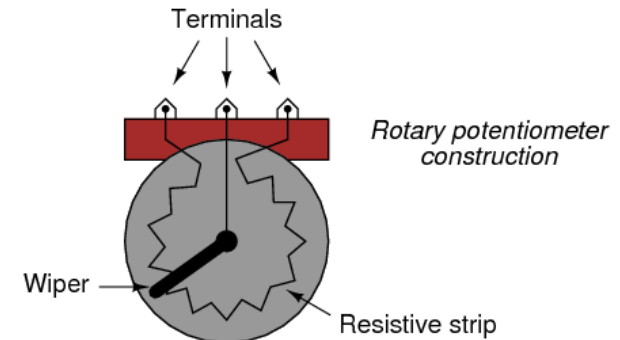
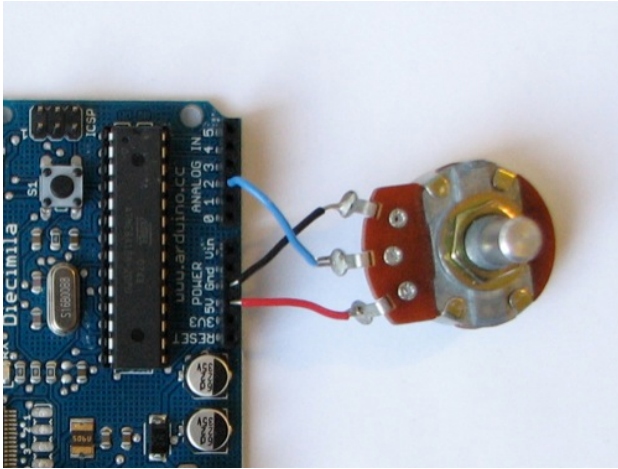
▼ ADC_Injected_ConversionMode

Number Of Conversions 0

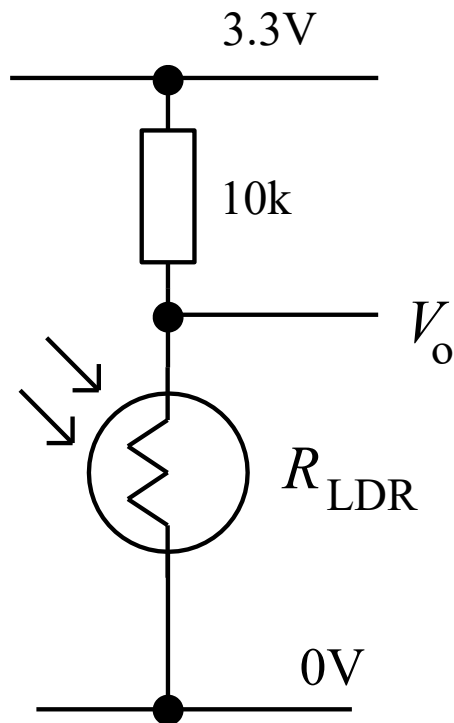
▼ WatchDog

Enable Analog WatchDog Mode ☐

Potentiometer as Voltage Divider



Simple Analog Sensors : the Light Dependent Resistor



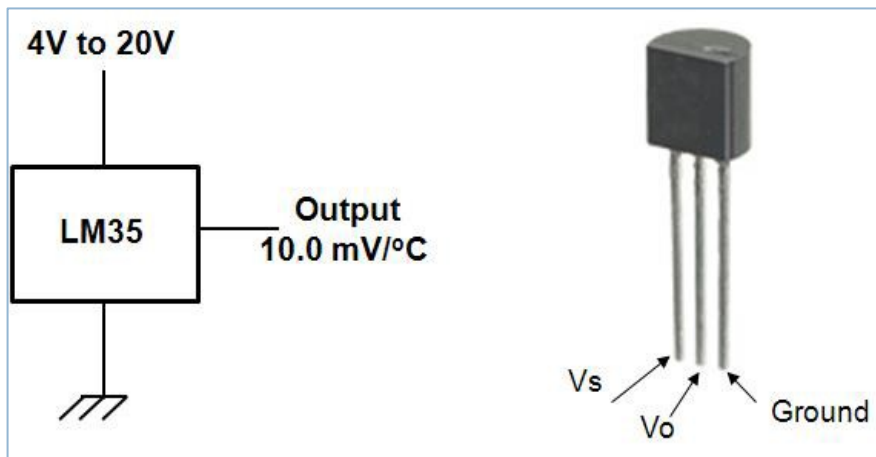
- LDR is made from semiconductor material.
- the brighter the light, the more electrons are released
- Released electrons are available to conduct electricity; the resistance falls
- Remove the light, electrons are back into their place; the resistance goes up again.



Illumination (lux)	$R_{LDR} (\Omega)$	V_o
Dark	$\geq 1.0 \text{ M}$	$\geq 3.27 \text{ V}$
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

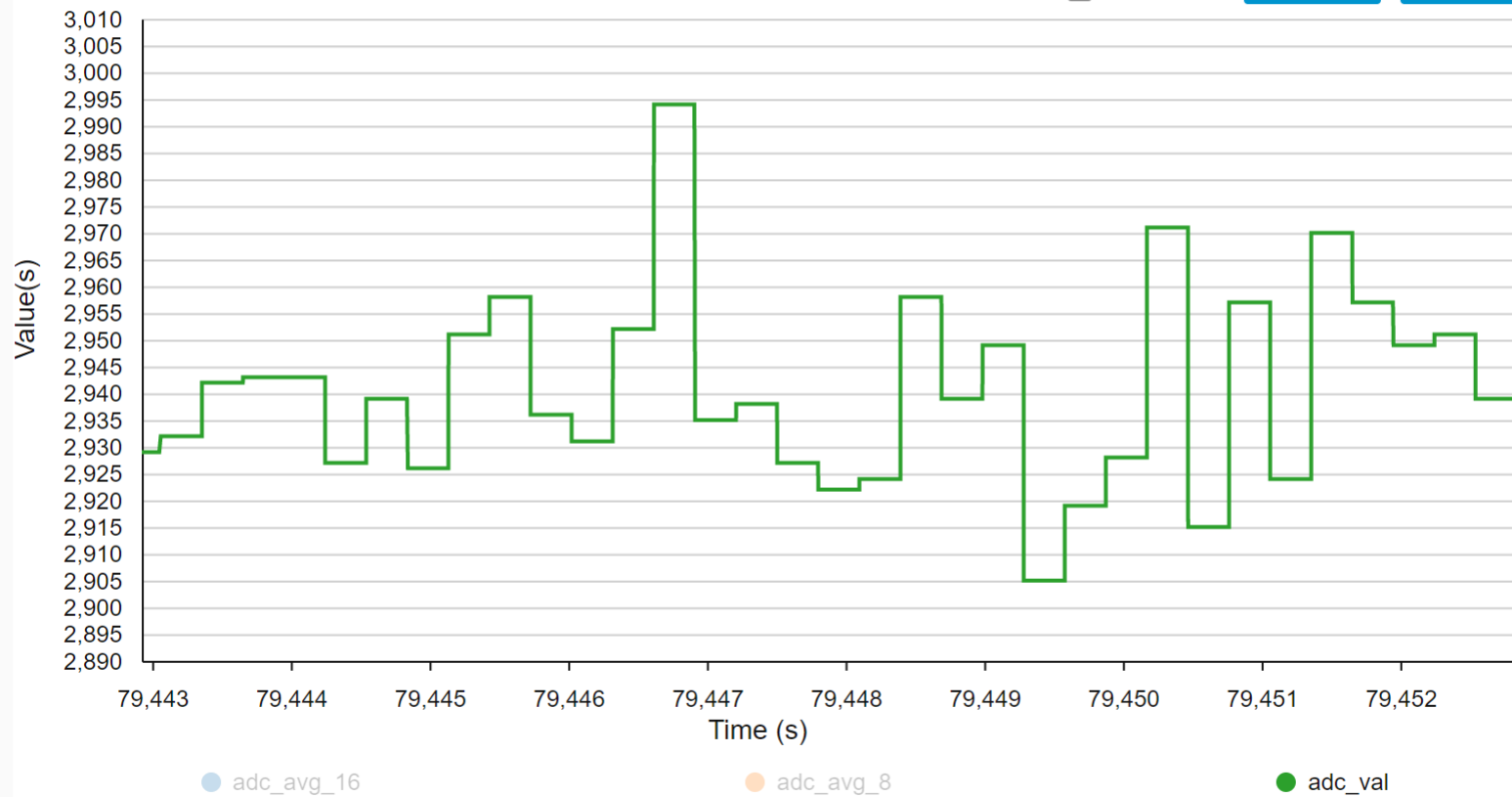
STOP ACQUISITION

CLEAR GRAPHS

☐ Show Points

Zoom

Show All



IMPORT DATA

Chart

START ACQUISITION

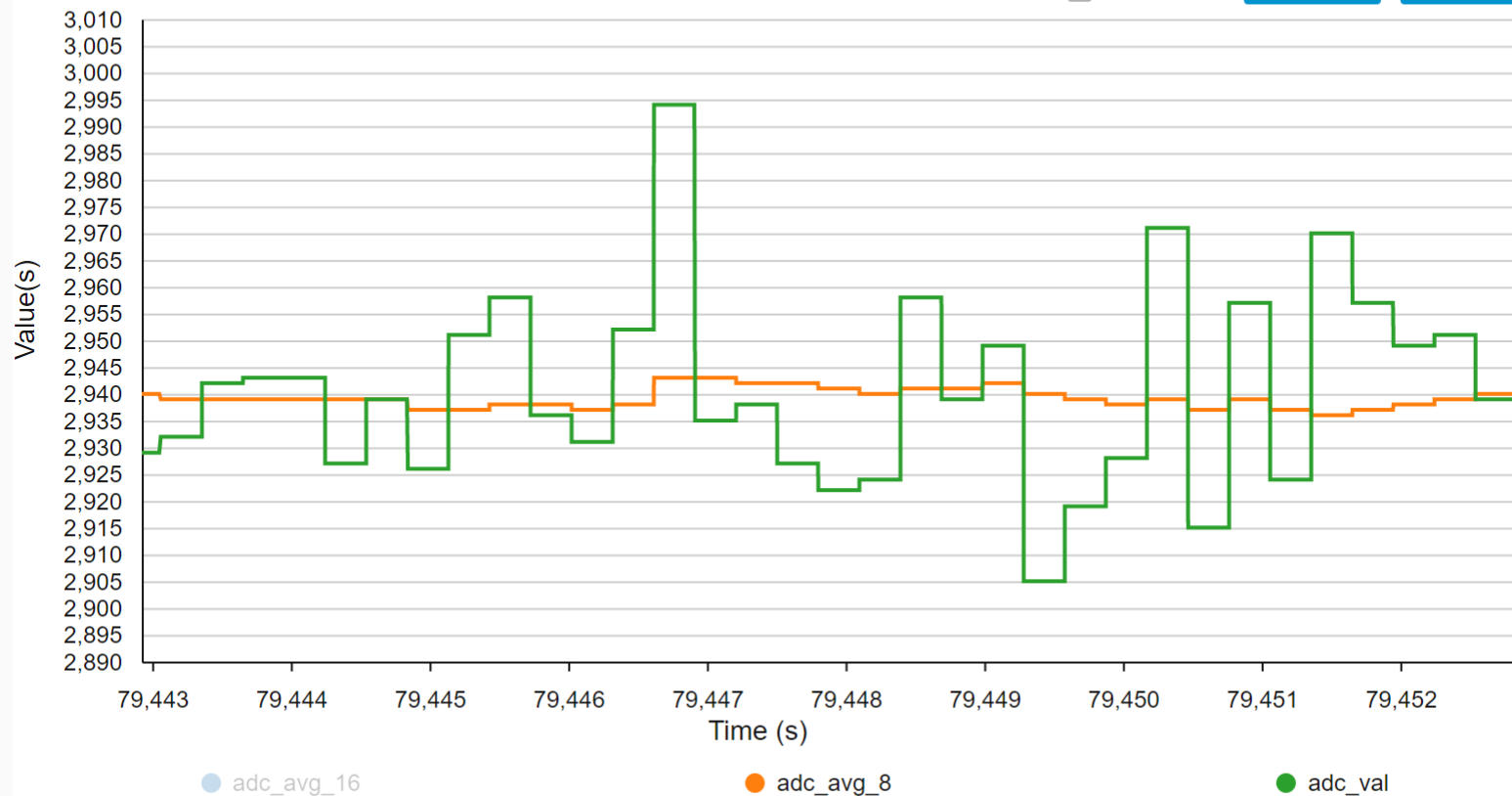
STOP ACQUISITION

CLEAR GRAPHS

☐ Show Points

Zoom

Show All



IMPORT DATA

Chart

START ACQUISITION

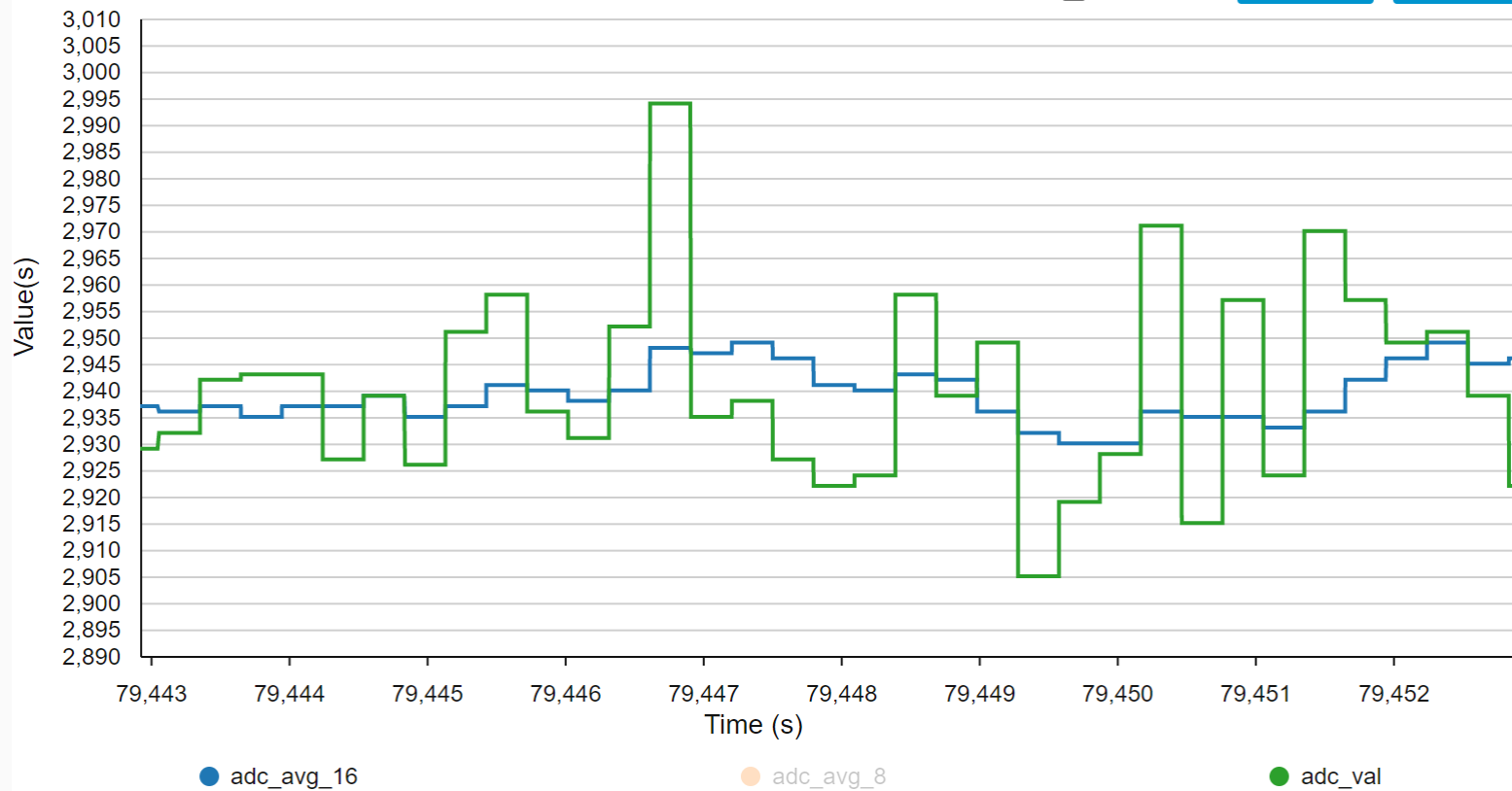
STOP ACQUISITION

CLEAR GRAPHS

☐ Show Points

Zoom

Show All



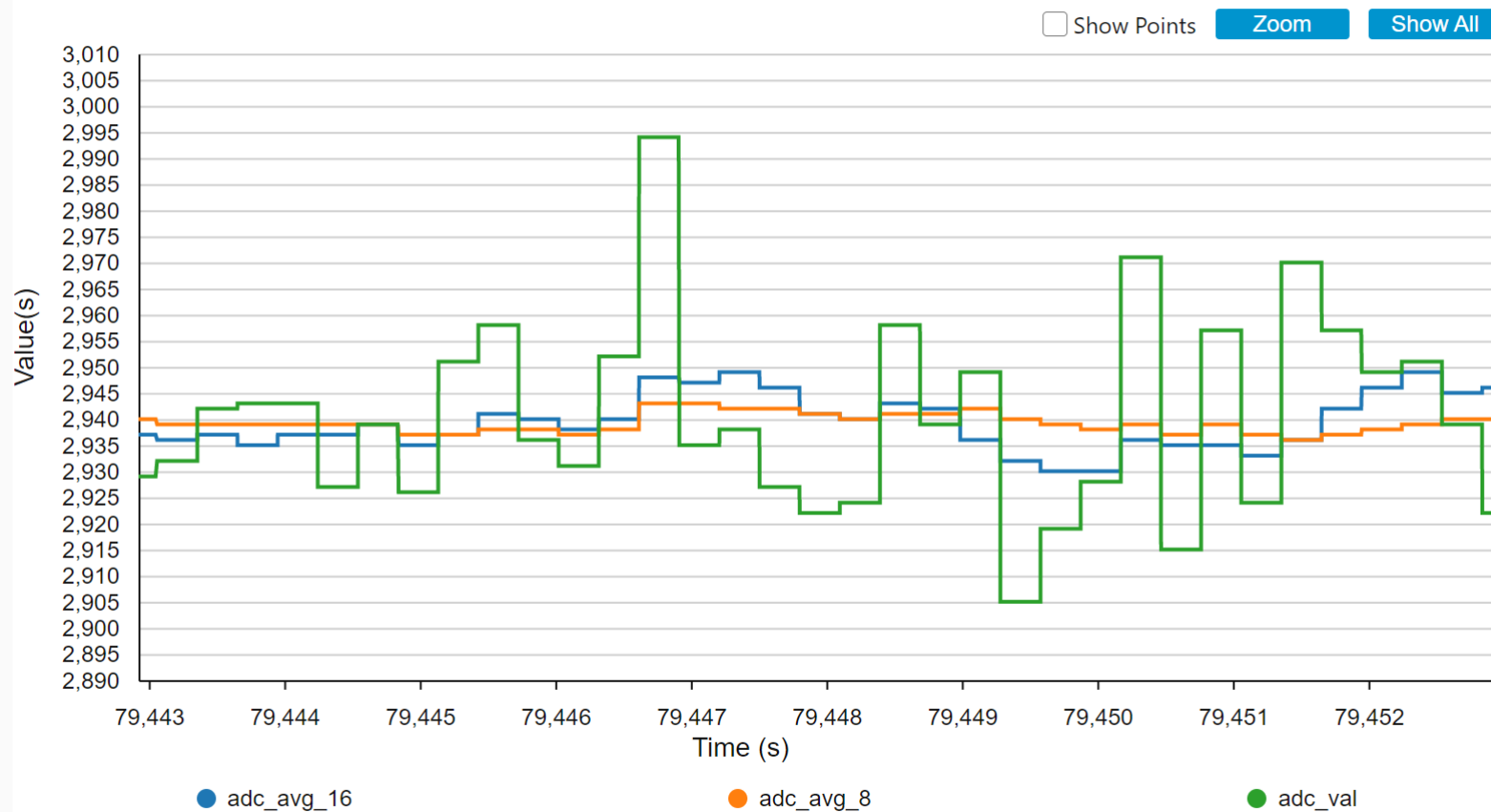
IMPORT DATA

Chart

START ACQUISITION

STOP ACQUISITION

CLEAR GRAPHS



IMPORT DATA

Chart

START ACQUISITION

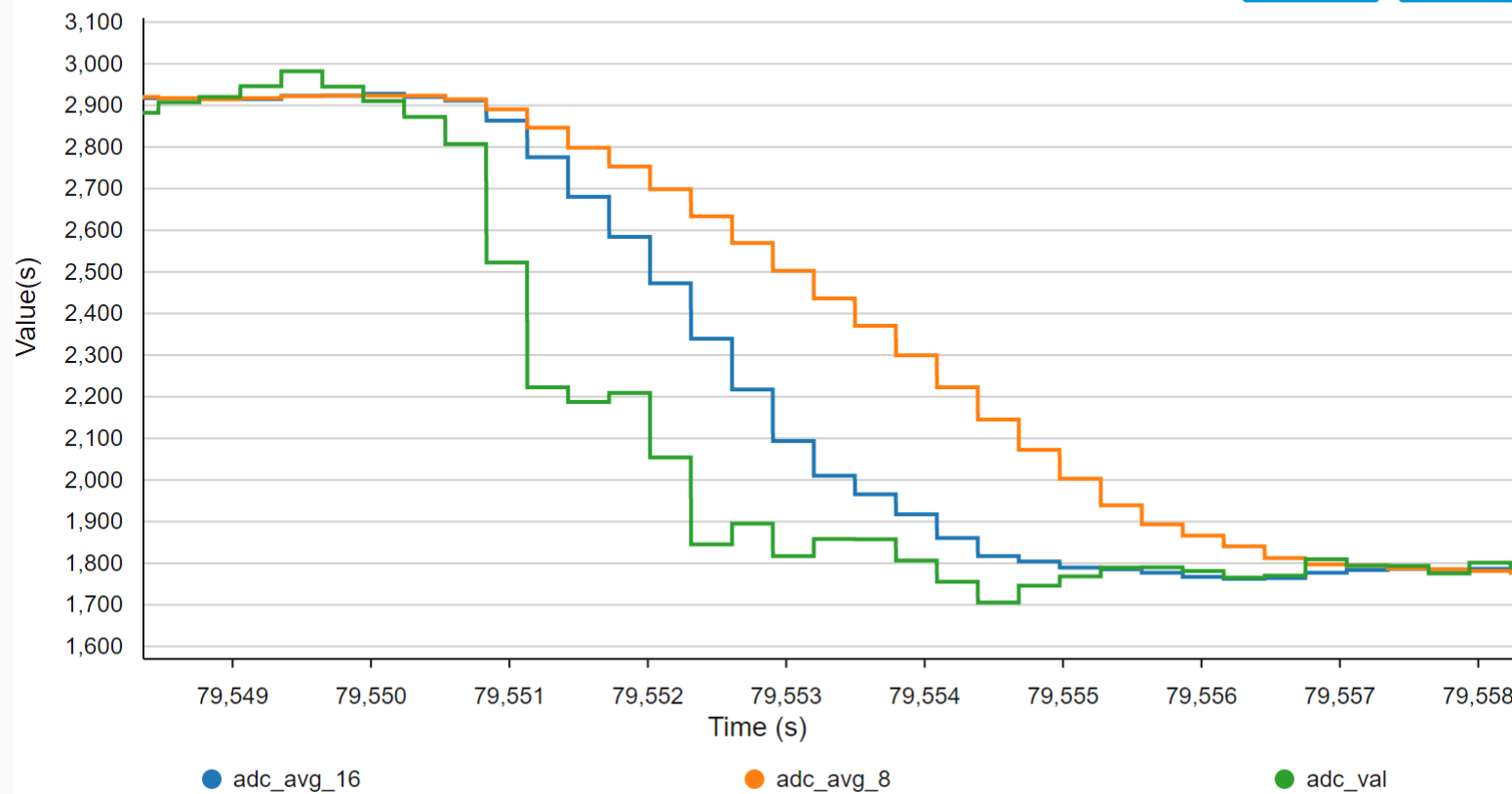
STOP ACQUISITION

CLEAR GRAPHS

☐ Show Points

Zoom

Show All



IMPORT DATA

Moving Average

```
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;  
}
```

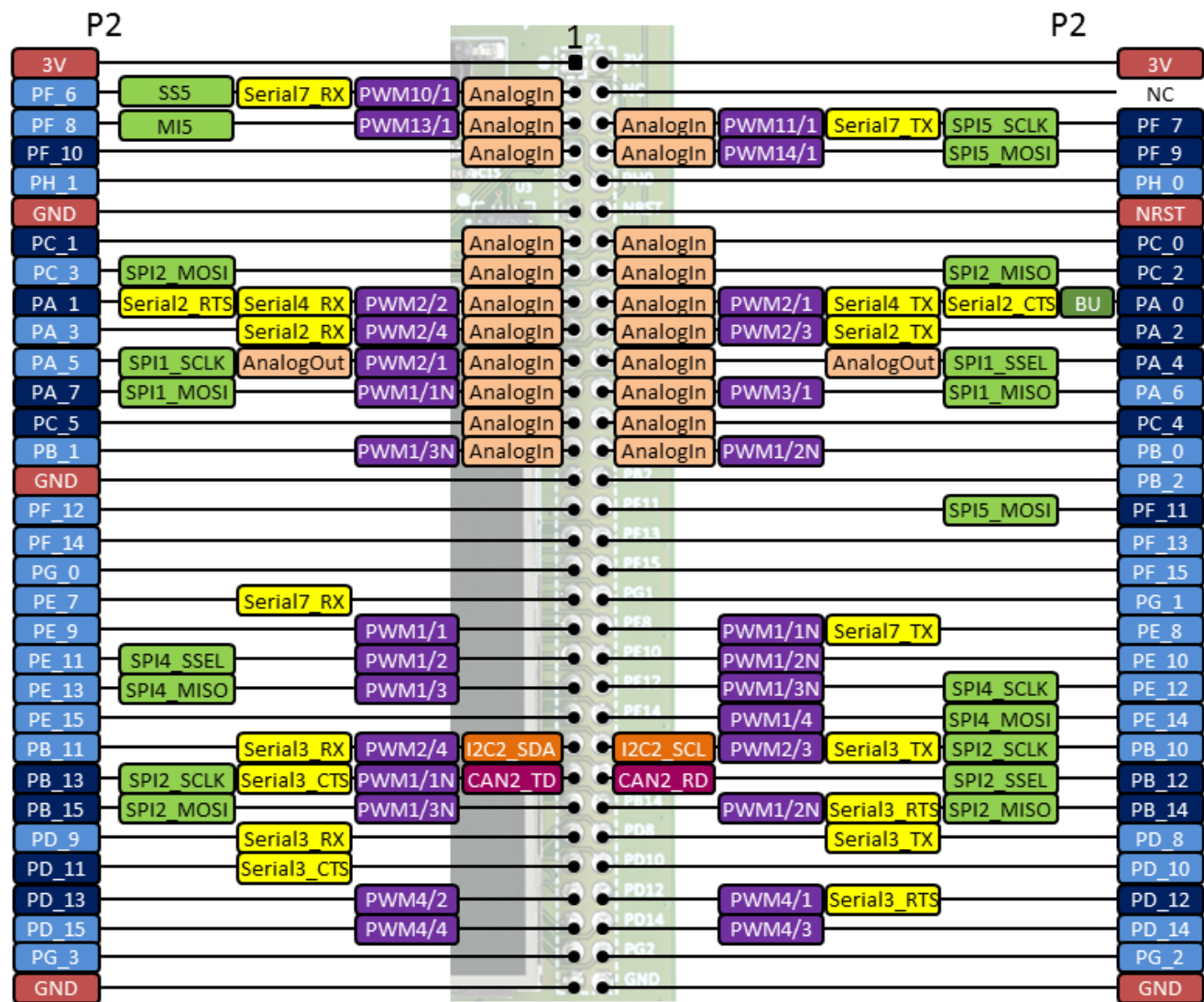
```
int average_16(int x) {  
    static int samples[8];  
    static int i = 0;  
    static int total = 0;  
  
    /* Update the moving average */  
    total += x - samples[i];  
    samples[i] = x;  
  
    /* Update the index */  
    i = (i==7 ? 0 : i+1);  
  
    return total>>3;  
}
```



life.augmented

DISCO-F429ZI

P2 HEADER
(top left side)



ADC1_IN13

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.